摘要

当前，随着深亚微米技术的发展，嵌入式 SRAM 将更多的应用于 SOC 系统中，并且在整个 SOC 中占据更多的面积。由于嵌入式 SRAM 的内嵌性，它的管脚并没有引到外部引脚上，并且由于外部测试会由于测试环境的复杂，很难做到全速测试，因此使用存储器内建自测试（memoryBIST）系统进行可靠性分析显得尤为重要。

作为本论文研究内容的存储器内建自测试（memoryBIST）芯片设计，不仅可以作为 IP 核，嵌入到 SRAM 中实现内部全速自动测试，而且可以单独流片，独立成为专门测试 SRAM 的测试芯片。本论文所设计研究的内容对当前和今后存储器有效和高速测试的发展都有着非常大的实际应用价值和广泛的市场前景。

本论文完成了存储器内建自测试（memoryBIST）芯片的 ASIC 设计与实现，该芯片是针对 4 颗不同容量的双端口 SRAM 测试芯片的设计，同时支持 March C+，March D2PF 两种主流存储器测试算法，同时将 March 算法扩展为字定向算法，在保证测试的故障覆盖率的同时，提高了测试效率，节约了测试的时间成本；在时序方面采用了并行处理的 PipeLine 结构，理论分析表明这种结构在减少测试时间方面是有效的，实现了全速测试；支持在高速内部测试的同时将内部的错误地址和错误数据信息以慢速向外输出，有效降低了高速测试条件下芯片的调试难度。

关键词：嵌入式 SRAM　MBIST　SRAM 故障模型　March 算法

Abstract

Nowadays, with the deep sub-micron process technology development, there will be more and more embedded SRAM in SOC, and taking up more and more area. because of its embedded characteristic, the embedded SRAM have no pin linking to the output port, and also hard to do full-speed test due to the complex exterior environment. So, it is very important and useful to impose MBIST system on memory DFT(Design For Test).

The design of memoryBIST for testing SRAM will not only work as IP core, which is embeded into SRAM in order to implement full-speed test, bus also can be work as an independent chip for testing SRAM. So the project in this thesis has very large application value and great market foreground in the present and the future.

This thesis is a ASIC design of memoryBIST, based on four two-port SRAMs with defferent cell density. The chip support March C+ and March D2PF memory testing algorithm. At the same time, we extend the March algorithm to a word oriented algorithm, which enhance the test efficiency and save the test cost under assuring the fault coverage; The Sequential design uses a PipeLine processing approach, because theoretical analysis shows that the approach can reduce the testing time effectively, achieving the full-speed testing. At the same time, it also support output the interior wrong address and wrong data at a comparative low speed, then decrease the diffculty of testing the chip at full-speed.

Key words: Embedded SRAM MBIST SRAM Fault Model March algorithm

南开大学学位论文使用授权书

根据《南开大学关于研究生学位论文收藏和利用管理办法》，我校的博士、硕士学位获得者均须向南开大学提交本人的学位论文纸质本及相应电子版。

本人完全了解南开大学有关研究生学位论文收藏和利用的管理规定。南开大学拥有在《著作权法》规定范围内的学位论文使用权，即：(1)学位获得者必须按规定提交学位论文(包括纸质印刷本及电子版)，学校可以采用影印、缩印或其他复制手段保存研究生学位论文，并编入《南开大学博硕士学位论文全文数据库》；(2)为教学和科研目的，学校可以将公开的学位论文作为资料在图书馆等场所提供校内师生阅读，在校园网上提供论文目录检索、文摘以及论文全文浏览、下载等免费信息服务；(3)根据教育部有关规定，南开大学向教育部指定单位提交公开的学位论文；(4)学位论文作者授权学校向中国科技信息研究所和中国学术期刊(光盘)电子出版社提交规定范围的学位论文及其电子版并收入相应学位论文数据库，通过其相关网站对外进行信息服务。同时本人保留在其他媒体发表论文的权利。

非公开学位论文，保密期限内不向外提交和提供服务，解密后提交和服务同公开论文。

论文电子版提交至校图书馆网站：http://202.113.20.161:8001/index.htm。

本人承诺：本人的学位论文是在南开大学学习期间创作完成的作品，并已通过论文答辩；提交的学位论文电子版与纸质本论文的内容一致，如因不同造成不良后果由本人自负。

本人同意遵守上述规定。本授权书签署一式两份，由研究生院和图书馆留存。

作者暨授权人签字：_____

20　年　月　日

南开大学研究生学位论文作者信息

论文题目					
姓　名		学号		答辩日期	年　月　日
论文类别	博士 □　学历硕士□　硕士专业学位□　高校教师□　同等学力硕士□				
院/系/所		专　业			
联系电话		Email			
通信地址(邮编)：					
备注：				是否批准为非公开论文	

注：本授权书适用我校授予的所有博士、硕士的学位论文。由作者填写(一式两份)签字后交校图书馆，非公开学位论文须附《南开大学研究生申请非公开学位论文审批表》。

南开大学学位论文原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师指导下，进行研究工作所取得的成果。除文中已经注明引用的内容外，本学位论文的研究成果不包含任何他人创作的、已公开发表或者没有公开发表的作品的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明。本学位论文原创性声明的法律责任由本人承担。

学位论文作者签名：

年　　月　　日

第一章 引言

本章主要介绍存储器的发展历史，并简要介绍了国内外 memoryBIST 的研究状况和发展方向，最后简要介绍本文所做的工作。

第一节 存储器发展简介

自从 1971 年底第一块 DRAM 的诞生，存储器已有近 40 年的历史。美国最开始成为存储器生产大国，市场占有率一度高达 91%。美国存储器设计与制造公司代表：英特尔，美光半导体。二十世纪 70 年代后期，日本凭借其大规模，低成本的生产优势，逐渐取得存储器生产领域的领先地位。日本存储器设计与制造公司代表：日立，NEC，富士通。二十世纪 80 年代的韩国，步日本的后尘，大量引进美国，日本的先进技术和生产设备，以其低廉的劳动力价格优势，成为存储器生产，销售大国。韩国存储器设计与制造公司代表为：三星半导体，现代半导体。德国也是生产和销售存储器主要国家之一，其公司代表为英飞凌科技。随着半导体工艺的不断等比例缩小，许多片外器件不断被集成到系统芯片（以下简称 SOC）中去。存储器由于其广泛的应用成为不可或缺的重要一员，存储器在 SOC 中的比例（面积）不断上升。到 2010 年，约 90%的硅片面积都被不同功能的存储器所占据[1]。存储器的种类很多，半导体存储器一般包括 Mask ROM，EPROM，EEPROM，SRAM，DRAM 和 Flash 等。目前，国际上存储器生产工艺已达 22 nm 的制程。

第二节 国内外 MBIST 研究情况

固定测试向量的 BIST 在当今的存储器测试领域中占据主导地位。固定的测试向量意味着测试向量按照预先设计好的算法生成（例如 March 算法）。对于 BIST 设计来说，主要采用的有两种技术：基于状态机的硬件技术和基于微代码的软件技术[2]。

状态机 BIST 根据设计的不同，可以生成一个单一，或者复杂的测试向量。

这种 BIST 通常在 ASIC 领域中都是使用特定软件（例如 Mentor 公司的 MBISTArchitect）生成基于某一算法的 rtl 代码，然后随硬件代码一并综合，嵌在芯片当中一同流片。然而，一个好的存储器测试方法需要一系列的测试算法来达到理想的芯片测试覆盖率，这就造成了状态机设计的复杂性；同时，这种软件生成基于状态机的 BIST，通常会同时生成几个，甚至几百个锁存器，这就有可能破坏整个 ASIC 设计的时序，使得我们的设计难以优化，甚至难以实现。

基于微代码的 BIST 是一种可编程的 BIST，是将软件指令输入 CPU 中，然后由 CPU 发出指令，在系统测试过程中生成相应的测试向量，随着工艺的发展而引入新错误的模型，在软件上也更加容易实现，这就使得基于微代码的 BIST 具有很大程度的灵活性。然而，这种 BIST 是依托于 CPU 的软件实现方法，它也有它自身的缺陷，首先，它要求硬件中必须有 CPU 这种强大的运算单元来维持 BIST 算法的向量生成，如果在系统中没有支持可编程的 CPU，那么这种 BIST 将是无法实现的，其次，为了支持 BIST 算法，在 SOC 设计中的 CPU 部分的设计难度又会增加，直接导致了整个设计的难度增加。

对于 BIST 设计来说，除了要减小引入 BIST 之后导致的存储器性能下降之外，一个很重要的标准就是如何减小引入 BIST 之后所带来的面积和引脚的消耗。由于 SOC 设计中存储器的面积在不断的增加，而且类型，容量，时序等方面都会不同，当前 BIST 技术面临的问题主要在于：如何减小 BIST 所占用的额外硅片面积，如何支持 BIST 自动诊断的能力，如何减小测试功耗的消耗，如何支持各种各样的存储器（如单端口，双端口存储器）。

第三节 本论文研究的内容与意义

为了不断适应更大数据处理量的需求，今天的片上系统（SOC）中使用的嵌入式存储器的容量越来越大，大部分芯片面积正在由逻辑部分主导转向存储器器件主导转变。除此之外，当今复杂的 SOC 设计大多都使用内嵌式存储器，而且规模庞大（256Mbits 或者更多），因此， SOC 芯片的良率将大大取决于这些内嵌存储器的良率，由于存储器的良率会随着存储器容量的增加而减小，除非我们加入特殊的设计和工艺，用以保证存储器的良率，否则整个 SOC 芯片的良率将变得不可接受。为了达到一个良好的芯片产品良率，内嵌式存储器还必须具备自动修复的能力，这样，内建自测试系统 BIST（Build-In-Self-Test），内建

自分析系统 BISA（Build-In-Self-Analysis），和内建自修复系统 BISR（Buil-In-Self-Repair）就有了存在的必要性和重要性。

本文主要介绍了基于 SRAM 的 BIST 技术（memoryBIST），本课题是苏州秉亮科技有限公司 SRAM 设计组的研发子项目，目的在于实现嵌入式双口 SRAM 内建自测试设计的新方法。本设计完成了存储器内建自测试（Memory BIST）芯片的 ASIC 设计与实现，属于前文提到的基于状态机的 BIST 设计，该芯片是针对一款 4 颗不同容量双端口 SRAM 测试芯片的设计，同时支持 March C+，March D2PF 多种主流存储器测试算法，支持将出错信息保存，并在较慢时钟频率下向外输出错误信息，方便流片后的 FPGA 片上测试中使用逻辑分析仪抓取错误信息。

第二章 集成电路可测性设计

memoryBIST 是一种结构性集成电路可测性设计技术（DFT，Design For Test），本章中详细介绍了 DFT 的基本概念，常用的 DFT 方法及其适用场合，分析了系统芯片可测试性设计所遇到的挑战。

第一节 集成电路测试和设计可测性基础

2.1.1 可测性设计目的和重要性

VLSI芯片是通过一系列的处理步骤制造的，这些步骤涉及光学、冶金学和光学等一系列复杂的工艺，芯片在这些过程中可能产生物理缺陷，导致芯片不能正常工作。因此对芯片进行测试成为芯片设计、生产的过程中一个必不可少的环节。

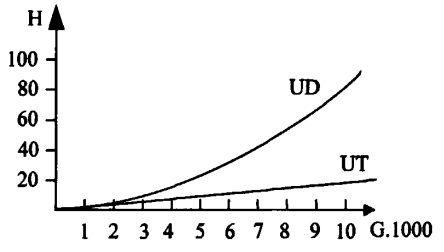
可测性设计是在1970年在CherryHill测试会议上提出的，然而可测性设计的必要性直至上个世纪70年代中期随着集成电路设计的发展才逐渐被人们认识。随后关于可测性设计设计方面的论文和研究成果越来越多，目前在一些重要的国际会议上，如国际测试会议（ITC），国际设计自动化会议（DAC）等都有专门的分组会。此外，一些可测性设计的规则已经成为集成电路设计的工业标准，如IEEE1149.1标准等。可测性设计已经成为集成电路设计领域一个极其重要的组成部分。

电路的可测性与产品的质量可靠性息息相关。产品成品率（Y）、故障级（DL）、故障覆盖率（T）的关系为式2-1

$$DL = 1 - Y^{(1-T)} \qquad (2\text{-}1)$$

如果要求故障级达到0.1%，在故障覆盖率为90%的情况下，成品率必须到99.1%，而实际的成品率几乎不可能达到90%，因此只有提高故障覆盖率才能降低故障级，减少劣质产品流入市场的概率，特别是在成品率比较低的况下，高故障率的测试可以弥补成品率低的缺陷。

综上所述，SOC时代的到来，芯片测试问题变得越来越重要。为了达到所需的故障覆盖率，同时又减小测试开销，人们逐渐把注意力集中到电路设计方面，对电路进行改动设计，使之容易测试。这种在设计过程中考虑可测性的设计方法称为可测性设计。图2.1显示了采用无约束设计和采用可测性设计后的测试开销的关系[9]。



H—测试开销 D—门数 UD—无约束设计 UT—可测性设计

图 2.1 测试费用与电路规模的关系

从由图 2.1 可以看出：对于无约束设计，测试开销随着电路规模的增大呈指数上升，而采用了可测性设计之后，测试开销与电路规模基本呈线性增长关系。因此，对于 VLSI，可测性设计是必不可少的。

2.1.2 测试类型

可测性设计的方法主要可分成两大类：一类是专项设计（Ad hoc Design 功能点测试），即按功能基本要求设计系统和电路，采取一些比较简单易行的措施，使它们的可测性得到提高；另一类是结构设计（Structured Design），它是根据可测性设计的一般规则和基本模式来进行电路的功能设计。专项设计方法主要针对组合逻辑电路的测试，而数字系统中故障诊断的困难往往是时序电路的测试。时序电路比组合电路更加难于测试的主要原因有：

1. 时序电路中存在着反馈线，而对反馈线的处理是比较困难的。

2. 由于时序电路中存在着存储元件，因此电路中存在着状态变量的初态问题，在没有总清零或复位的条件下，这些状态变量的初态是随机的，必须寻找一个复位序列使这些状态变量转移至已知的确定状态。

3. 时序元件，尤其是异步时序元件，对竞争现象是异常敏感的。因此其产生的测试序列，不仅在逻辑功能上要满足测试要求，而且要考虑到竞争态对测试过程的影响。为了简化时序电路的测试向量生成的复杂程度，提高故障覆盖率，需要提高对时序电路的内部状态的控制和观察能力，增加可控性和可观性，因此提出了基于结构设计的可测性设计方法。

所谓结构设计方法就是从可测性的观点对电路的结构提出一定的设计规则，使得设计的电路容易测试，主要有扫描设计（Scan Design），边界扫描设计（Boundary- Scan），内建自测试设计（Built-In Self-Test-BIST）等[3,4,5]。本文所采用的内建自测试设计就是属于结构性设计方法。

第二节 可测性设计方法

可测试性技术的方法可分为功能点测试、基于扫描技术的结构化测试和内建自测试。

2.2.1 功能点测试

功能点测试技术可用于特殊电路和单元的测试。它是针对一个已经定型的电路设计中的测试问题而提出的。该技术有分块、增加测试点、利用总线结构等几种主要方法。

分块法采用的技术有机械式分割、跳线和选通门等。机械式分割是将整个电路分割为多块。这样虽然使得测试生成故障模拟的工作量减少，但是却不利于系统的集成，费用也大大增加。采用跳线的方法则会引入大量的 I/O 端口。而选通门的方法则需要在设计中引入大量的输入、输出端口以及完成选通功能所必须的模块。

增加测试点是提高电路可测试性最直接的方法。其基本方法是将电路内难于测试的节点引出，作为测试点，如果测试点直接用作系统的原始输入，则可以提高该电路节点的可控性，如果测试点用作系统的原始输出，则可以提高电路的可观察性。该方法的缺点是由于引脚数目的限制，所能引入的测试点数目非常有限。

利用总线结构类似于分块法。它将电路分成若干个功能块，并且与总线相

连，可以通过总线测试各个功能模块，改进各功能模块的可测试性。这种方法的缺点在于不能检测总线自身的故障。

功能点测试技术的缺点在于它不能解决成品电路的测试筛选生成问题，只能用来辅助分析测试；另外，它需要在电路中每个测试点增加可控的输入端和可观察的输出端，因此而增加了附加的连线与 I/O 端口，给后端的布局布线带来了较多的麻烦，也使得芯片面积的开销较大。

2.2.2 扫描测试

结构化 DFT 技术对电路结构进行总体上的考虑，只增加了用于测试的内部逻辑电路，就可以访问芯片内部电路节点，按照一定的 DFT 规则进行测试电路设计，具有通用性好和自动化程度高的特点。

扫描设计是一种应用最为广泛的可测性设计技术，是主要的时序电路的可测试设计方法。测试时能够获得高达100%的故障覆盖率。扫描设计是通过将电路中的时序元件转化成为可控制和可观测的单元，再把这些时序元件连接成一个或多个移位寄存器（又称扫描链）。测试时扫描链可以通过扫描输入端将其置成特定状态并通过扫描输出端将其中的内容移出观察，测试数据在扫描链上时串行移动的。假设电路中的时序元件是由图2.2(a)所示的D触发器组成，2.2(b)则为一个在D触发器的基础上设计的具有扫描功能的触发器。

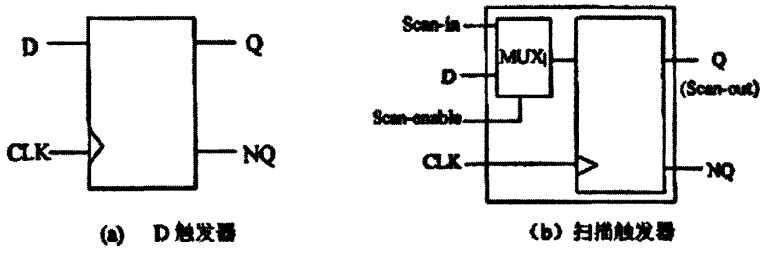


图 2.2 扫描触发器的结构

从图2.2(b)中可知扫描触发器主要是在原触发器的D输入端增加了一个多路选择器，通过扫描控制信号（Scan-enable）来选择触发器的输入数据是正常工作时的输入信号（D）还是测试扫描数据（Scan-in）。

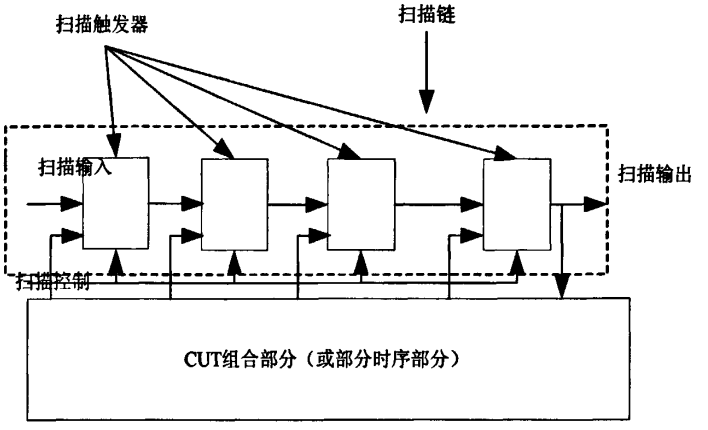


图 2.3 扫描设计的基本结构

扫描设计就是利用经过变化的扫描触发器连接成一个或多个移位寄存器，即扫描链。图2.3为扫描设计的基本结构。这样的设计可以把复杂的时序电路的测试向量生成转化为组合电路（全扫描设计）或部分时序电路（部分扫描设计），明显的降低了测试向量生成的复杂度。

由上可知，扫描技术是指通过将电路中任一节点的状态移进或移出来进行测试定位的手段，其特点是测试数据的串行化。通过将系统内的寄存器等时序元件重新设计，使其具有可扫描性，测试数据从芯片端口经移位寄存器等组成的数据通路串行移动，并在数据输出端对数据进行分析，以此来提高电路内部节点的可控制性和可观察性，达到测试芯片内部节点的目的。扫描技术分为全扫描技术、部分扫描技术和边界扫描技术。

所谓全扫描技术就是将电路中所有的触发器用可扫描触发器替代，使得所有的触发器在测试的时候链接成一个移位寄存器链，称为扫描链。这样，电路在测试时就可以分成纯组合逻辑的测试和移位寄存器链的测试。电路中所有的状态可以直接从原始输入和输出端得到控制和观察。全扫描技术可以显著的减少测试生成的复杂度和测试费用，但这是以牺牲芯片面积和降低系统速度为代价的。

部分扫描的方法是只选择需要观察的关键路径上的一部分触发器构成扫描链，降低了扫描设计的芯片面积开销，减少了测试时间。其关键技术在于如何选择触发器。对部分扫描技术的研究主要在于如何减少芯片面积、降低对电路

性能的影响，提高电路的故障覆盖率和减小测试矢量生成的复杂度等方面。

边界扫描技术是各 IC 制造商支持和遵守的一种扫描技术标准，起先主要用于对印刷电路板的测试，它提供一个标准的测试接口简化了印刷电路板的焊接质量测试。它是在 IC 的输入输出端口处放置边界扫描单元，并把这些扫描单元依次连成扫描链，然后运用扫描测试原理观察并控制芯片边界的信号。边界扫描技术也可用于对系统芯片进行故障检测，但是由于这种测试观测方法要将所有的并行输入/输出数据串行化，测试时间相当长，因此这种方法目前一般用于对板级系统的互连测试与电路板之间的互连测试。

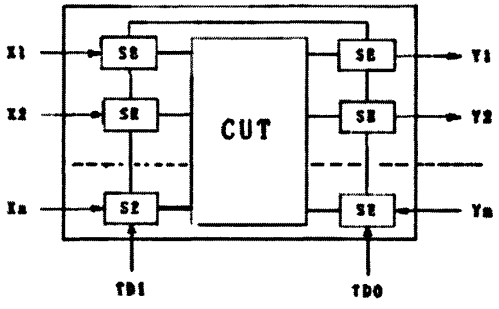


图 2.4 具有边界扫描结构的 VLSI

边界扫描的基本原理是在靠近待测器件的每一个输入渐出管脚处增加一个边界扫描单元，并把这些单元连接成扫描链，运用扫描测试原理观察并控制待测器件边界的信号。在图2.4中，与输入节点X_1、X_2、X_n和输出节点Y_1、Y_2、……、Y_m连接的SE即为边界扫描单元，它们构成一条扫描链（称为边界扫描寄存器—BSR），其输入为TDI（Test Data Input），输出为TDO（Test Data Out）。在测试时由BSR串行的存储和读出测试数据。此外，还需要个测试控制信号：选择TMS（Test Mode Select）和测试时钟TCK（Test Clock）来控制测试方式的选择。

边界扫描技术降低了对测试系统的要求，可实现多层次、全面的测试，但实现边界扫描技术需要附加一定的芯片面积，同时增加了连线数目，且工作速度有所下降。边界扫描技术是一种扩展的自治测试技术，它在测试时不需要其它的测试设备，不仅可以测试芯片或PCB板的逻辑功能，还可以测试IC之间或PCB板之间的连接是否存在故障。

2.2.3 内建自测试

内建自测试（BIST Build-In-Self-Test）[30]技术对数字电路进行测试的过程可分为两个步骤：首先将测试信号发生器产生的测试序列施加到被测电路，然后由输出响应分析器检查被测电路的输出序列，以确定电路是否存在故障以及故障的位置。

BIST 主要完成测试序列生成和输出响应分析两个任务。通过分析被测电路的响应输出，判断被测电路是否存在故障。因此，对数字电路进行 BIST 测试，需要增加三个硬件部分：测试序列生成器、输出响应分析器和测试控制部分。

在测试序列生成器中，有确定性生成、伪穷举测试生成和伪随机测试生成等几种方法。

确定性测试方法是一种针对特定的电路故障进行测试的方法，虽然可以得到很高的故障覆盖率，但硬件开销大，仅在测试码个数较少的时候采用。

伪穷举测试的方法是把所有可能输入都加以计算的测试方法。它的最大特点是故障覆盖率可以达到 100%，但其计算量与输入端子呈幂次方关系，因此计算量很大。如果将电路分为多个原始输入变量互相独立的块，则测试数将大大减少。伪穷举法就是这样一种压缩测试向量的方法。伪穷举法也具有很高的故障覆盖率，但伪穷举法对电路进行划分比较困难，有相当的局限性。而且由于加入了附加硬件，可能对电路性能产生负面效应。

伪随机测试是一种广泛使用的测试方法，该方法可以对被测试电路产生大量的测试代码，而且硬件电路开销较小，同时具有较高的故障覆盖率。LFSR（Linear Feedback Shift Register，线性反馈移位寄存器）就是这样一种测试代码生成电路。

实现输出响应分析的方法有 ROM 比较逻辑法、多输入特征寄存器法和跳变计数器法等。ROM 比较逻辑法是将正确的响应存储在芯片内的 ROM 中，在测试的时候，将其与测试响应进行比较，但这种方法会因为占用太多的芯片面积而毫无实用价值。多输入特征寄存器方法是将被测试电路中各节点的响应序列进行处理，得到与测试响应序列等长的特征字（Signature），然后与无故障电路节点的响应序列特征值进行比较，如果两者相同，则说明电路正常，否则表明被测试电路有故障存在。跳变计数器法是通过比较输出响应的跳变总数，来判断被测试电路是否正常工作，因此需要存储和比较跳变次数，从而使得所需要

的存储空间与测试时间都得到大幅度的降低。但是后面两种方法是以牺牲故障覆盖率为代价的。

实现 DFT 的工具应该首推 Mentor 公司。Fastscan 可以用于全扫描逻辑电路的测试；Flextest 则可以用于解决部分扫描设计问题；LBISTArchitect 则用来生成逻辑电路的 BIST 部分，适用于 IP 或宏模块的内建自测试设计；MBISTArchitect 可以用来实现存储器的 BIST；BSDArchitect 可以用来生成边界扫描电路。

Synopsys公司也有自己的DFT实现工具：DFT Compiler用来完成可测试性设计综合；TetraMAX用来生成ATPG（Auto Test Pattern Generation）测试向量；VERA Developers Kit则是测试平台开发和测试向量自动生成工具。

第三节 存储器测试结构和技术

2.3.1 存储器内建自测试

今天的设计已经普遍含有 50%的嵌入式存储器，且这部分的比例预计在未来几年中还会加大。很明显，为实现全面的系统级芯片（SOC）测试，必须制定一种高质量的存储器测试策略。存储器紧凑的结构特征使其更容易受到各类缺陷的影响。存储器阵列工作模式本质上主要是模拟的，来自存储器件的弱信号被放大到适当的驱动强度，且存储器单元的信号传输只涉及到很少的电荷。所有这些设计特点都使存储器阵列更容易受到错综复杂的制造缺陷的影响。而紧密的存储器阵列封装造成了这样一种情况，即相邻单元的状态在存在缺陷的情况下可能会发生误操作，因此某些缺陷可能只在特定的数据模式下才会暴露。此外，这些缺陷类型很多是具有时间相关性的，因此只有在正常工作频率下才会被发现。存储器内建自测试（memoryBIST）是 SOC 设计中用来测试嵌入式存储器的标准技术，它以合理的面积开销来对单个嵌入式存储器进行彻底的测试。最常见的存储器 BIST 类型包括可完成三项基本操作的有限状态机（FSM）：将测试模版（pattern）写入存储器、读回这些模版并将其与预期的结果进行比较。为对嵌入式存储器进行存取，memoryBIST 一般将测试多路复用器插入到地址、数据及控制线路中。

memoryBIST[5,6,7,8]技术通过将外部测试功能转移到芯片或安装芯片的封装上，使得人们不需要复杂、昂贵的测试设备，同时由于 BIST 与待测电路集成在

一块芯片上，使测试可按电路的正常工作速度、在多个层次上进行，提高了测试质量和测试速度。内建自测试电路设计是建立在伪随机数的产生、特征分析和扫描通路的基础上的。采用伪随机数发生器生成伪随机测试输入序列；应用特征分析器记录被测试电路输出序列（响应）的特征值；利用扫描通路设计，串行输出特征值。当测试所得的特征值与被测电路的正确特征值相同时，被测电路即为无故障，反之，则有故障。被测电路的正确特征值可预先通过完好电路的实测得到，也可以通过电路的功能模拟得到。由于伪随机数发生器、特征分析器和扫描通路设计所涉及的硬件比较简单，适当的设计可以共享逻辑电路，使得为测试而附加的电路比较少，容易把测试电路嵌入芯片内部，从而实现内建自测试电路设计。

2.3.2 存储器 BIST 组成

memoryBIST通常采用一种或多种算法为测试存储器一种或多种缺陷类型而特别设计，memoryBIST电路包括测试向量产生电路、BIST控制电路、响应分析器三部分。

1. 测试向量生成电路。

测试向量产生电路可生成多种测试向量，不同的测试算法实现的电路所产生的测试向量内容也不同。如何生成一个简单有效的测试向量是衡量BIST电路的好坏的关键因素。传统的测试向量生成策略是将要生成的向量存储在在线ROM中，测试时顺序将ROM中的测试向量输入到被测试电路中。这种测试方法不需要再去生成所需的测试向量，因此速度比较快，但是当数据量比较大的时候，ROM电路将占据大量的空间，使得额外的面积开销变得无法接受。然后人们又发明了穷举法来生成所需要的向量，穷举法穷举各种取值来组成要生成的测试向量，这种方法可以达到很高的故障覆盖率，但是穷举法也会生成很多重复的测试向量，同时随着测试输入端口的增加，测试时间又会变得很长，测试的成本也会很高。目前较常使用的是一种伪穷举法，将存储器电路划分成很多块后再分别采用穷举测试。但是，对存储器电路的划分比较困难，而且要引入附加的硬件，对电路的性能有影响。由于穷举法的种种缺陷，人们又引入了伪随机法。伪随机法的特点是：伪随机数发生器经过初始化后能自动地产生测试向量，其测试向量的数目与伪穷举法的测试向量差不多，仅仅是没有包含全

零的情况，而且硬件开销小，测试响应分析结果容易存储、分析。因此被人们广泛地采用。

2. BIST控制电路

BIST控制电路通常由状态机实现，控制BIST对存储器的读写操作。设计状态机按照测试算法所需要的步骤控制BIST电路对存储器进行有规律的读和写操作，同时控制输出电路的比较工作的时序。

3. 测试响应分析的策略

响应分析器既可以用比较器实现，也可以用压缩器多输入移位寄存器（MISR）电路实现，它对照已知正常的存储器响应，比较实际存储器模型响应并检测器件错误。

测试向量分析主要任务是分析输出的响应是否正确。最初，人们使用类似测试向量生成策略的方法，将正确的向量存储在在线 ROM 中，但是同样由于在线 ROM 会占据大量的面积，当测试向量比较多的时候，这种方法也会变得无法接受。同时，面对大量的向量测试，如果一个一个地去将输出响应向量与正确向量进行对比，也会浪费大量的测试时间，增加测试成本。由于存储器本身的特殊性，它测试输出不会像普通的数字逻辑电路那样可以预测输出，因此它的测试就无法使用数字测试设计中常使用的压缩技术分析测试响应，只能按照测试向量的原始数目去挨个做对比，但是由于测试向量生成电路、BIST 控制电路和测试响应分析设计所涉及的硬件比较简单，适当的设计可以共享逻辑电路，不需要单独为测试响应分析提供单独的正确响应数据，使得为测试而附加的电路比较少，容易把测试电路嵌入芯片内部，从而实现内测试电路的设计。

第三章 SRAM 工作原理及故障模型介绍

由于晶体管的密集（指数型增长）、布线高密度、高复杂度、时序更严格、频率更高、功能更复杂等客观因素，在实际生产过程中嵌入式存储器更易发生物理故障。例如：固定故障，耦合故障，转换故障，相邻模式敏感故障（这些故障在后文中将会讨论）。又由于物理缺陷的产生和影响过于复杂，且直接检测物理故障的难度非常大，所以有必要将物理缺陷转变为描述出错行为的故障模型，也就是将物理检测转变为功能测试。从而回避了物理缺陷检测的复杂度，提高了测试效率。

本章将简单介绍 SRAM 基本结构及工作原理，以及 memoryBIST 最关心的 SRAM 工作时序，并在此基础上分析了单端口和双端口存储器的故障模型，为后文选定 memoryBIST 所需的测试算法打下基础。

第一节 SRAM 电路基本介绍

3.1.1 SRAM 工作原理

SRAM有同步与异步之分，异步SRAM采用内部事件产生时钟信号来控制整个电路的工作，比较典型的是通过地址转换探测电路（ATD：address transition detection）电路来产生时钟信号。异步SRAM的功耗比较小，但时序比较复杂，难以控制，且读写速度较慢。而同步SRAM则采用统一的外部时钟信号来协调电路的工作，由于有统一的外部时钟，同步SRAM的功耗较小，时序也较为简单，速度较快。本次设计所涉及的SRAM就是四颗同步双端口SRAM。

SRAM读取速度是由地址取数时间来衡量，它是指从地址输入到数据读出的延迟时间，由从地址输入到数据输出关键路径上的延迟决定。优化关键路径上的延迟是提高SRAM性能的关键。如前文所述，memoryBIST有必要工作在SRAM的极限速度下，全速测试才可能重现更多的存储器内部故障。表3.1给出了SRAM读操作的关键路径。

表3.1 SRAM读操作的关键路径

预充电	行，列地址输入缓冲单元	行，列地址译码器	字线	位线	读敏感放大器	输出缓冲器

同步 SRAM（以下均指同步 SRAM）电路结构较为规整，总体结构如图 3.1 所示。整体电路可以划分为存储阵列，地址译码电路，数据输入输出缓冲电路，时序控制电路，敏感放大电路，列复用电路等六大部分。其中地址译码部分可分为行地址译码和列地址译码两组电路，在大容量分块存储体设计中，还需增加一个块地址译码电路来选择存储块。存储阵列由基本的存储单元在水平方向共享字线，在垂直共享位线排列而成。存储单元采用六管互补 CMOS 基本单元，实现单端口写入和单端口读出的方式。预充电电路在数据读出或写入之前，将位线充电到一个高电平值。敏感放大器在读操作时，将位线上小信号摆幅放大到标准的逻辑电平值，提高数据读出速度。随着工艺的不断等比例缩小，器件尺寸也在不断变小，极大的提高了 SRAM 的容量。而与此同时，在深亚微米工艺中，随着导线宽度 W 的减少，电阻会增大。其次，导线间距 S 变小，线间耦合作用非常显著。互连线延迟成为一个不可忽视的因素。为了提高数据存取速率，必须在电路结构上，做一定的优化。

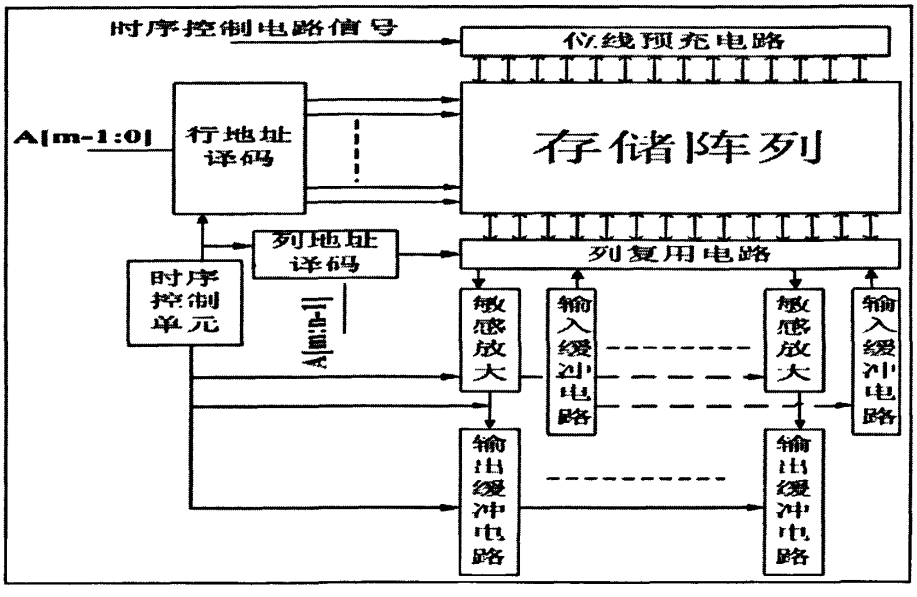


图 3.1 SRAM 总体架构

SRAM 外部引脚分别为地址总线，数据输入总线，数据输出总线，输出使能信号，时钟信号，片选信号，读写控制信号等。其外部接口比较简单，如图3.2 所示。其外部引脚如表 3.2 描述。

表 3.2 SRAM 外部引脚

名称	类型	描述
A[m-1:0]	输入	地址 （A[0]为低有效位）
DI[w-1:0]	输入	输入数据 （DI[0]为低有效位）
CK	输入	时钟信号
CS	输入	芯片使能（高有效）
WEB	输入	读/写使能信号（低为写使能信号）
OE	输入	输出使能信号
DO[w-1:0]	输出	输出数据 （DO[0]为低有效位）

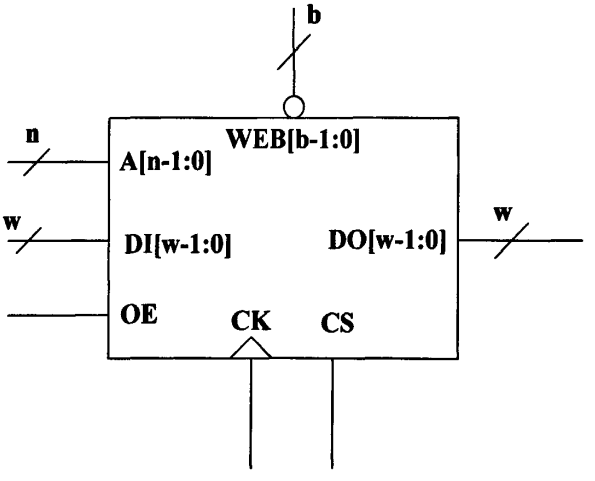


图 3.2 SRAM 外部接口

SRAM在外部控制信号OE，CS，WEB，CK的作用下，处于不同的工作状态，实现读，写，静态和高阻等四种工作模式。SRAM在系统时钟的上升沿采入外部输入的地址，数据和控制信号。各信号只需满足一定的建立时间和保持时间，SRAM即可在使能信号确定的模式下以系统时钟确定的频率工作，进行读操作或写操作，或者处于静态模式。

表3.3 SRAM工作真值表

OE	CS	WEB	DO	工作模式	功能描述
X	L	X	上一次读出的数据	静态模式	地址输入被禁止，存储器数据保持不变，但不能新的读取或写入。输出数据保持稳定状态.
H	H	L	输入数据	写模式	在数据输入总线D[n-1:0]上的数据写入到地址总线A[m-1:0]所指定的存储单元中；并被输出到数据输出总线DO[n-1]上. 具备字节写功能
H	H	H	存储单元内的数据	读模式	由地址总线A[m-1:0]指定的存储单元内的数据被输出到数据输出总线DO[n-1:0]上
L	X	X	Z	高阻模式	输出数据处于高阻状态

3.1.2 SRAM 基本工作时序

SRAM 工作模式分为：读模式，写模式，静态工作模式和高阻模式。但一般从系统应用的角度而言，高阻模式是不必要的。memoryBIST 关心的主要是 SRAM 的读模式和写模式，以及静态工作模式。

图 3.3 表示了 SRAM 读模式下的时序。从图上可以看出，时钟上升沿到来之后，数据经过时间 Taa 之后，出现在输出引脚上，这段时间称为输出延迟时间，因为数据输出有一个输出延迟时间，所以为确保正确获取 SRAM 的输出数据，在 memoryBIST 时序设计的时候我们需要延迟一个周期进行取数据比较，这在后文的 memoryBIST 时序分析中会涉及到。

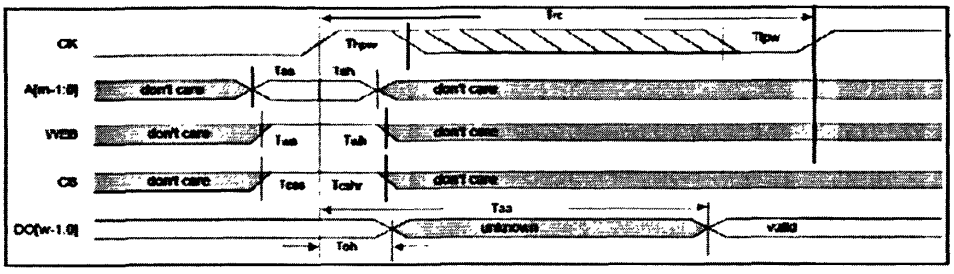


图 3.3 SRAM 读工作模式时序

在 SRAM 设计时，为了确认输入数据的正确性，写入的数据被写回到数据输出引脚上。这种设计称为：write-through。如图 3.4 表示了 SRAM 写模式下的时序。输入数据最短经过 Twdx 时间后，出现在数据输出引脚上。

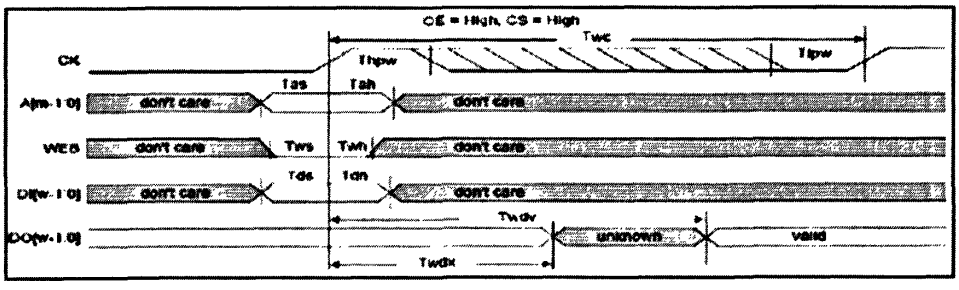


图3.4 SRAM写工作模式时序

当SRAM不进行读或写操作时，只要系统不掉电，SRAM存储的数据都不会丢失，此时SRAM处于静态工作模式，系统只消耗较低的静态功耗。数据输出引脚上保持上一次输出的数据状态。图3.5为SRAM静态时序图。表3.4是上图中SRAM时序参数的说明。

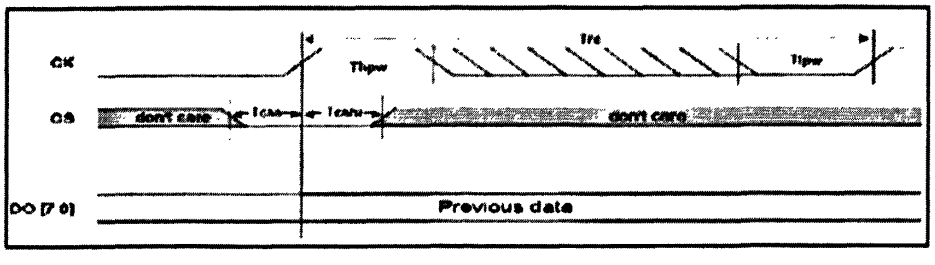


图3.5 SRAM静态工作模式时序

表3.4 SRAM时序参数说明

参数	描述	参数	描述
Trc	系统工作时钟周期	Tcshr	CS信号保持时间
Thpw	时钟高电平宽度	Toh	输出数据保持时间
Tlpw	时钟低电平宽度	Taa	数据访问时间
Tas	地址输入建立时间	Tds	输入数据建立时间
Tah	地址输入保持时间	Tdh	输入数据保持时间
Tws	WEB信号建立时间	Twdv	时钟上升沿后,数据有效时间
Twh	WEB信号保持时间	Twdx	时钟上升沿后,数据有效时间
Tcss	CS信号建立时间		

3.1.3 SRAM 存储单元工作原理

1．存储单元基本结构

SRAM 存储单元是存储器主要功能单元，单口 SRAM 多采用六管存储单元结构，如图 3.6 所示。

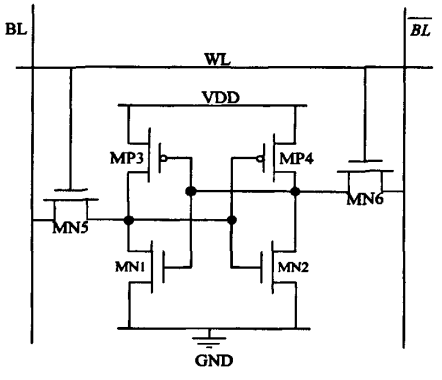


图 3.6 六管 SRAM 存储单元

静态六管存储单元（6T）由一对交叉耦合的反相器来锁存一位二进制数。如图 3.6 所示,其中 MN1、MP3 组成反相器 1 与 MN2、MP4 组成反相器 2 构成

双稳电路，MN5 和 MN6 则被称为传输管，它们在对存储器进行读写操作时，完成将存储单元与外围电路连接或断开的作用。整个电路呈对称结构排布，对应的管子在尺寸与性能上保持一致。两根互补的位线来传送数据的正反信号，提高了 SRAM 在读写时的噪声容限。

本设计中双口 SRAM 采用 8T 结构单元，如图 3.7 所示。

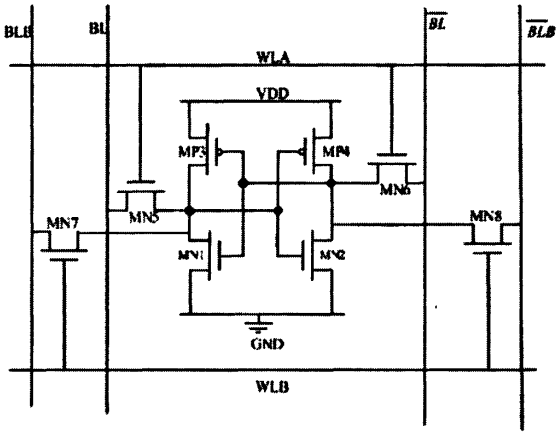


图 3.7 八管 SRAM 存储单元

和六管存储单元比起来，八管存储单元多了一对传输管 MN7 和 MN8，一条字线 WLB 和互补的两条位线 BLB、$\overline{BLB}$。两条读写路径分别为两个端口的读写操作独立使用。双端口 SRAM 的读写控制机制和单端口 SRAM 是一样的，只是数据路径不一样。下文为了便于描述和分析，关于存储单元工作原理的描述将以六管单元（6T）为例。

2. 存储原理

存储单元结构是有两个反相器首尾连接而成的电路，如图 3.6 所示。图 3.8 显示的是单元里两个反向器的电压传输特性曲线，两条曲线共有三个交点：A、B 和 C，其中 A、B 两点表示电路工作在稳态，而 C 点则意味着电路处于亚稳态。假设此时电路偏置在 C 点。假设由于噪声引起了偏置点的一个小偏移 δ，这个偏移将会沿电路环路被放大和再生，这是由于沿此环路的增益大于 1 造成的。如图 3.8(a)中所示，一个小偏移 δ 加在一个反相器上（偏置在 C 点），放大的偏置又加到第二个反向器并再次放大。如此往复，偏置点将会从 C 点移开，直到达到 A 或 B 中的一个工作点，稳定下来。由此 C 是一个不稳定的工作点。每一个

偏移（甚至是非常小的偏移）都会被反复放大，都会使工作点远离它原来的偏置。反之，A 和 B 则是稳定的工作点，如图 3.8(b)所示。由于工作点 A 和 B 下反相器的环路增益比 1 要小很多。即使从这两个工作点有相当大的偏移 δ 也会被减小直至消失。

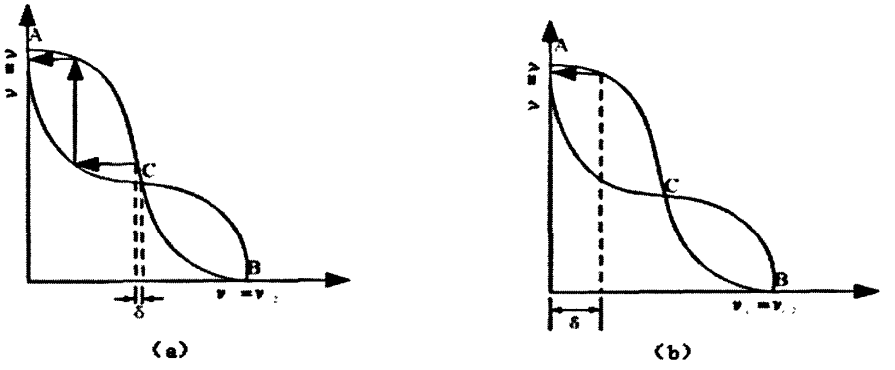


图 3.8 亚稳态与稳态工作点

由上可知，交叉耦合的两个反向器形成了双稳态电路——也就是一个电路具有两个稳定状态，每一个对应一个逻辑状态：存放 1 或 0（相对应位置 A 和 B）。在不存在任何触发以及在不掉电的情况下，电路能够稳定保持在某个状态，因而“记忆”了一个状态值。实现了 SRAM 存储高电平和低电平两种状态的基本功能。使用这种电路被用作数据存储电路，它完全可以实现对数字信号的存储和非破坏性读出。

3．SRAM 读操作分析

读操作是实现将存储单元所存储的电平值传输到所对应的位线上，然后向外输出。读操作必须保证存储单元的状态在读的过程中不发生偏转，否则会导致读出错误的数据。

现在存储器设计都采用预充电模式，如图 3.9 所示。所谓预充电模式是指在每次读写操作结束后，位线低电平，导通管 MN5 和 MN6 截止，由位线预充电路 MN7，MN8，MN9 对位线充电，使之统一恢复到高电位，为下一次读写做准备。其中 MN7 和 MN8 作为“充电管”给位线充电，MN9 作为“平衡管”，用来平衡读写操作前字线上的电压不平衡。

假设存储单元存储的是数据“1”，即节点 Q、Q_上所存储的信号分别为“1”

、“0”。读作之前，导通管 MN5 和 MN6 截止，数据被串联的反相器“锁存”在存储单元内，两根互补的位线都被“预充电”成高电平。读操作发生的时候，地址译码器译码之后的地址指向该存储单元，字线“WL”高电平，导通管 MN5 和 MN6 导通，由于 Q 点高电平，MN2 导通，位线反上的寄生电容通过导通的 MN2 和 MN6 进行放电到 GND，同样由于 Q_低电平，位线通过 MN3 和 MN5 充电，依然保持预充电时的高电平。至此，存储单元一对相反的“1”和“0”信号传送到了一对相反的位线上，继续通过敏感放大器放大位线上得到的信号，并向外输出数据，完成 SRAM 的读功能，从而保证了位线读出的信号与存储单元所存储的信号一直。读操作完毕，相应字线重新恢复为低电平。由上面的分析我们可以看出在对存储单元进行读操作时，对位线 B 的充电是通过 PMOS 负载管 MP3 与传输管 MN5 来完成的，由于 NMOS 管在传输高电平时有阈值电压损失，且存储单元的 PMOS 管一般采用小尺寸，加之大容量存储器位线上的寄生电容通常会达到 PF 的量级，因此，充电将会是一个很长的时间，这样势必会影响整个存储器的读取速度，这就是为什么现在存储单元设计通常采用预充电模式的原因，更快的充放电时间能得到更快的读写速度，从而得到更快的 SRAM 工作速度。

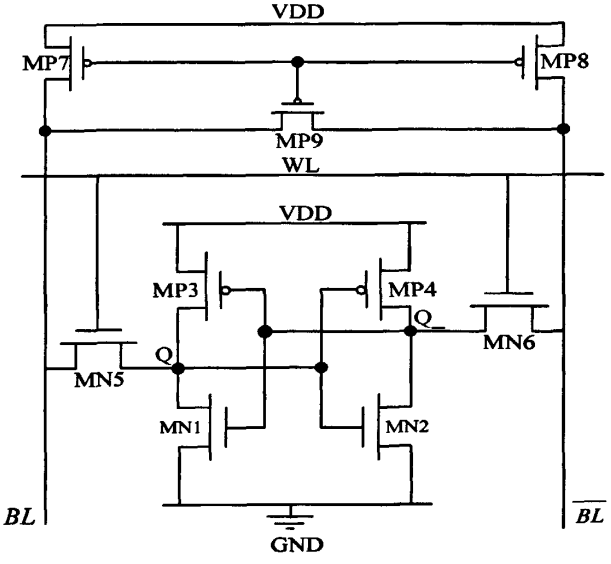


图 3.9 带有预充电模式存储单元

4. 写操作分析

写操作过程与读操作正好相反，它要使存储单元的状态按照写入的数据进行相应的翻转。如果要写入的数据与已经存储的数据是同一个状态，那么存储单元没有任何翻转的过程，这个保持的过程很快，且功耗很低。假如需要写入一个相反的状态，我们仍然假定存储单元（图 3.9）所存储的数据为“1”，即 Q 点值为“1”，Q_的值为“0”，而此时要写入的数据为“0”，即要通过写操作使存储单元的状态发生翻转。写入数据的时候，字线 WL 得到高电平，传输管 MN5 和 MN6 导通，位线 BL 和 $\overline{BL}$ 上分别加上低电平和高电平，MN5 作为传输门工作，Q 点的高电平通过 NM2 放电到 0，Q_点的低电平虽然不能通过 MN6 充电到 1，因为 NMOS 管的传输门有阈值损失，但是我们根据前文所分析的存储单元工作原理，反向器的正反馈将会把 Q_点拉到高电平“1”，使得存储单元工作在稳态下，这样就改写了 Q 和 Q_的状态，写入数据被存储下来，完成了存储器的写操作。BL 与 $\overline{BL}$ 分别为“1”和“0”时的写操作就不再敷述，原理与此相同。

第二节 SRAM 基本工作故障模型分析

研究存储器的检测方法，就必须先建立存储器单元的故障模型，也就是要求把物理故障模型化为逻辑故障。常用的测试机理是比较有故障电路与好的电路的逻辑行为，将物理故障模型转化为逻辑故障，通过检测故障模型达到检测物理缺陷，从而回避了对物理缺陷分析的复杂度。

在数字逻辑测试中，固定故障是最常使用的故障模型。随着 VLSI 设计尺寸的进一步缩小、越来越多的故障模型比如跳转故障、延迟故障、桥接故障被用于数字逻辑测试中。但是存储器是一个高密度的设计，不同于一般的数字系统，仅这些模型并不能充分证明存储器功能的正确性。因此，越来越多的存储器故障模型被应用于测试。并且针对特殊工艺或特殊架构的存储器故障模型也愈来愈多地被提及。本节将分析主要的几种故障模型，基本能包含 90%以上的 SRAM 故障类型。

对于 SRAM 来说，其主要的故障模型可以分为三类[10]：存储单元阵列故障、译码部分逻辑故障、读写控制逻辑故障（包括灵敏放大器、写驱动以及其它控制逻辑）。

3.2.1 单端口存储器存储单元阵列故障机理分析

存储单元阵列的基本元素是存储单元（图 3.9），我们可以将存储单元阵列的故障模型等价为存储单元的故障模型。在正常工作状态下，假设 MN1 导通，MN2 截止，表示存储“0”；MN1 管截止，MN2 导通，表示存储“1”。假设 Q_点被固定在低电平（如 Q_点接地），强制 MN1 管截止，即存储单元永远处在“1”状态，就不能存储“0”数据。同理如果 Q 点被固定在低电平，MN2 管截至，即被固定在“0”状态，此时再也存储不了“1”数据。如果字线或者位线发生了互联线的断路（这或许是因为生产工艺厂家的工艺波动，也可能是因为互联线上电流造成的电迁移现象），这也将导致一个或多个单元无法获取，外部表现为该单元被固定在某个状态下。

如果存储单元电路中的某个元件损坏，或者不同的存储单元之间之间发生短路、断路，都会导致触发器的双稳态的锁存功能失效，不能存储“1”或者“0”数据，实现不了由“0”改变为“1”（UP 故障）或由“1”改变为“0”（DOWN 故障）的转换，实现不了 SRAM 的正确读和写的功能。存储单元与存储单元之间的短接或耦合，也造成一个或多个存储单元状态的改变必然会引起另一个存储单元的状态改变。译码器，敏感放大器故障等等这些部件发生了功能故障，也会导致读写错误的发生。

我们根据存储器故障发生的表现形式归纳起来，存储单元故障有[10,11,12,13]：

1．固定故障（Stuck-At Fault，简称SAF）：表现为阵列中一个或多个单元的一位或多位固定为“1”或“0”(s-a-1/0)；图3.10是存储器单元的正确转换图。S0表示逻辑0单元的状态，而S1是表示逻辑1的状态。图3.11是故障发生时的状态转换，s-a-0故障可通过W1（写1）操作敏化，而s-a-1可通过W0（写0）操作敏化。敏化之后通过R1（读1）操作检测s-a-0故障，通过R0（读0）操作检测s-a-1故障。

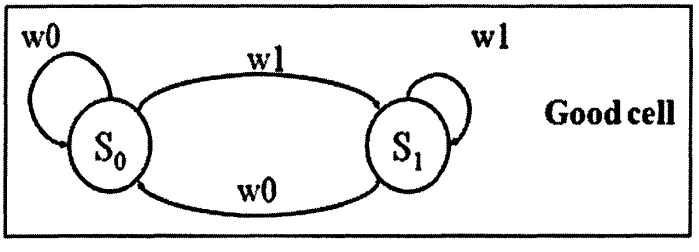


图3.10 无故障单元的状态图

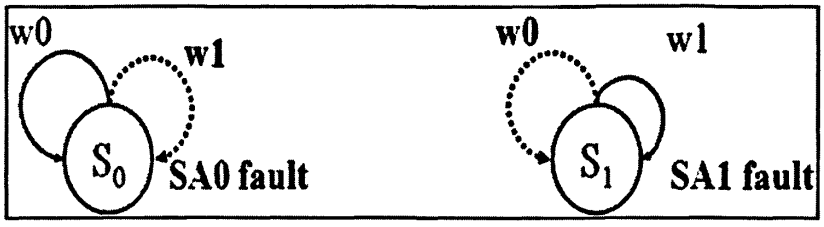


图3.11 固定故障单元的状态图

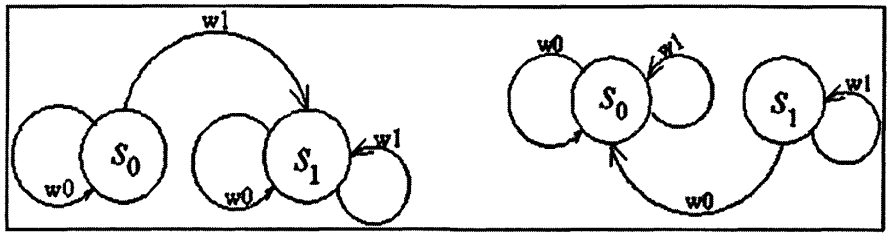


图3.12 转换故障的状态图

2．单元开路故障（Stuck-Open Fault，简称SOPF）：表现为某个单元的存储内容永远无法获取，这时候存储器输出是不确定的，可能是固定在“1”或者“0”，表现和固定故障单元一样；也可能输出敏感放大器的上一次数据；还可能是随机的。输出“1”和“0”的可能性都是50%。

3．状态转换故障（Transition Fault，简称TF）：可分为向上转换故障和向下转换故障，如图3.12所示。向上转换故障是指不能从“0”变到“1”（0→1）；向下转换故障是指不能从“1”变到“0”（1→0）。有时候状态转换故障也表现为固定故障，例如当具有状态故障的单元加电到状态“0”时，表现为s-a-0故障，然而，与另一个单元耦合的故障可引起这个有故障单元回复到“1”状态，导致表现为无故障发生，因此固定故障不能模型化转换故障。根据van de Goor[6]，必须检测和定位所有转换故障的测试矢量具有下列必要条件：每个单元必须经历向上转变和向下转变。每个转变之后，在经历更进一步转变之前读它。

4．耦合故障（Coupling Fault，简称CF）：指某些存储单元的跳变导致其它存储单元的逻辑值发生非预期的变化，耦合故障的情况比较复杂一些。耦合故障又包括：

(1)．倒置耦合故障（CF_{in}），一个存储单元的写“0”或写“1”操作将使另一存储单元的逻辑值倒置，即由“0”变为“1”，或者由“1”变为“0”。

有两种倒置耦合故障：CF_{in}（#；ε），CFin（∃；ε），其中“ε”表示存储单元的值倒置；“#”表示对存储单元写“1”操作，“∃”表示对存储单元写“0”操作。

(2). 固化耦合故障（CF_{id}），一个存储单元的写“0”或者写“1”操作将使另一存储单元的值固定在一个确定值，这个值可能是“0”，也可能是“1”。有四种固化耦合故障：CF_{id}（#；0），CF_{id}（#；1），CF_{id}（∃；0），CF_{id}（∃；1）。

(3). 状态耦合故障（CF_{st}），一个存储单元i的值决定了另一存储单元j的值。有四种状态耦合故障：（0；0），（0；1），（1；0），（1；1）。限于两个存储单元之间的耦合故障称双耦合故障，多个存储单元之间的耦合故障称k-耦合故障。k-耦合故障的测试非常复杂，一般要对耦合单元的位置做一些限制。假设C_i和C_j是不同字的两个存储单元。双耦合故障是以下四种方式的一种：C_i（源）0→1的转变导致C_j（目标）0→1的转变；C_i（源）0→1的转变导致C_j（目标）1→0的转变；C_i（源）1→0的转变导致C_j（目标）0→1的转变；C_i（源）1→0的转变导致C_j（目标）1→0的转变。要检测这种错误，存储器必须能按两个完全相反的地址顺序进行读写。假设所选的地址计数器使RAM只能按一个地址顺序进行读写，如i总是先于j被访问，在j中存在上述耦合错误，则在对i字进行写周期内，j字的故障使得其中的内容发生改变，但在随后j的写操作将覆盖其中已有错误的内容，而被正确的值所取代，从而该错误不会被检测。但若先对j写再对i写，这种情况将不会发生。由于并不能预测源和目标以及它们的访问顺序，因此要求产生每个单元地址的地址计数器要有按相反方向计数的功能。

5. 邻近图案敏感故障[10]

存储单元中的数据受与其邻近的其他单元中的数据组成的特定图案的影响，称为邻近图案敏感故障。可分为主动、被动和静态三种模式。给定一种特定的邻近图案，图案中一个单元数据的改变将影响另一个单元中的数据，称为主动模式。一种特定的邻近数据图案使一个单元中的数据固定不可改写，称为被动模式。一种特定的邻近单元图案使另外一个单元中的数据为一个特定值，称为静态模式。检测邻近图案敏感故障是非常复杂的，需要许多种不同的检测途径。要将所有这些检测方法在BIST电路中实现是不现实的，因为这要大大增加芯片面积并需产生很长的测试向量。

3.2.2 地址译码故障

地址译码电路是SRAM电路中非常重要的组成部分。由于外部可用端口数量有限，存储器的操作不可能直接指向每一个具体存储单元进行操作，所以SRAM设计通常使用一组地址译码器对单元进行编码，称为存储单元的“地址”，每一个地址对应一个存储单元，如果地址译码器失效，将会导致一些存储单元无法存取，而如果地址译码器由于内部错误指向了多个存储单元，也会导致地址译码故障，无法正确读取所期望的单元。为了减少外部引脚需求数量，译码器的设计一般都采用两级或多级译码电路的设计方法，多级译码器的设计可以减少第一级译码器的引脚数，但是这种设计出错的几率也随之增加；同时地址还需要根据存储阵列的划分而区分不同的行和列，也增加了出错的可能性。译码器电路中，可以区分出地址输入总线、输出驱动线和中间连线，这些线之间的相互短路，或者线本身的断路，以及与地火电源线短路等等故障都会导致译码功能失效，无法正确指向所需的存储单元。

根据译码器出错的表现形式，以上故障可总结为两种情况，一种是某个存储单元无法被任何地址选中；另一种是某个存储单元可以被多个地址选中。这两种故障都可以用存储单元的故障形式表示，若为第一种故障，则可以看成是应该被选中的单元固定为“0”或“1”的故障，而若发生了第二种故障，可看成是多个存储单元的耦合故障。因此，译码器逻辑部分有故障时，可转换成存储阵列的错误，从而不需特别考虑该部分的测试。也就是说，存储单元与译码器逻辑故障具有不可区分性[14,15]。

3.2.3 读写逻辑故障

读写逻辑将数据信息从输入输出引脚传送到存储器单元阵列。随着集成度的提高，每一对位线上连接的单元数在增加，使位线加长，位线电容增大，在一定的位线摆幅和驱动电流条件下，将使读操作时位线通过单元放电速度很慢，为了提高读出速度，必须缩短位线放电时间。减小C_L是很难的，增大I_d又受到芯片尺寸的限制，一个解决办法就是减小位线上的电压摆幅，也就是互补位线上的电压差应尽量小，在正常读操作时通过灵敏放大器对两根互补位线上的小电压差进行放大。因此采用高增益高灵敏度的读出放大器是减少存储器阵列延迟，提高SRAM读取速度的重要手段。写驱动电路对于写入正确的数据起着重要的作

用。

读写逻辑中的障碍有三种形式：

(1) 存储单元中的一位或多位固定为“1”或“0”；

(2) 单元中的一位或多位固定为开路故障；

(3) 存储单元中的任意两位之间有状态耦合障碍。

这几种情况都可以看作是存储单元的错误。第一种可看成是与有固定障碍的输入、输出线相对应的存储单元的固定故障；第二种也同理，第三种可以看作是存储单元之间的耦合故障，即读写逻辑故障也具有与存储单元故障的不可区分性。

3.2.4 双端口故障模型

双端口存储器的两个端口可以同时访问存储器，且端口间是相互独立的。所以双端口故障工作不仅可能存在单端口的所有故障，还包括：

1. 单个单元故障（2PF1），通过两个端口 A，B 激活故障的单元与出现故障的单元是同一个单元；

2. 两个单元故障（2PF2），分为两种情况：一是应用两个连续的操作一个到耦合单元，一个到被耦合单元，激活被耦合单元的故障。一是两个连续的操作同时应用到耦合单元，激活被耦合单元的故障；

3. 三个单元故障（2PF3），应用两个连续的操作到两个不同的耦合单元激活被耦合单元的故障。

以上分析可知，对于双端口特有的故障是由于双端口同步操作所敏化[16,17]，所以当我们需要探测双端口故障的时候，必须在操作中加入两个端口同时工作的状态，敏化双端口故障的发生，从而达到检测故障的目的。从某种意义上说，单端口故障是一种弱错（weak fault），而双端口故障则是一种强错（strong fault），可以理解为多个弱错敏化了一个强错的发生。

第四章 测试算法分析

内建自测试的实质是测试算法在芯片内部的硬件实现，所以测试算法的好坏在很大程度上决定了BIST的质量。衡量一个算法的好坏有两个因素：故障覆盖率和测试所需时间。因此针对近年来设计尺寸的缩小，新的故障模型的出现，新的测试算法也在不断涌现。本章在前一章SRAM故障模型分析的基础上，讨论了几种流行的测试算法，并分析了本次BIST设计所采用的算法以及基于这些算法进行内建自分析的原理。

第一节 存储器测试所使用的几种算法

1. ALL0/ALL1 算法

ALL0/ALL1 是最基本，且最简单的存储器测试算法。它能有效检测固定故障的发生。ALL0/ALL1 的基本步骤是首先对存储器所有的存储单元写“0”，然后读存储器，接着用写操作把所有存储了“0”的单元全部改写成“1”，最后读刚才写入的“1”。如果存储器输出与期望值相同，那么表示存储器功能良好，没有错误发生。

2. GALPAT（跳步）和 Walking1/0（走步）算法

GALPAT（GALloping PATtern）和 Walking1/0 算法具有相似的操作过程。首先它们将除了基本单元以外的所有单元（普通单元）写入“0”或“1”，而基本单元中写入相反的“0”或“1”测试过程中，基本单元“走”遍整个存储器，也就是说使存储器中的每个单元都有机会成为基本单元。两种算法的区别在于读基本单元的方式：走步法每“走”一步，总是对普通单元进行读操作，然后读基本单元；跳步法也是每“走”一步对所有单元进行读操作，只是每读一个普通单元都要读一次基本单元。整个过程的操作次数按照如下公式计算 $N+N(2+2(N-1)+N=2(2N^2+N)$，其算法复杂度 $O(N^2)$。

漫游0/1过程可以检测下列故障：

(1) SAF故障。因为漫游过程只对每一个单元的1以及0都进行了读写操作。

(2) 跳转故障。因为漫游过程每一个单元存在有1→0以及0→1的变迁。

(3) 译码电路的故障。因为漫游过程重复测试地址译码的功能性。

(4) 部分状态耦合故障。因为漫游过程任意两个单元的00、01和10状态被将读出来。

3. CheckBoard算法

CheckBoard算法是存储器测试最常用的一种简单有效的算法。测试的过程是首先对每一个存储单元赋值，赋值的结果期望每一个单元与其紧密相邻的各个单元的值都不同，如图4.1所示；然后把整个存储阵列分为两块，所有状态为0的形成分块W和所有状态为1的形成分块B；接着分别对分块W和分块B进行读写；最后用另外一个逻辑值重复上述过程。

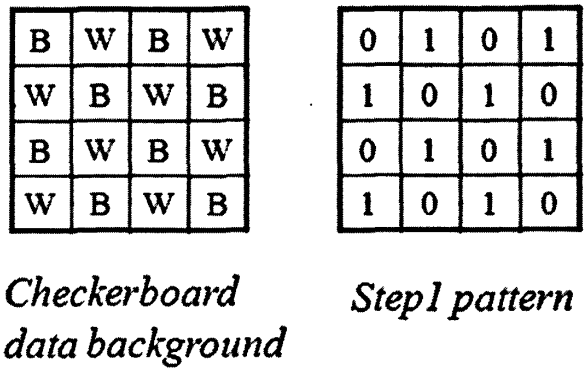


图4.1 CheckBoard算法

CheckBoard测试的算法过程为：

(1)对分块W（B）中的单元写0（1）；

(2)读所有的单元；

(3)对分块W（B）中的单元写1（0）；

(4)读所有的单元。

CheckBoard测试算法简单易行，且对故障出现率较高的SAF故障和相邻单元间的短接故障有很好的覆盖，所以在BIST算法中应用广泛。

4. March测试算法[18,19,20,21,22]

基本的March测试简称为MATS，对每个存储器只需10次读/写操作，其复杂程度与存储容量N成正比。MATS的第一个操作过程是初始化存储器阵列（对全部存储单元写“1”或“0”），接着读每一个单元，然后再对单元改写再读出，

然后按照相反的测试向量重复操作。

March测试的第二个过程是重复第一个过程，但处理单元的先后顺序与第一个过程相反，即第一个过程中第一个读和求补的单元，在第二个过程中是最后一个要处理的单元。MATS用算法表述如下：

```
For I = 1 to N; Do
        Write 0 at cell I
Continue
For I = 1 to N; Do
        Read cell I to verify 0
        Write 1 at cell I
Continue
For I = 1 to N; Do
        Read cell I to verify 1
        Write 0 at cell I
        Read cell I to verify 0
Continue
End
```

March 测试是常用的一种存储器测试算法。为了测试不同的故障，衍生了很多 March 算法的变种，这些算法的变种不同之处就在于以不同的方式（如地址递增或地址递减方式）遍历存储器的每个存储单元，并且在每次遍历过程中对所有存储单元进行同样的操作（在不同的遍历过程中对存储单元的操作方式可以不同）。

不同的遍历方式和存储单元操作方式能检测到的故障模型是不同的，所需要的测试时间也是不同的，根据存储器的遍历方式和存储单元的操作方式，March 测试算法又可以细分为 MATS，MATS+，MATS++，March X，March C-，March C+，March A，March Y 等算法，这些算法的测试流程如表 4.1 所示。这里，“⇑”表示地址升序变化；“⇓”表示地址降序变化；“⇕”表示地址升序或降序变化均可。W0 表示写入一个数据测试背景图形；W1 表示写入一个相反的数据测试背景图形。R0 和 R1 分别表示读出并验证一个数据测试背景图形和一个相反的数据测试背景图形。

由表 4.1 我们可以看到，March 及其变种算法就是按照一定的地址顺序，对

存储器中的每个单元遍历 0 和 1 两个相反的数据，不断地进行读和写到操作，其中对于双端口的测试算法March D2PF在读写的过程中还需要区分不同的行和列进行读写操作，以诱导不同类型的故障发生，达到检测 SRAM 的目的。

表4.1 March及其变种算法流程

MATS {⇳(W0)；⇳(R0, W1)；⇳(Rl)}
MATS+ {⇳(W0)；⇑(R0, W1)；⇓(R1, W0)}
MATS++{⇳(W0)；⇑(R0, W1)；⇓(Rl, W0, R0)}
March X {⇳(W0)；⇑(R0, Wl)；⇓(Rl, W0)；⇳(R0)}
March C-{⇳(W0)；⇑(R0, W1)；⇑(R1, W0)；⇓(R0, W1)；⇓(Rl, W0)；⇳(R0)}
March C+ {⇳(W0)；⇑(R0, W1, R1)；⇑(R1, W0, R0)；⇓(R0, W1, R1)；⇓(Rl, W0, R0)；⇳(R0)}
March D2PF { $\Updownarrow(W0:n)$； $\Uparrow_{c=0}^{C-1}(\Uparrow_{r=0}^{R-1}(W1_{r,c}:R0_{r+1,c}))$； $\{\Uparrow_{c=0}^{C-1}(\Uparrow_{r=0}^{R-1}(W1_{r,c}:R1_{r+1,c}))$ ； $\{\Uparrow_{c=0}^{C-1}(\Uparrow_{r=0}^{R-1}(W0_{r,c}:R1_{r+1,c}))$； $\{\Uparrow_{c=0}^{C-1}(\Uparrow_{r=0}^{R-1}(W0_{r,c}:R0_{r+1,c}))$ ； $\{\Uparrow_{c=0}^{C-1}(\Uparrow_{r=0}^{R-1}(W1_{r,c}:R0_{r,c+1}))$； $\{\Uparrow_{c=0}^{C-1}(\Uparrow_{r=0}^{R-1}(W1_{r,c}:R1_{r,c+1}))$ ； $\{\Uparrow_{c=0}^{C-1}(\Uparrow_{r=0}^{R-1}(W0_{r,c}:R1_{r+1,c}))$； $\{\Uparrow_{c=0}^{C-1}(\Uparrow_{r=0}^{R-1}(W0_{r,c}:R0_{r+1,c}))\}$

第二节 MARCH 算法应用分析

对于 March 算法如何检测故障，文献[18,20]给出了很好的分析，这里不再赘述。本节将重点分析本次 BIST 所采用的 March 算法变种以及用 March 算法如何进行内建自分析。

在论文前文章节中曾提到随着 SOC 设计向纳米技术转移，存储器缺陷将越来越多，其产生机理也会越来越复杂。并且芯片的速度也会越来越快，目前已达到 GHz 的量级。这就意味着测试时间的减少在降低测试成本上所贡献的百分比将越来越少，BIST 设计的趋势将向故障覆盖率靠拢。

为了达到较高的故障覆盖率，本次 BIST 设计将同时采用 March C+[20]以及 March D2PF[23]两种算法，保证了很高的测试覆盖率。先由 March C＋算法覆盖单端口故障，再由 March D2PF 算法覆盖双端口故障。

但是对于March算法来说，其测试数据背景只有全“0”和全“1”两种，而如今存储器的位宽一般都在8位或者16位，甚至更高，我们无法针对每一位进行操作。而且通常位测试采用反复的应用全“0”和全“1”来实现的，这种方法的缺点是测试的时间长，且不能测试字内故障。字内故障是指存储器中同一个字的不同位之间的故障，即同一地址内的耦合故障，因此可采用字测试来覆盖这些故障。所谓字测试是指采用多个以8位字宽为长度的数据背景作为测试向量，以激活耦合故障发生。存储器字长与数据背景数两者的的关系如式4-1所示，其中RAM字宽m，所需的最少数据背景数K。

$$K=\log_2 m+1 \tag{4-1}$$

从上面的讨论可以看出测试算法扩展为字定向算法，主要是依靠数据背景的扩展，通过数据背景的扩展，激活字内耦合故障的发生。下文将主要以March C+算法为例，讨论如何将测试算法扩展为字定向算法，March D2PF算法的字定向扩展方法与此类似。

MarchC+ {(w0);	(r0,w1,r1);	(r1,w0,r0);	(r0,w1,r1);	(r1,w0,r0);	(r0)}
M0	M10 M11	M20 M21	M30 M31	M40 M41	M5

图 4.2 March C+算法操作序列

基于 March C+算法的故障模型查找表如表 4.2 所示，表中 M10、M11、M20、M21、M30、M31、M40、M41 和 M5 表示相应 March 元素的读操作（如图 4.2 所示），例如 M10 表示 R0 操作。各种 CF 故障我们用 CF（i；j）来统一表示，CF（i；j）表示存储单元“i”的状态和状态的变化决定了存储单元“j”的状态，其中“i”为耦合单元，“j”为被耦合单元。表中的 L 表示“i”的地址比“j”的地址低，H 表示“i”的地址比“j”的地址高，“－”表示与地址高低无关。如果读出的值与写入的值相同，那么表 4.2 中的相应元素为“0”表示无故障，否则表中相应元素为“1”表示有故障。

表 4.2 是以 SAF(1)为例说明如何得到它的故障检索类型，首先执行 March 元素 M0 对所有存储单元写“0”操作，接着执行 M10 对每个存储单元读“0”操作，由于是 SAF(1)故障，读“0”操作出现故障，因此 M10 为“1”，继续执行 M11 读“1”操作，由于是 SAF(1)故障，在 M11 中的读“1”操作不会出现

故障，故M11为“0”，依次执行下去，得出SAF(1)的故障模型检索为“100110011”。类似的得出其它故障模型检索。

表4.2 March算法故障查找表

故障模型		M10	M11	M20	M21	M30	M31	M40	M41	M5
SAF(0)		0	1	1	0	0	1	1	0	0
SAF(1)		1	0	0	1	1	0	0	1	1
TF(0→1)		0	1	1	0	0	1	1	0	0
TF(1→0)		1	0	0	1	1	0	0	1	1
CF_{in}(#; ε)	L	0	0	0	1	1	0	0	0	0
CF_{in}(∃; ε)	L	0	1	1	0	0	0	0	1	1
CF_{in}(#; ε)	H	0	1	1	0	0	0	0	1	1
CF_{in}(∃; ε)	H	0	0	0	1	1	0	0	0	0
CF_{id}(#; 0)	L	0	0	1	0	0	1	1	0	0
CF_{id}(#; 1)	L	0	0	0	1	1	0	0	0	0
CF_{id}(∃; 0)	L	0	1	1	0	0	0	1	0	0
CF_{id}(∃; 1)	L	1	0	0	0	1	0	0	1	1
CF_{id}(#; 0)	H	0	1	1	0	0	0	1	0	0
CF_{id}(#; 1)	H	1	0	0	0	1	0	0	1	1
CF_{id}(∃; 0)	H	0	0	1	0	0	1	1	0	0
CFid(∃; 1)	H	1	0	0	1	1	0	0	0	0

本文中待测试存储器字宽为8位，根据式4-1，至少需要4个8位数据背景，分别是：{00000000，11111111}；{10101010，01010101}；{11001100，00110011}；{11110000，00001111}。通过以上数据背景的改进，使March C+算法可以覆盖字内故障，进一步提高了故障覆盖率。

改进后March C+算法的新测试序列就图4.3所示。我们看到，测试算法的数据背景由原来2个测试向量增加到了8个测试向量，所以与此同时，March C+测试算法的测试时间会提高到原来的4倍。由此可知，数据背景的扩展也是以牺牲测试时间为代价的。

通过上面的数据背景扩展以及March C＋和March D2PF的组合算法，使得本次BIST的故障覆盖率达到了95%以上，达到了设计要求。

```
{↑↓(W00000000);
↑(R00000000, W11111111, R11111111);  ↑(R11111111, W00000000, R00000000);
↓(R00000000, W11111111, R11111111);  ↓(R11111111, W00000000, R00000000);
↑↓(R00000000, W01010101);
↑(R01010101, W10101010, R10101010);  ↑(R10101010, W01010101, R01010101);
↓(R01010101, W10101010, R10101010);  ↓(R10101010, W01010101, R01010101);
↑↓(R01010101, W00110011);
↑(R00110011, W11001100, R11001100);  ↑(R11001100, W00110011, R00110011);
↓(R00110011, W11001100, R11001100);  ↓(R11001100, W00110011, R00110011);
↑↓(R00110011, W00001111);
↑(R00001111, W11110000, R11110000);  ↑(R11110000, W00001111, R00001111);
↓(R00001111, W11110000, R11110000);  ↓(R11110000, W00001111, R00001111);
↑↓(R00001111);}
```

图 4.3 扩展数据背景后的 March C+算法

但是做为一颗测试芯片，仅仅知道有故障产生还远远不够，还必须知道故障发生的位置以及发生了何种故障，这样才方便我们去调试。所以本次 BIST 设计加进了内建自分析模块（BISA），其目的就是通过测试算法将故障的信息向外输出，得到具体出错的存储器地址和出错数据。

第三节 小结

对于 BIST 设计来说，测试算法是设计的核心，如何选择高效的算法是 BIST 设计的关键。本章首先介绍介绍了几种常用的测试算法：ALL0/ALL1 算法、GALPAT（跳步）和 Walking1/0（走步）算法、CheckBoard 算法、March 测试算法，总结了这几种算法的特点；接着扩展了 March 算法的数据背景，分析了改进后算法的组合以及算法如何能够覆盖故障模型，最终选定本次 BIST 设计所需的测试算法。了解了各种算法的故障覆盖率之后，为今后我们设计 BIST 芯片挑选合适的算法提供了可靠的依据，选定了设计所需的测试算法，下面将进入具体的设计实现阶段。

第五章 芯片前端电路设计

基于前面提出的存储器测试算法，本章介绍了该基于 ASIC 流程的 memoryBIST 芯片前端 RTL 级电路设计的详细设计细节。并且根据设计规范要求对芯片的 RTL 级设计分模块加以详细介绍，以使我们更加清楚具体电路的设计方法和分析过程。

第一节 芯片设计规范

5.1.1 基本功能

该芯片是一款基于 March 系列算法的 SRAM 测试芯片，完全符合 SRAM 时序要求，为了实现 SRAM 设计的 200MHz 的全速测试，我们使用了一颗 PLL 为 SRAM 和 BIST 提供片内高速时钟，实现在 SRAM 测试芯片内部全速测试 SRAM 的基本读写功能，保证测试时间，通过 March C+算法和 March D2PF 算法保证测试覆盖率。

5.1.2 芯片特性

- 提供 March C+，March D2PF 算法支持；
- 支持 200MHz（5ns）测试速度；
- 支持 BISA，自动保存错误信息并向外输出；
- 工艺：0.18um 1P6M Faraday STD, IO, STC SRAM；
- 工作环境：芯片内核电压范围为 1.62V～1.98V，
  温度范围为－40℃～125℃；
- 规模：BIST 部分大约 **425*84um^2**，SRAM 部分 **872*496um^2**。

5.1.3 系统框图

图 5.1 为该芯片的系统框图，通过 MCSEL[1:0]信号选择不同的 SRAM

MACRO，TCLK 为经过 PLL 之后的高速时钟信号输入 SRAM，TF 信号为错误发生信号（高有效），TS 为测试结束信号（高有效），AOUT[13:0]为错误地址输出信号， DOUT[7:0]为错误数据输出信号，通过 MUX 电路选择输出不同的 MACRO 的错误信息。

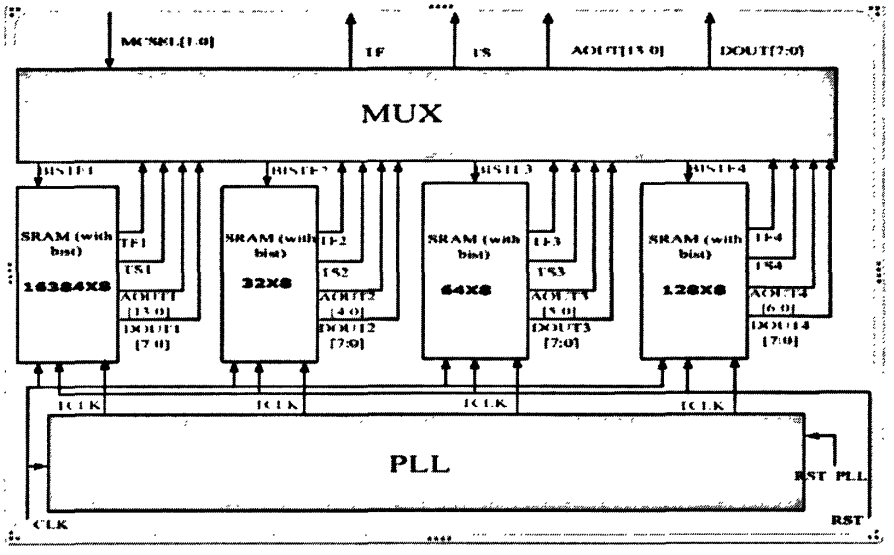


图 5.1 BIST 整体架构

5.1.4 BIST 时序分析

一般来说 BIST 测试分为三步：往存储单元写入数据，从存储单元读出数据，读出数据与期望值比较。对于传统 SRAM 测试来说，一般都是将这三个步骤分开，各占用一个时钟周期，完成一个地址的读，写和比较工作一共需要三个周期（如图 5.2 所示），这是一种比较浪费测试时间的做法。对于同步电路设计来说，这样做的好处是结构简单，便于设计。

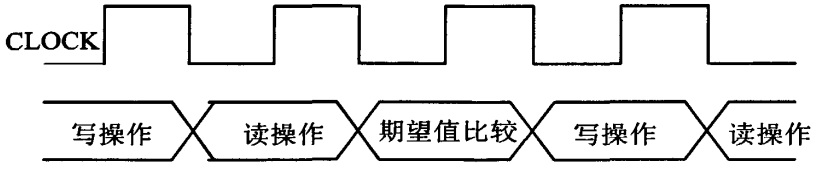


图 5.2 传统 BIST 时序设计

5.1.5 有并行结构的 BIST 时序

虽然本设计将故障覆盖率放在第一要素，但是测试时间也是芯片成本中不得不考虑的一个重要因素。因此可以考虑将期望值比较与读或写操作放在一起进行，减少一个周期占用。图 5.3（以下简称方案 A）和图 5.4（以下简称方案 B）提出了两种结构的 BIST。相比传统 BIST 时序设计需要 3 个时钟周期来说，下面两种都只需要 2 个时钟周期就可完成，测试时间大大减少。

图 5.3 所示的设计是将读操作和期望值比较放在同一个时钟周期里完成，不同的是读操作是在时钟上升沿触发的，而期望值比较是在读操作之后的下降沿进行的。两步操作的时间间隔是半个时钟周期 T/2。在 SRAM 时序分析章节中已经提到，对于存储器读操作，从上升沿触发开始，要经过一段时间 Taa 之后，数据才能送到输出端口。假设这段延迟时间比较长，超过了半个周期的时间，那么在下降沿采样将会导致采样错误，造成 BIST 本身失效。而且在实际情况下，互联线上的延时也可能会使得采样得到的数据不是最终 SRAM 输出的数据，最终能否满足设计要求需要冒很大的风险，使得方案 A 无法实行全速设计（full speed）。

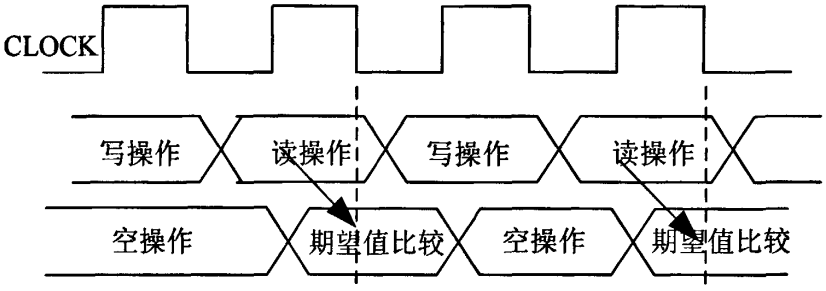


图 5.3 方案 A 时序设计

再来看方案 B，如图 5.4 所示。方案 B 将期望值比较操作延迟一个时钟周期，与写操作同步进行。可是这样一来，虽然一个周期的延时足够读出正确的 SRAM 输出，但是比较器部分会在读操作的阶段有一个空闲周期，比较浪费资源。

由于本次待测试的双端口 SRAM 特性，它支持同时读写同一个存储单元（此特性由 SRAM 内部设计决定，当同时读写同一个存储单元时，SRAM 内部将读信号做了一定的延时，SRAM 将执行先读后写的顺序，先将存储单元的内容读

出来，然后写入新的数据），为了保证测试达到 SRAM 支持的最大速度，本设计中读写将会在一个周期内完成，读先前写入的数据送去比较，同时写入新的测试数据。比较数据将会在读写下一个存储单元的时候完成，这时比较的将是上一个读写周期所操作的数据，由于相差一个周期，这种并行的时序设计既保证了读数据的正确性，又保证了测试速度，实现设计的主要任务：全速测试。

如前文所述，对于有些故障例如读延迟故障只有在频率上限附近才会出现，而测试如果不能达到频率上限，就无法测试这些故障，从而无法满足故障覆盖率这一关键因素。根据本次待测 SRAM 的特殊时序性质，本设计所采取如图 5.5 所示的时序，完成读，写，和比较三个操作只需要一个时钟周期，最大限度地提高了 SRAM 的工作周期。

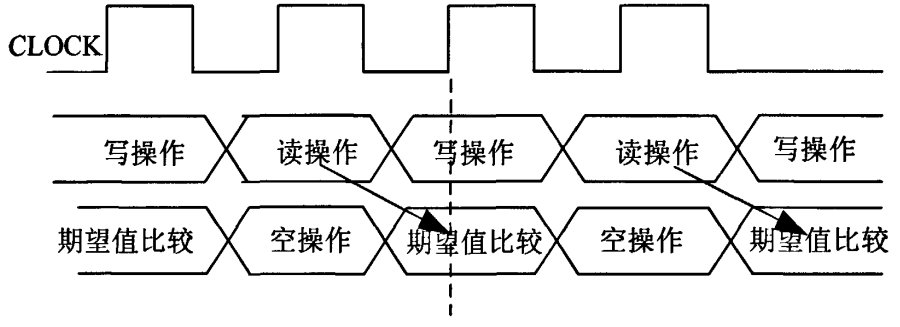


图 5.4 方案 B 时序设计

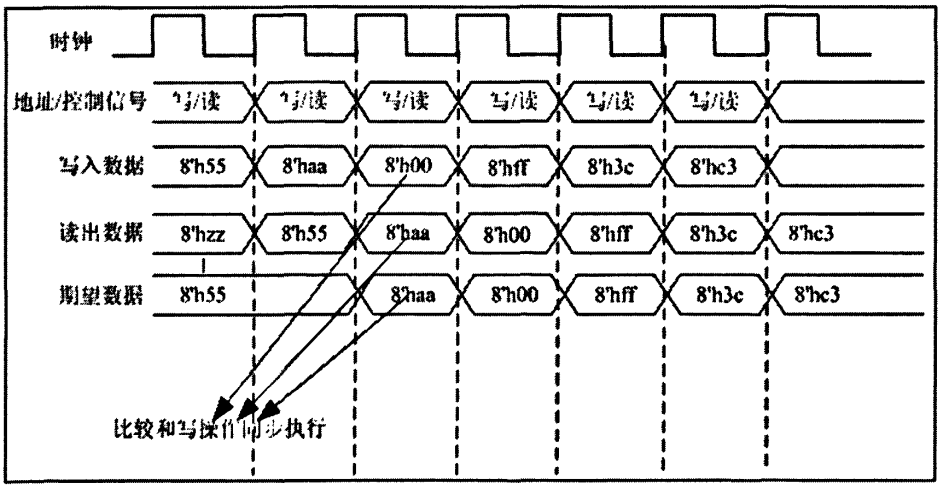


图 5.5 BIST 工作时序图

5.1.6 测试流程分析

本设计待测试的是 4 颗双端口的 SRAM，双端口 SRAM 有两个读写端口 A 端口和 B 端口，为了保证测试的覆盖率，对于每一颗 SRAM，我们都按照如下顺序进行测试：首先按照单端口 SRAM 的测试方法，用 March C+算法测试 A 端口的读写功能（写入数据和读出数据均有 A 端口操作），接着再走一遍 March C+算法测试 B 端口的读写功能（写入数据和读出数据均有 B 端口操作），然后使用 March D2PF 算法测试 SRAM 的双端口读写功能（由 A 端口写入数据，B 端口读出数据），通过这样的测试顺序，既保证了测试覆盖了每一个端口单独工作的情况，也覆盖了双端口同时工作的情况。

第二节 电路模块详细设计

确定了芯片的功能特性和寄存器组织结构，下面进入到具体电路模块的设计阶段。

5.2.1 时钟生成策略

芯片制造之后我们将会 FPGA 上进行测试，然而 FPGA 板上的晶振难以提供我们需要的高达 200MHz 的时钟波形，而如果测试达不到一定的速率要求，如前文所述，SRAM 的某些错误就无法暴露出来，也就失去了内建自测试的真正含义，为了实现全速测试我们使用了一颗 PLL ip 为我们提供稳定的高速时钟波形，如图 5.6 所示。

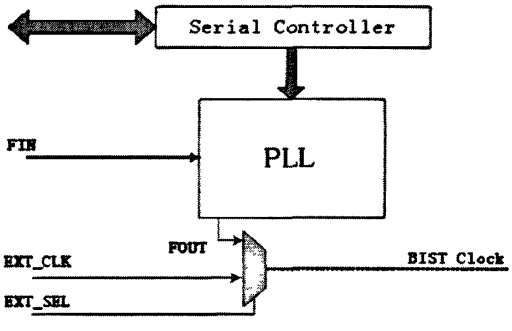


图 5.6 异步复位同步装置

上图中我们看到，BIST 的工作时钟有两个来源 FOUT 和 EXT_CLK。其中，FOUT 是 PLL 产生的倍频时钟，而 EXT_CLK 则是来自 IO 的时钟。至于为什么不直接使用 FIN，也就是 PLL 的源时钟，作为外部来源的时钟，主要是因为考虑到PLL在高速情况下可能对FIN这类时钟信号造成干扰，也就是说在FloorPlan时，FIN 会和 PLL 模块连接得很近，以期 PLL 获得最佳的时钟波形，如果 FIN 再连接到其他位置难免会受到 crosstalk 效应的影响，故而直接使用一个独立的外部时钟来减少这种不必要的干扰。倍频时钟 FOUT 和外部时钟 EXT_CLK 使用一个 EXT_SEL 信号加以选择。需要低速测试时，由 EXT_SEL 选择 EXT_CLK；反之，在全速测试过程中，则选择 PLL 的输出时钟。同时，为了支持在 FPGA 上向外慢速输出错误信息，支持 BISA 功能，如果探测到 SRAM 中出现错误，EXT_SEL 将使得 BIST 切换工作在高速和低速状态下，实现高速测试，低速向外输出错误信息。表 5.1 为所选用 PLL 的管脚内容。

表 5.1 PLL 芯片管脚

Pin脚名称	内容说明
CIN	外部时钟输入端
CKOUT	倍频时钟输出端
MS0	倍频数配置引脚1
MS1	倍频数配置引脚2
MS2	倍频数配置引脚3
MS3	倍频数配置引脚4
TCKI	测试时钟输入端
TCKO	测试时钟输出端
TEST	测试使能信号输入端

5.2.2 BIST 模块设计

图 5.7 为 BIST 芯片内部的框图，从图中我们可以看到，BIST 主要模块为地址生成模块，测试向量生成模块，向量比较模块，和 BISA 模块。由 TCLK 为 BIST 提供高速测试时钟，CLK 为慢速向外输出用的时钟，BISTE 信号为 BIST 的使能信号，AOUT 为 SRAM 错误地址信息，DOUT 为错误数据信息。

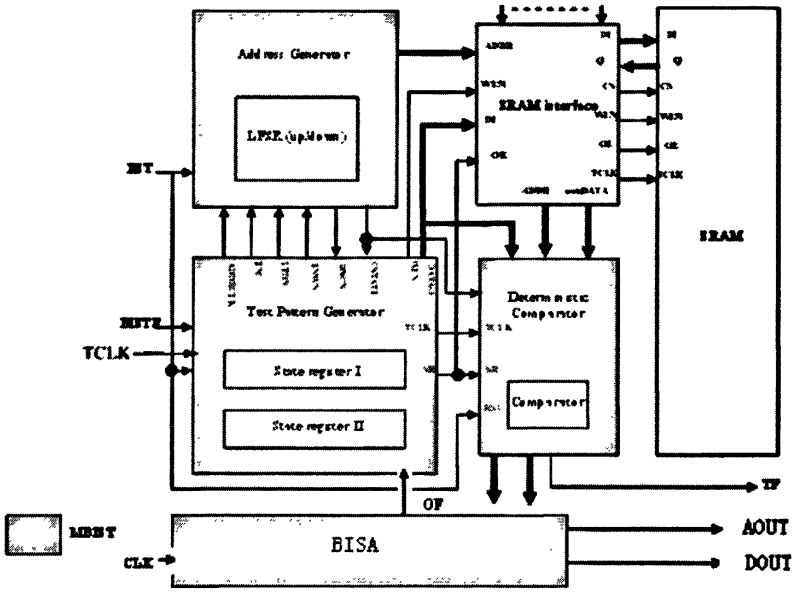


图 5.7 BIST 内部架构图

5.2.3 测试向量生成器

测试向量生成器是实现核心算法的核心部件，一般采用可编程控制器或有限状态机（FSM）控制器来设计。可编程控制器通过外部指令来实现测试算法的序列，地址的范围选取以及测试背景数据的选择。所以这种设计便于调试，移植性好，但缺点也很明显，设计较为复杂，综合出来的电路面积较大，同时速度也慢；而 FSM 控制器的算法序列由状态机实现，所以速度较快，同时与可编程控制器相比面积也小很多。由于本次 BIST 设计采用了组合算法，故障覆盖率满足设计要求，同时加入的 BISA 模块可以将故障信息输出，无需通过外部指令来选择算法达到故障分析的目的。所以本次设计的 BIST 控制器采用了有限状态机设计。状态数据取决于采用 March C+和 March D2PF 算法中的 March 元素总数。并按照测试算法步骤对其它模块施加控制信号，响应其他模块的反馈信号，控制测试的结束与否。具体来说，它体现了如下几个功能：

1．接收外部启动 SRAM 自测试的信号（CEN）；

2．提供 SRAM 所需的测试读使能和写使能信号；

3．控制地址生成器的地址变化顺序提供所需的操作地址；

4. 控制数据产生器选择数据背景；
5. 控制比较器数据背景及与 SRAM 存储数据的比较；
6. 响应地址产生器的计数边界信号，以此控制下一状态的控制信号；
7. 响应 BISA 的中断请求信号及向 BISA 输出 March 状态信号；
8. 输出测试结束信号。

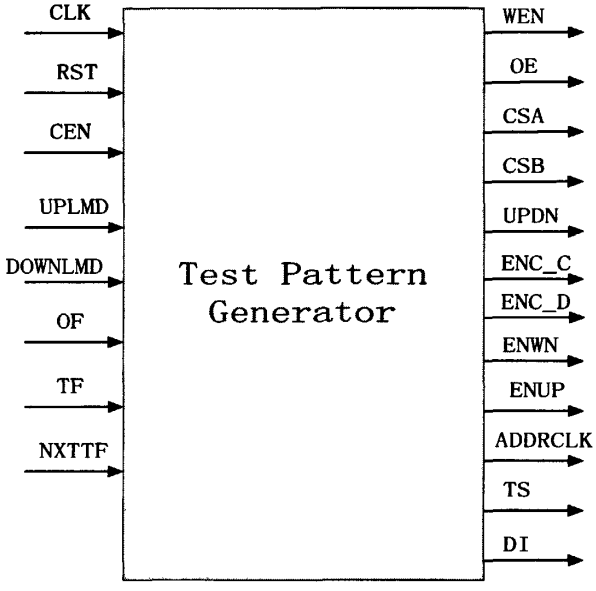


图 5.8 测试向量生成器端口描述

表5.2 向量生成器引脚说明

引脚名	类型	说明
CLK	输入	BIST 测试工作时钟
RST	输入	BIST 复位信号
CEN	输入	BIST 使能信号
UPLMD	输入	地址边界信号（上边界）
DOWNLMD	输入	地址边界信号（下边界）
OF	输入	决断模块输出结束信号
TF	输入	错误发生信号（高有效）
NXTTF	输入	TF 信号的下一个信号
WEN	输出	SRAM 的写使能信号（高有效）

续表 5.2

OE	输出	SRAM 的读使能信号（高有效）
CSA	输出	SRAM A 口使能信号（高有效）
CSB	输出	SRAM B 口使能信号（高有效）
DI [DWIDTH-1:0]	输出	数据背景输出：具体位数由 SRAM MACRO 决定
UPDN	输出	地址递增或递减决定信号（1 为地址向上递增，0 为地址向下递减）
ENC_C	输出	FSM 状态机使能信号（高有效，兼 LFSR 地址生成器使能信号）
ENC_D	输出	FSM 状态机使能信号（高有效，兼二进制地址生成器使能信号）
ENWN	输出	地址边界信号（到达下边界）
ENUP	输出	地址边界信号（到达上边界）
ADDRCLK	输出	更换地址信号
TS	输出	BIST 测试结束信号

5.2.4 地址产生器设计

地址产生器也是BIST设计的一个重要单元，一个地址对应一个单元。地址计数器用来生成测试算法所需的操作地址，计数周期应该等于存储单元的总数。本设计待测试四个存储器，其中最大的地址为16384，所以采用一个14位的计数器。又因为本文所采用的March算法的序列中，需要地址递增的顺序和地址递减的两种操作顺序，因此它应该有上下计数的功能。因此既可以用二进制计数器实现严格递增/递减的地址序列，也可以用经简单改造后的线性反馈移位寄存器[24]（Liner Feed Back Shift Register，LFSR）。

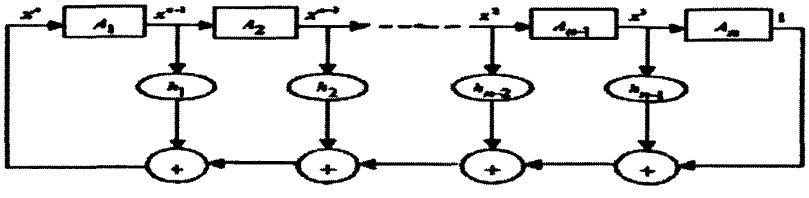


图 5.9 线性反馈移位寄存器结构

LFSR电路（图5.9）是内测试电路中最基本的模块结构。由一串寄存器和异或门组成，它的设计与操作是基于循环编码理论中的多项式算法。图5.9给出了n位的线性反馈移位寄存器的一般结构形式。其中：A1, A2, A3∧A是寄存器串联成的移位寄存器；⊕表示异或门电路；h_i是反馈控制，$h_i \in \{0,1\}$，当h_i=0时，则表示该反馈线不存在，当h_i=1时，表示A_i的输出经h_i反馈线和异或门反馈到移位寄存器的输入。该线性反馈寄存器的特征多项式为：

$$h(x) = x^n + h_1 x^{n-1} + h_2 x^{n-2} + \Lambda + h_{n-1} x + 1 \qquad (5\text{-}2)$$

由上可知，当串联的寄存器组处于全零或者全一的状态时，移位寄存器组将无法产生新的地址输出序列，LFSR电路将一直保持全零或者全一的状态，我们称之为“死状态”。为了顺序产生我们所需要的地址，我们必须使用经过修改的LFSR电路，表5.3给出了一些已经过修改的LFSR电路的使用方法。由表可知，我们只需要从串联的寄存器组中取某几位进行一定的操作，然后输入到串联寄存器组的起始位，就可以顺序产生我们所需要的地址序列。图5.10为有全零状态的长度为12的LFSR电路。

表 5.3 LSFR 反馈位选择查找表

N	XNOR From	N	XNOR From
3	3，2	4	4，3
5	5，3	6	6，5
7	7，6	8	8，6，5，4
9	9，5	10	10，7
11	11，9	12	12，6，4，1
13	13，4，3，1	14	14，5，3，1
15	15，14	16	16，15，13，4
17	17，14	18	18，11
19	19，6，2，1	20	20，17
21	21，19	22	22，21
23	23，18	24	24，23，22，17
25	25，22	26	26，6，2，1
27	27，5，2，1	28	28，25

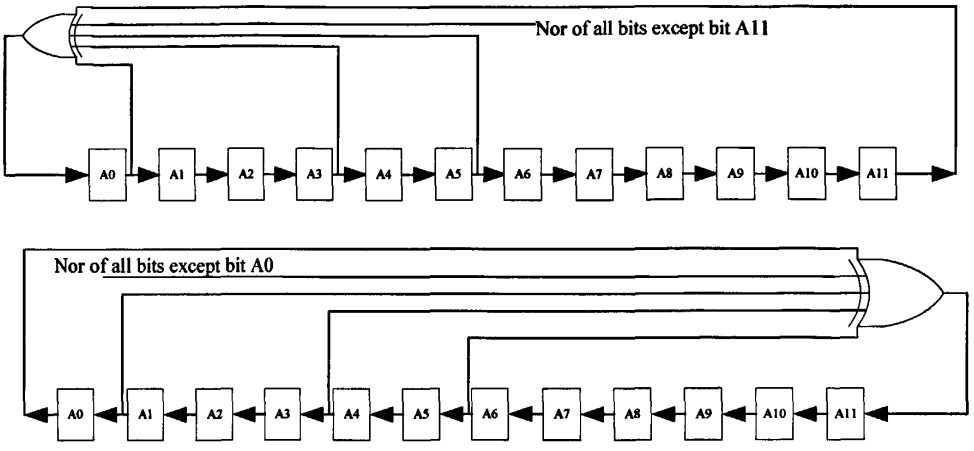


图5.10 有全零状态的长度为12的LSFR

与同样位数的二进制计数器相比，LFSR计数器的的硬件开支较小，且速度也快，但由于March D2PF算法需要对相邻的行和列进行操作，单一的采用LFSR产生地址已不能满足要求，所以地址产生器选取上述两种方案，其中March C+算法采用了LFSR计数器，March D2PF采用了二进制计数器来实现。

基于上面的设计思路，地址产生器模块端口描述为图5.11所示，其端口信号功能描述如表5.4所示。其工作模式为根据算法需要，由测试向量生成器发送ENC_C信号（高电平有效）使得LSFR或二进制计数器开始计数，由ENUP控制其向上计数，而ENDN控制其向下计数，ADDRA[11:0]输出地址，为减少pin脚消耗，测试单端口功能时复用ADDRA[11:0]将读写地址送到存储器的A端口或B端口；当LSFR向上或向下计数到达边界地址时，计数器向控制器发送一个脉冲的UPLMD或DOWNLMD信号，由测试向量生成器模块依据March算法的序列发送下一步操作。当基于March C+算法的所有序列操作完之后，测试向量生成器又会向计数器发送ENC_D信号（高电平有效），这时切换计数器中的二进制计数器开始工作，ADDRA[11:0]和ADDRB[11:0]到达边界地址时，向测试向量生成器反馈边界信号，这时测试向量生成器依据March D2PF算法发送下一步序列的操作。由于设计增加了BISA模块，当有故障发生时，该模块在慢速下输出故障信息，故需要一定的等待周期。此时地址产生器必须进入中断模式，这个等待状态依然由ENC_C和ENC_D这两个使能信号控制，等待状态下地址计数器保持当前状态，直到中断结束时，才恢复计数。

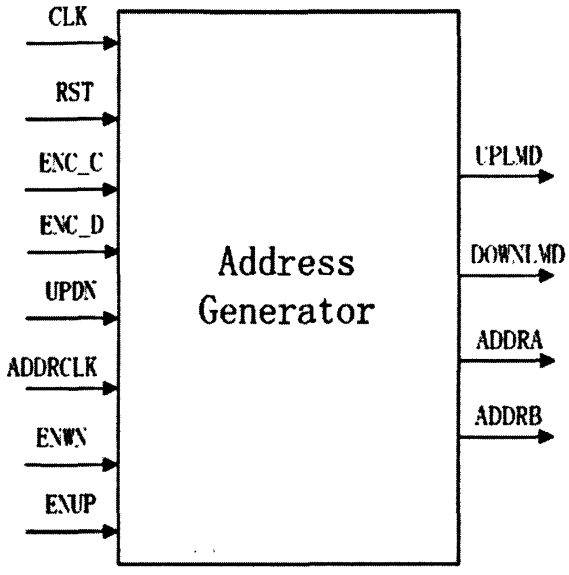


图 5.11 地址产生器端口描述

表5.4 地址生成器引脚说明

引脚名	类型	说明
CLK	输入	测试工作时钟
RST	输入	复位信号
ENC_C	输入	LFSR 地址跟新使能信号
ENC_D	输入	二进制地址跟新使能信号
UPDN	输入	地址递增或递减决定信号（1 为递增，0 为递减）
ADDRCLK	输入	地址更替信号（高有效）
ENWN	输入	地址边界信号（到达下边界）
ENUP	输入	地址边界信号（到达上边界）
UPLMD	输出	地址到达上边界信号
DOWNLMD	输出	地址到达下边界信号
ADDRA	输出	输出读写存储器地址到 SRAM
ADDRB	输出	输出读写存储器地址到 SRAM

5.2.5 数据比较器设计

在端口进行读操作时，比较器把从 SRAM 读出的数据和比较数据产生器的数据（也就是先前写入 SRAM 的数据）进行比较，如果数据一致，则表示 SRAM 没有发生错误。如果数据不一致，则将错误地址和数据通过 FADDR 和 FDATA 发送到 BISA 模块进行保存输出。比较器还通过 TF 信号向向量生成器和地址生成器反馈慢速向外输出的中断信号。其端口描述如图 5.12 所示，端口信号功能描述如表 5.5 所示。

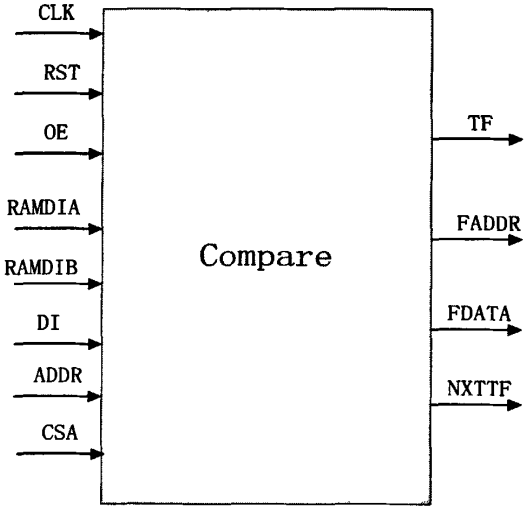


图 5.12 比较器端口描述

表5.5 数据比较器引脚说明

引脚名	类型	说明
CLK	输入	测试工作时钟
RST	输入	复位信号
OE	输入	SRAM 的读使能信号
RAMDIA	输入	A 端口 SRAM 输出数据，用来做比较的数据
RAMDIB	输入	B 端口 SRAM 输出数据，用来做比较的数据
DI	输入	原先输入 SRAM 的数据
ADDR	输入	待比较数据的地址信息

续表 5.5

CSA	输入	SRAM A 端口使能信号(用来区分比较数据是来自 A 端口还是 B 端口)
TF	输出	错误发生标志信号（高有效）
FADDR	输出	错误发生地址信息
FDATA	输出	错误数据信息
NXTTF	输出	错误标志信号下一个状态

5.2.6 BISA 设计

由于嵌入式存储器中晶体管数量巨大而结构规整，收集并分析测试数据对物理缺陷的分析具有重要意义。本次设计与传统 BIST 模块相比，还增加了内建自分析（BISA）模块，本模块的主要功能是接收比较器模块输出的错误指示信息，以及错误地址和数据，并在慢速时钟周期下对外输出这些信息，方便 FPGA 测试条件下逻辑分析仪采集输出信号。

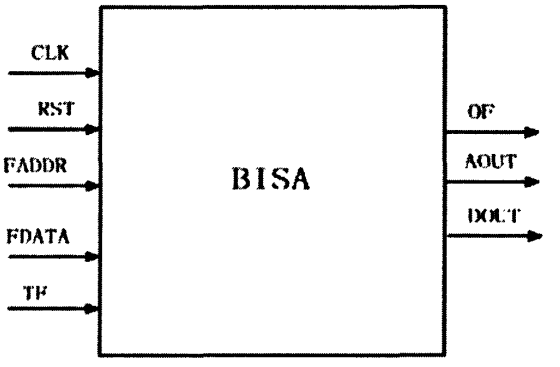


图5.13 BISA端口描述

表5.6 BISA模块引脚说明

引脚名	类型	说明
CLK	输入	测试工作时钟
RST	输入	复位信号
FADDR	输入	错误地址信息

续表 5.6

FDATA	输入	错误数据信息
TF	输入	错误发生提示信号（高有效）
OF	输入	错误向外输出结束信号（高有效）
AOUT	输入	慢速向外输出的地址信息
DOUT	输出	慢速向外输出的数据信息

存储器的故障自分析功能是对BIST输出端输出的故障信息进行分析并最终确定故障模型和位置的过程。BISA 模块将会切换工作在 CLK 这个慢时钟下面，他将向其它模块传递一个保持信号，使得在慢速向外输出错误信息的同时，其它模块处于等待状态而不会继续检查错误，为了保证这个跨时钟域的信号能够正确传输过去，用 OF 信号做了一个握手信号的。错误发生的时候，将输出信号 TF 上拉到高电平，表明存储单元有故障发生，这时数据产生器，地址产生器和比较器的使能信号将被拉低，处于一个等待状态，在慢速时钟下将错误数据输出，然后再将 TF 拉低，OF 信号拉高表明错误信息已经输出结束，其余模块的使能信号得到这个 OF 信号的高电平信号之后，重新回到高电平，使得 BIST 继续脱离等待状态，继续工作，一个周期过后，OF 会自动回复低电平。

前文中已经提到，由于时序采用了并行处理结构，即比较操作的地址将会比读写操作地址的上一个操作地址，又由于共用了地址产生器模块和测试向量产生，所以比较器模块要插入一级寄存器，用以保证拿去跟 SRAM 输出数据（地址）做比较的数据（地址）确实是我们原来所输入的数据（地址）；而对于 BISA 模块来说，由于在比较器模块已经延迟了一个时钟周期，故这里不需要插入寄存器。

第三节 小结

本章系统阐述了 BIST 系统设计的的架构，在传统架构上细分了模块，为了便于获取更多的出错信息，设计增加了内建自分析（BISA）模块，把故障出现的地址和具体出错数据以慢速方式输出；在时序上采用并行处理的结构，实现了全速设计（full speed）。

第六章 ASIC 实现及流片验证

基于上一章对芯片内部结构功能及电路模块的详细介绍，本章将依据 ASIC 芯片设计的标准流程对其后的各级仿真验证过程、综合、静态时序分析、布局布线和流片后测试的相关内容加以具体介绍。同时在各小节中也将对 ASIC 的设计流程同步进行相应说明。

第一节 系统仿真与 FPGA 验证

6.1.1 系统前仿真

使用硬件描述语言 Verilog 将上一章中介绍的电路结构进行行为级 RTL 描述。为了验证每个模块在功能上是否符合要求，我们还需要边写测试代码（即 TestBench），为该模块加上激励信号，通过仿真观察代码输出是否符合功能要求，最后还要把各个功能模块统一成一个顶层文件，仿真验证系统功能。

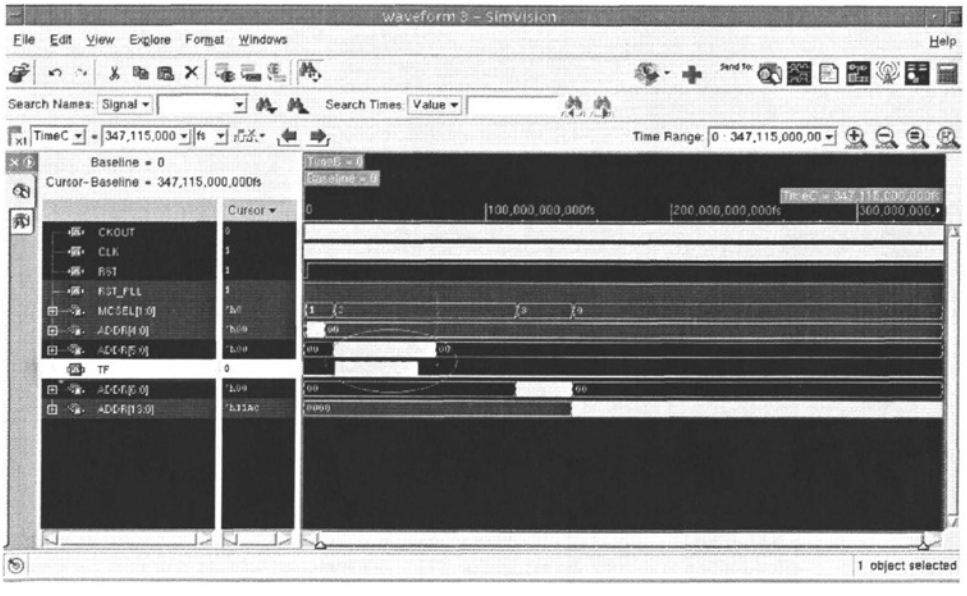


图 6.1 TestChip 的整体仿真

图 6.1 是本设计 SRAM TestChip 的整体仿真，图中我们可以看到由 MCSEL[1:0]分别选择了不同的 SRAM MACRO，其中第二个 SRAM 被人为地加入了错误信息，所以在 MCSEL[2]为一（即选中这个有错误信息的 SRAM 模块）的时候，TF 不再像其它测试其它 SRAM 的时候那样一直为低电平，这时候它一直在低电平和高电平之间转换，表明系统正在记录错误信息并向外输出，实现原先设计的 BIST 和 BISA 功能。

图 6.2 是有错误信息的模块的详细波形。由图中我们可以看到，首先由 CSA 选中 SRAM 的 A 端口，连续发生错误信息，人为加入的错误地址为所有地址，错误的数据位 8'hcc，测试 B 端口的时候没有发现错误。

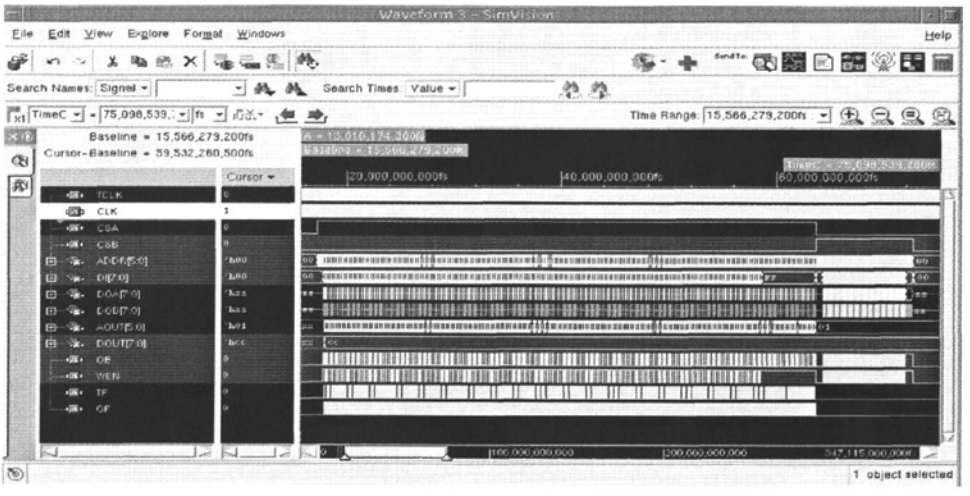


图 6.2 BISA 功能

图 6.3 显示的是 BISA 模块是如何实现慢速输出错误信息并实现使用握手信号实现信号跨时钟域转换的。首先 OE 和 WEN 都是高电平，表明在 TCLK 的一个周期内，对 SRAM 进行读 8'h00，写 8'hff 的操作，然后 WEN 拉低，进行比较，由图中我们可以看出 SRAM 输出的信息是 8'hcc，并不符合我们原来 MARCH 算法所输入的 8'h00 信息，这时候就发生了错误，TF 信号拉高提示错误发生，所有的 BIST 模块得到这个 TF 的高电平状态，将全部处于一个等待状态，一直到慢时钟 CLK 的上升沿到来，将错误地址信息 6'h21 和错误数据信息 8'hcc 向外输出，OF 拉高，提示慢速向外输出完成，TF 又会继续回复低电平状态，BIST 模块将脱离等待状态，继续探测错误信息，如此往复，一直按照所要求的算法

测试完成。

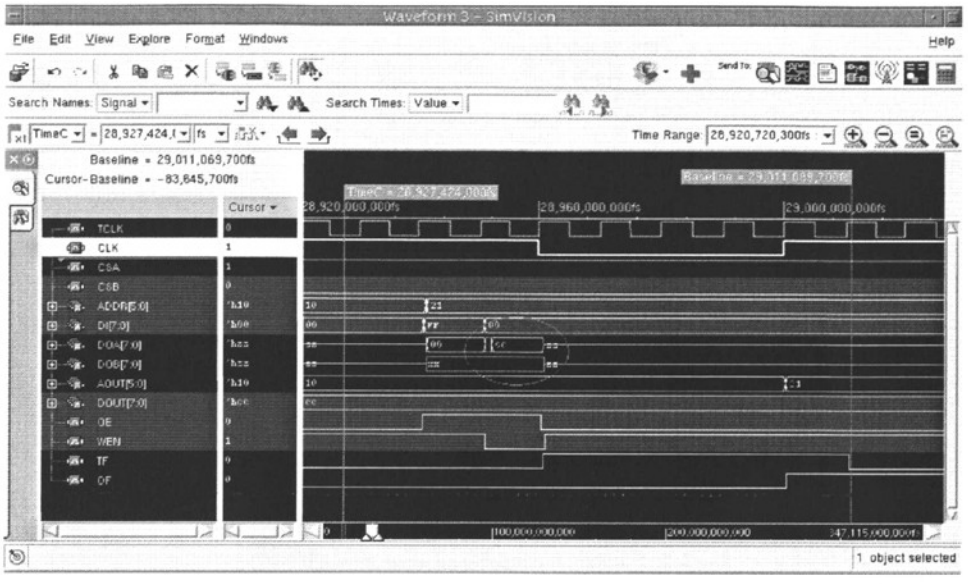


图 6.3 BISA 信号跨时钟域的实现

图 6.4 是 BISA 实现的波形。我们可以看到，错误地址信息和错误数据信息得以连续的保存下来，握手信号 TF 和 OF 也很好的解决了跨时钟域的信号传输问题，保证 BIST 功能的正确实现。

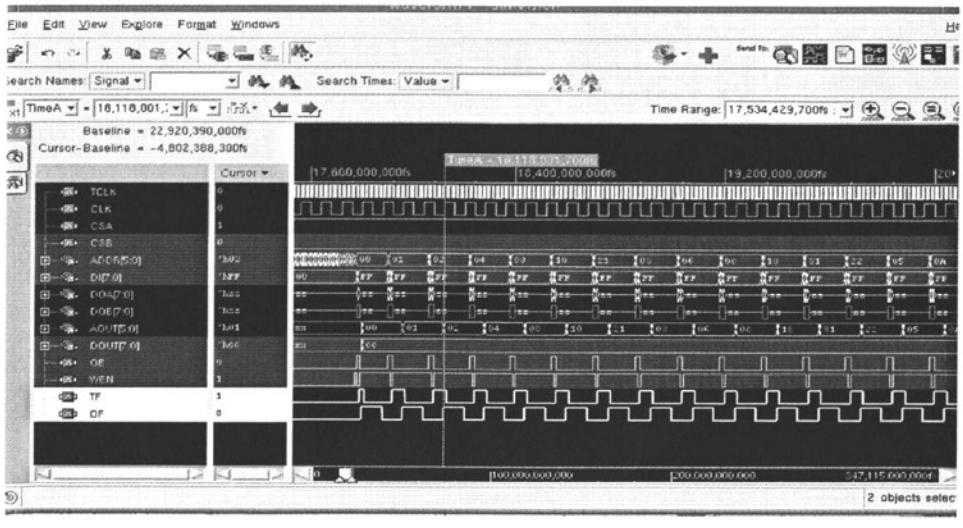


图 6.4 BISA 功能的实现

6.1.2 FPGA 板级验证

对于 BIST 的验证，在设计时可以通过 FPGA 来进行。FPGA 功能强大，调试灵活，而且可根据设计者需要实现其设计。现代集成电路设计中，芯片的规模和复杂度正呈指数增加。尤其在 ASIC 设计流程中，验证和调试所化的时间约占总工期的 70%。为了缩短验证周期，在传统的仿真验证的基础上，涌现了许多新的验证手段，基于 FPGA 的功能验证就是其中之一。目前，基于 FPGA 的设计手段非常灵活，我们通过 FPGA 实现 BIST 验证，可以配套使用许多电子仪器和调试工具，如逻辑分析仪。这些电子仪器的使用也大大加强了设计者对芯片的验证和调试能力。所以 FPGA 在实际硬件上实现芯片设计，可以帮助我们查找并解决在 RTL 仿真阶段不易查找出的问题。

本节讲述了 BIST 设计在 FPGA 的电路实现过程以及板级测试过程。我们采用的是 Altera 公司的 Cyclone EP1C6Q240C8 芯片，采用的开发板为 Eznios360（图 6.5），所用的软件工具为 quartus 6.0，逻辑分析仪是 Tektronix 公司的 TLA5202（图 6.6）。

首先我们用 Verilog 实现了整个 BIST 系统设计。用 quartus 软件进行编译，下载到 FPGA 里面。由于本次 BIST 设计的频率要求是 200MHz，而开发板晶振只提供 50MHz，所以调用了 FPGA 内部的锁相环模块进行倍频达到要求。由于该 BIST 所依托的 IP 核还未曾流片，所以用了另外一颗外部引脚、大小都和本次设计一样的 SRAM 做为测试的对象。

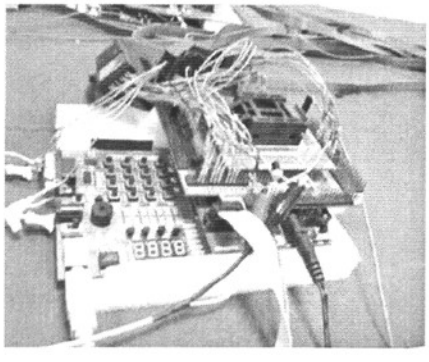


图 6.5 Eznios360 开发板

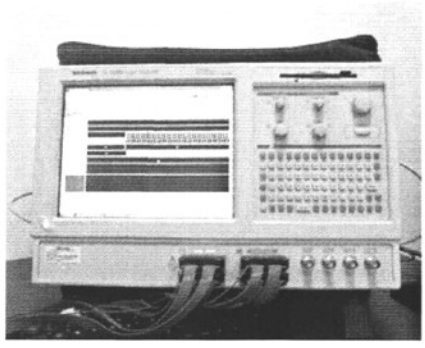


图 6.6 TLA5202 逻辑分析

6.1.3 门级仿真与后仿真

所谓门级仿真，指的是 RTL 级电路通过综合工具综合成为门级网表，并且已经通过布局前静态时序分析（pre-STA）后，对门级电路进行的动态仿真。而所谓后仿真，指的是对门级网表经过布局布线工具和布局后静态时序分析（post-STA）得到的电路网表所进行的动态仿真。虽然这两步仿真在 ASIC 设计流程中分别位于综合和布局布线之后，但是从设计验证方面考虑，将其放在这部分予以介绍。

门级仿真、后仿真相对于前仿真（即功能仿真）而言，主要的区别是在测试激励中所使用的并不是硬件描述语言 Verilog 建立的 RTL 级电路模型，而是需要调用标准单元的门级 Verilog 模型和门级电路网表，同时通过在 Testbench 中使用$sdf_annotate 命令，将门级电路的标准延时文件（.sdf）反标到门级电路中。这样，在门级仿真和后仿真中将带有门级电路的延时信息（Cell Delay）和连线延时（Wire Delay），更加真实的反映了芯片在正常工作模式下的情况。若仿真结果与前仿真结果在功能上完全一致，则可以确定电路设计和内部时序可以满足芯片实际工作的要求。标准单元库通常包括三个工艺 corner 的库文件，即 Fast、Typical、Slow，三种 corner 对应的电路模型区别见表 6.1 所示。因此在理论上，门级网表也对应三种情况，需要分别进行三种 corner 的动态仿真分析。

由于门级网表包含大量的标准单元，相对 RTL 级模型其抽象级别更低，再加上大量的时序信息、延时信息，同时要考虑三种 corner 的情况，造成了门级仿真、后仿真的仿真速度大大下降。对于规模较大的电路设计，更是如此，动

辄就需要几天甚至一两个星期的仿真时间，大大增加了整个芯片的设计周期。

表 6.1 三种不同工艺 Corner 对应的标准单元模型

Corner	工作电压	工作温度	工艺偏差	单元/连线延时
Fast	高	低	小延时倾向	小
Typical	典型	典型	典型	典型
Slow	低	高	大延时倾向	大

从 ASIC 设计验证方法学的角度来考虑，芯片的电路设计在通过前仿真后，只要覆盖率达到一定要求，并且能够顺利通过 pre-STA 和 post-STA 时序分析，那么就可以保证芯片的电路功能和时序方面可以满足设计要求。那么我们为什么还要花费大量的时间和精力来进行门级仿真和后仿真呢？这是因为上面的结论是建立在一定的条件基础之上的，即 pre-STA 和 post-STA 所施加的约束符合电路实际工作的条件或者约束更加严格。但在实际的芯片设计过程中，由于设计人员所施加的约束很难保证 100%满足上面的前提，故而从保证芯片流片成功率的角度来看，我们还需要使用门级网表来进行动态仿真，以弥补我们可能在 STA 中施加的约束不够准确。考虑到电路已经可以完全通过了前仿真，所以在门级仿真和后仿真中，我们就可以不再需要对大量的激励向量进行全面的动态仿真，只需要对基本的电路功能进行分析即可，这样也就可以节省大量的动态仿真时间了。

第二节 电路综合

在完成 RTL 级电路设计，通过了功能仿真/前仿真和 FPGA 验证后，就需要进入 ASIC 设计流程的第二个阶段——电路综合。

所谓综合，是将硬件描述语言（Verilog 或者 VHDL）描述的 RTL 电路模型，转换为门电路结构，即使用芯片工艺厂 Foundry 提供的或者自己设计的基本电路单元（综合库）实现我们用语言描述的 RTL 级的电路功能，这个过程就称为综合。

综合的过程可以细分为三个步骤：

1. 转译（Translation）：读入电路的 RTL 级描述，将语言转译为每条语句

对应的功能块以及功能块之间的拓扑结构，不做任何的逻辑重组和优化。

2. 优化（Optimization）：基于所施加的一定的时序和面积约束条件，综合工具按照一定的算法对转译结果作逻辑重组和优化。
3. 映射（Mapping）：根据所施加的一定的时序和面积的约束条件，综合工具从目标工艺库（Target Technology）中搜索符合条件的单元来构成实际电路。

由 Foundry 提供的工艺库，具有一系列的基本单元，如与非、或非、反相器、锁存器、触发器、选择器等等。对这些单元的电路特性可以进行描述，如：单元的面积、输入端电容、输出端驱动能力、单元的逻辑功能和时序等等。综合的目标就是用工艺库文件提供的这些单元来实现用 RTL 代码描述的逻辑功能，并满足设计者提出的面积和时序要求。

我们使用的综合工具是 Synopsys 公司的 Design Compiler（DC）。综合的过程主要包括设定综合库、读入设计、设定工作环境、设定约束条件、选择综合策略、设计优化、分析并解决出现的问题。综合库及常用环境变量可以在.synopsys_dc.setup 文件中设定，其他综合约束命令可以写到脚本文件（.tcl 文件）中。在综合时通过调用该脚本文件，DC 就可以自动快速地完成对电路的综合过程。综合的方法一般有从上到下（Top-Down）和从下到上（Bottom-Up）两种方法，这两种方法各有个的优势，也有的设计综合采用两种方法，这个一般根据具体设计而定，由于 BIST 设计的规模不大，这里采用了从上到下（Top-Down）的设计方法。

1．电路约束

电路的约束需要考虑设计所处的实际工作环境，原则上是越贴近最后流片出来后芯片的工作状态越好，当然这显然是不太可能的。所以我们需要给电路留出一定的设计裕度，往往需要在约束中施加更加严格的条件，通常情况下可以留出 20%～30%的设计裕度。下面简要介绍芯片顶层的约束脚本。

⑴ 读入 RTL 级代码。

读入 RTL 级设计文件，若存在多个文件，通常应首先读入底层模块。

```
rdv ../rtl/clkGen.v                          //先读各个子模块
rdv ../rtl/addrGen.v
rdv ../rtl/patGen.v
rdv ../rtl/comp.v
rdv ../rtl/DataSave.v
rdv ../rtl/bist.v                            //最后读入整个设计的顶层模块

current_design bist
```

(2) 设置顶层文件、操作环境和连线负载模型。

操作环境是指芯片制造后要工作的实际环境，包括：操作温度、供电电压、工艺偏差等等，因此在综合库中一般提供了最坏（slow）、典型（typical）和最好（fast）三种 Corner 模式，通常在开始综合时选用最坏模式，以使芯片在最苛刻的环境下都可以正常工作。实际电路中，金属连线具有分布的电阻和电容，当连线较长时会对信号的延时产生较大的影响。

```
# set lib & operation condition                                    //设置综合库所用环境
set_min_library fsa0a_c_sc_wc.db -min_version fsa0a_c_sc_bc.db
set_operating_conditions -max WCCOM -max_library fsa0a_c_sc_wc \
                         -min BCCOM -min_library fsa0a_c_sc_bc
# set wire_load                                                    //设置线负载模型
set auto_wire_load_selection false
set_wire_load_mode enclosed
set wire_load_model -name enG5K
```

(3) 建立时钟。

建立时钟可以是施加在实际的基本输入引脚上的，是设计者希望芯片能够运行的时钟，或者是为全组合逻辑创建的一个虚拟时钟。在布局布线阶段建立的时钟树，各时钟的边沿不能够保证完全对齐，即产生时钟偏移（clock skew），若 DC 将时钟作为理想时钟，那么在进行时序分析时就会和实际情况产生误差，故而可以通过设置时钟偏差来接近实际情况。时钟信号往往驱动很多门，因此这个时钟信号的边沿会很差，可以通过插入 BUFFER 的方法来使时钟信号驱动更多的单元门，但是这会使得在不同路径上出现时钟偏移，从而带来时序上的一些问题。Synopsys 推荐的做法是：通过 P&R 工具来建立时钟树，而不是由 DC 根据驱动的门数来决定，从而将 Clock 作为一个理想时钟，以禁止 DC 插入时钟 BUFFER，可通过设置 set_dont_touch_network 命令予以实现。

本设计我们要提供两个时钟一个是从芯片时钟引脚进来的原始时钟，另一个是 PLL 倍频出来之后的高频率时钟。

```
# set clock
set CLK_PERIOD 70.0
create_clock -name  CLK -p $CLK_PERIOD [get_ports CLK] -waveform {0 40}            //原始时钟
create_clock -name CKOUT -p 8 [get_pins FXPLL031HAOA_APGD/CKOUT] -waveform {0 4}    //倍频时钟

set_clock_uncertainty 0.3 [get_clocks CLK]
set_clock_uncertainty 0.3 [get_clocks CKOUT]

set_clock_latency 0.5 [get_clocks CLK]
set_clock_latency 0.5 [get_clocks CKOUT]

set_clock_latency -source 0.5 [get_clocks CLK]
set_clock_latency -source 0.5 [get_clocks CKOUT]

set_ideal_network -no_propagate CLK
set_ideal_network -no_propagate CKOUT

set_clock_transition 0.1 [get_clocks CLK]
set_clock_transition 0.1 [get_clocks CKOUT]

set_dont_touch_network [get_clocks CLK]
set_dont_touch_network [get_clocks CKOUT]
```

(4) 设置输入驱动和输出负载。

驱动值一般用特定的驱动阻抗来设置输入端口的驱动强度，也可以使用某个驱动单元（如：BUF、INV 等）的等效驱动阻抗来进行设置。在综合时通常把时钟信号作为理想时钟，故而设置为无限驱动强度。

输出负载用负载电容来度量，以此告诉综合工具下级电路的负载值为多少，这样综合出的电路才能够为芯片下级电路提供足够的驱动能力。如果负载值取得过小，下级电路可能由于驱动过小而不能正常工作；如果取得过大，则会增加上级电路的实现难度。在此，选择了一个库文件里面一个中等大小的 buf 来作为输出负载和输入驱动。

```
# set loading & driving cell
set_driving_cell -lib_cell INV1 -pin O [all_inputs]
set MAX_LOAD [expr ([load_of fsa0a_c_sc_wc/BUF1/I] * 10)]
set_max_capacitance [expr ($MAX_LOAD/1)] $all_in_ex_clk
set_load [expr ($MAX_LOAD * 4)] [all_outputs]
set_drive 0 CLK
```

(5) 设置输入延时和输出延时。

set_input_delay 可以用来对寄存器路径的输入进行约束，该命令设置了信号

到达当前设计输入端口所用的时间。设电路的时钟周期为 T，如果设置的输入延时为 d0，那么表示从上一级寄存器时钟上升沿到达 D 端所用的时间为 d0，因此从 D 端到本级寄存器的延时最大值为 T-d0，如图 6.7 所示。

同理，对于输出端口可以使用 set_output_delay 来设置信号到达外部寄存器所用的时间。若设置的输出延时为 d1，表示从输出端到达下一个寄存器所需的时间为 d1，因此从本级寄存器送到输出端所用的时间不能够大于 T-d1，如图 6.8 所示。

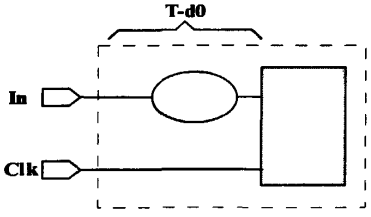


图 6.7 输入延时

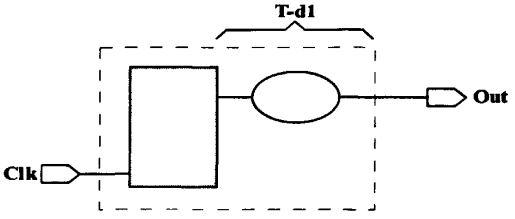


图 6.8 输出延时

```
# set input delays & output delays
set all_in_ex_clk [remove_from_collection [all_inputs] [get_ports "CLK "]]
set i_min_delay 0.2
set i_max_delay 1
set_input_delay  -min $i_min_delay -clock CLK $all_in_ex_clk
set_input_delay  -max $i_max_delay -clock CLK $all_in_ex_clk

set o_delay 2
set_output_delay -max $o_delay -clock CKOUT [all_outputs]
```

(6) 设置无效路径。

设置无效路径，使得在一些路径上的时序约束无效。通常用于设置异步信号和跨时钟域信号。由于我们在设计中已经对异步信号进行过同步处理，故而其不会存在潜在的时序问题，因此可以将其设置为无效路径。系统中有两个时钟域，由于设计上使用了握手信号来保证信号的建立时间和保持时间，所以我们也不需要考虑两个时钟域之间的信号传递，所以也将其设置为无效路径。

```
# set fault path
set_false_path -from RST
set_false_path -from RST_PLL
set_false_path -from [get_clock CLK] -to [get_clock CKOUT]
set_false_path -from [get_clock CKOUT] -to [get_clock CLK]
```

(7) 编译并保存结果及报告文件。

根据上面定义的各种时序和面积等约束条件，进行编译综合。综合后的门级网表可以.v 或者.db 文件的形式保存下来，此外还需要保存标准延时信息（.sdf 文件），以便进行随后的门级仿真。

```
compile -map_effort high  -ungroup

write        -format  db      -hierarchy -output "./db/ethe_top_tt.db"
write        -format  verilog -hierarchy -output "./netlists/ethe_top_tt.v"
write_sdf    -version 1.0                        "./sdf/ethe_top_tt.sdf"
```

(8) 分析报告。

除门级电路及时序信息外，还需要将分析报告文件进行保存，以便随后进行分析以确认综合是否达到预期设计目的。

```
check_design >./report/design.rpt
check_timing >>./report/design.rpt
report_area                                              >./report/area.rpt
report_constraint -all_violators -verbose                >./report/violation.rpt
report_timing -path full -delay max -max_paths 10 -nworst 1 >./report/timing.rpt
report_timing -path full -delay min -max_paths 10 -nworst 1 >>./report/timing.rpt
report_net                                               >./report/net.rpt
```

2. 综合结果

电路综合完成之后，可以通过 DC 生成的报告文件，来检查综合结果是否满足设计要求，包括时序、面积、电路结构等方面。若没有满足设计要求，则就需要重新设置约束条件重新综合，或通过手动修改网表的方式得到所需的门级电路。如果经上述方法仍达不到希望的设计结果，则就需要考虑修改 RTL 级电路来满足整体芯片面积和时序方面的要求。

图 6.9 为综合工具 DC 报告的其中一颗 BIST 的门级电路面积为 2575um^2。

图 6.10 为综合工具 DC 报告的电路违规情况，如时序、输出负载、最大电容等等违反约束条件的情况。从图中可知综合结果无违规情况发生。

图 6.11 为综合后得到的时序报告。

```
****************************************
Report : area
Design : bist
Version: Y-2006.06-SP2
Date   : Sun Sep 28 15:14:05 2008
****************************************

Library(s) Used:

    fsa0a_c_sc_wc (File: /home/project/sd038/sd038ai/simulation/eou/digital/UMC_18_GII_LIB/FSA0A_C_SC_TP_2005Q4v1/

Number of ports:                44
Number of nets:                 70
Number of cells:                 7
Number of references:            6

Combinational area:       1358.000000
Noncombinational area:    1217.000000
Net Interconnect area:      undefined  (Wire load has zero net area)

Total cell area:          2575.000000
Total area:                 undefined
1
```

图 6.9 电路综合得到的面积报告

```
****************************************
Report : constraint
         -all_violators
Design : bist
Version: Y-2006.06-SP2
Date   : Sun Sep 28 15:13:33 2008
****************************************

This design has no violated constraints.

1
```

图 6.10 电路综合得到的违规报告

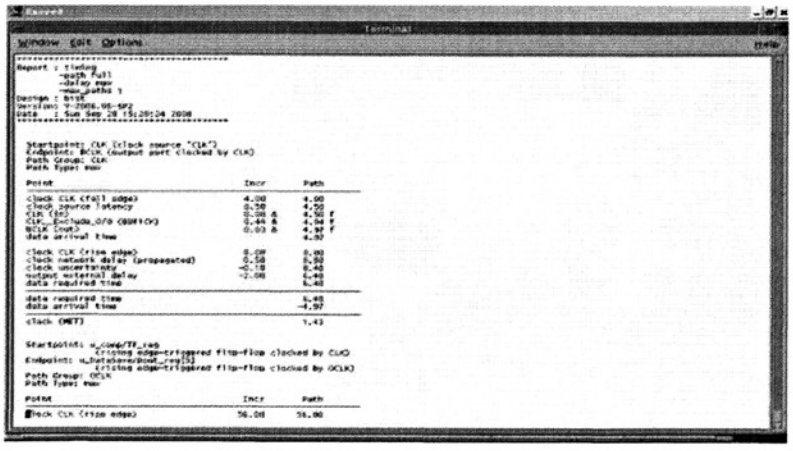


图 6.11 电路综合得到的时序报告

第三节 静态时序分析

静态时序分析 STA（Static Timing Analysis）是设计验证方法中的一个关键要素。电路网表中的每一个存储单元和锁存器都必须满足标准单元库中规定的时序要求。这些要求包括建立时间、保持时间和各种延迟时间（单元延时、连线延时等）。

静态时序仿真的优点是速度快，覆盖率高，因为静态时序仿真是根据 RTL 的拓扑结构考察每一个寄存器的时间。

在芯片设计过程中，电路设计经过综合、扫描链插入、时钟树生成、布局布线等变换过程。在每一个步骤之后都需要进行静态时序分析以确保芯片满足其时序要求。由于我们的设计中并没有进行扫描链插入的操作，而且时钟树生成纳入布局布线阶段，因此实际设计过程中进行的静态时序分析可以分为布局布线前静态时序分析 pre-STA 和布局布线后静态时序分析 post-STA。

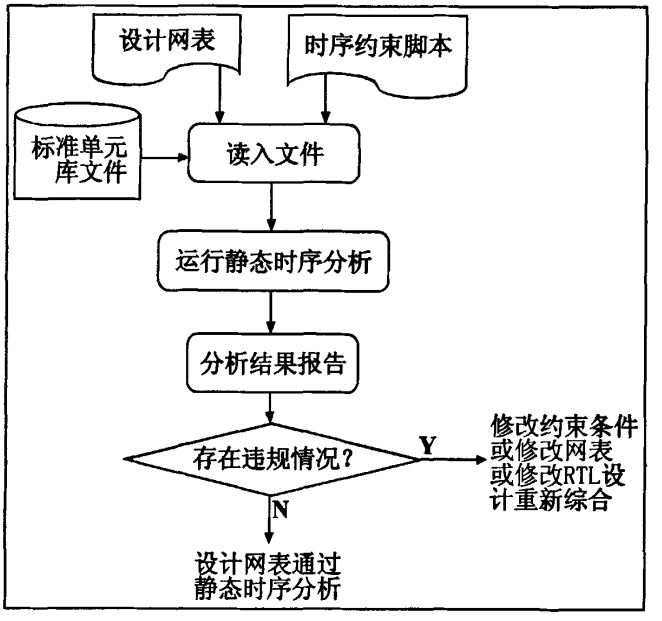


图 6.12 静态时序分析的流程

如图 6.12 所示，静态时序分析按以下基本步骤执行：

(1) 输入。静态时序分析工具需要设计对象的门级网表、标准单元库文

件（.lib 文件）和时序约束脚本文件。对于布局布线后的 post-STA 还需要使用 P&R 工具得到的精确延时文件（.sdf 文件或.spf 文件）；

(2) 读入文件。读入所有需要输入到工具中的文件，以便进行时序分析；

(3) 运行分析脚本。根据约束脚本对输入门级网表和时序加以验证，并以报告的形式生成验证结果；

(4) 分析结果。通过静态时序分析工具所生成的报告，检查设计网表是否满足时序要求。若存在 Violations，则需确定出现违规情况的原因并予以解决。

分析脚本中需要定义以下内容：

- 设计环境选择，需分别在 Fast、Typical 和 Slow 三种 corner 下进行分析；工作时钟的定义，如时钟周期，时钟抖动（Jitter）以及时钟偏差（Skew）；
- 时序约束条件，如建立时间、保持时间等；
- 边界条件，如输入信号延时、输出信号延时、输出端容性负载限制；
- 报告文件的设置。

我们采用的静态时序分析工具为 Synopsys 公司的 PrimeTime，约束分析脚本只需将综合脚本通过 transcript 命令进行转换，再进行简单修改即可。由于布局布线后加入了时钟树的分布，并且通过 P&R 工具得到了更精确的 RC 寄生延时信息，故而在 post-STA 的分析脚本中需删除在 pre-STA 脚本里对时钟抖动和时钟偏差的估计，同时删除之前预估设置的 wire_load 模型。图 6.13 和 6.14 分别为 post-STA 分析得到的违规报告和分析覆盖率报告文件，从中可以看出芯片完成布局布线后的门级网表通过了静态时序分析，而且覆盖率达到了很高的水平，setup、hold 为 92%，recovery、removal 为 92%。

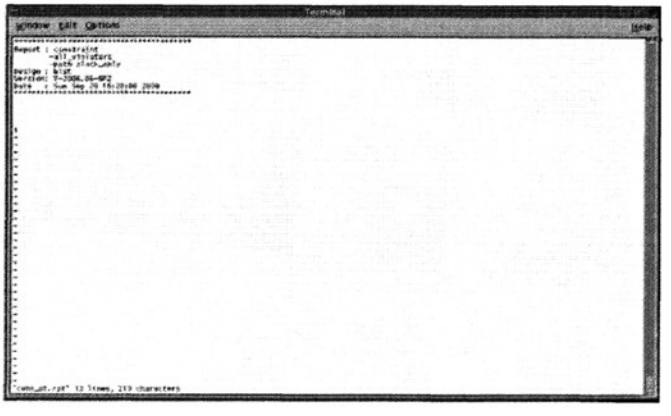


图 6.13 布局布线后静态时序分析——违规报告

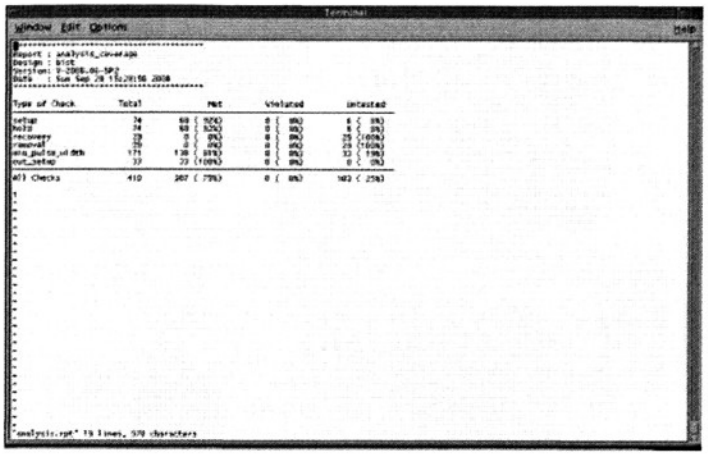


图 6.14 布局布线后静态时序分析——覆盖率报告

第四节 布局布线

6.4.1 电源布线

P/G 网络设计是大规模电路设计中的重要环节，也是布图规划时的重点考虑因素。为了给内核提供足够的电流，必须合理地设置环型电源和带状电源。具体做法是首先在整个内核周围设置一个环形电源，再根据芯片规模的大小，有选择地增加环线或电源带线。电源环尺寸的确定与电迁移和 IR 压降两个因素有关。在多层金属的 CMOS 工艺中，顶层金属带线较厚，对应的方块电阻值最小，因而从理论上说选择顶层金属可使供电环占有较小的芯片面积。然而，金属层的选择还受其他因素的影响。以本设计中的 0.18um CMOS 工艺为例，该工艺共有 6 层金属，而标准单元库中的基本单元都采用第一层金属作为电源线。因此，若在芯片内部使用了顶层金属，就需要从顶层到底层的通孔将电源网线相连，这样做会使得供电网线上的阻值较大，影响供电性能。

一旦选定了金属层，就可以根据各种数据计算满足要求的电源环宽度。考虑到IR压降的影响，从电源焊盘到每个子单元的IR 压降必须保证在0.1VDD 以内，如果超过这个值，就需要通过增加电源带线的线宽来改善这一状况。由于

0.1VDD 的 IR 压降包含了电源带线和电源环的压降，所以电源带线上只允许 0.05VDD 的 IR 压降。一般电源带线两头都接到电源环上的，所以最坏的情况下，通过优化电源线的宽度使得网格中间的 IR 压降满足 Vdrop_strap<5%VDD 即可。

6.4.2 布局

对标准子单元的合理布局有助于面积最小化及减少布线的拥塞，提高整个设计的质量。在设计大规模电路时，需要进行基于时序的布局，对每条路径作时序分析，以减少因不满足时序要求而进行的迭代次数。虽然综合时已通过线负载模型对网表进行了优化，但逻辑上的层次关系和实际的布局信息并不相同。为了实现时序和面积的优化，需要将布局后实际的版图信息返标到综合工具 DC 中，使 DC 通过读取接近实际情况的布局信息优化电路的延时，这样做可以综合出更好的设计结果，综合时尽可能兼顾到电路的拥塞情况，让电路结构和布局在时序和拥塞两方面都能得到满足，从而达到最优。

布局时我们还要综合考虑存储器硬核单元的摆放位置和 Pin 脚朝向，以利于最后的布线时序，必要时还需要划分具体的边界条件，指导工具在指定的位置摆放指定的标准单元，总之所有操作的目的都是希望最后布线步骤中，令线上延迟和串扰所导致的时序变差的影响变到最小。

6.4.3 时钟树综合

在同步设计中，单个时钟源信号要驱动众多寄存器，因其带载能力的问题，时钟源信号不可能直接驱动寄存器。并且，也会因为从时钟源到时钟端的布线过长带来时延过大，所以现在时钟网络的分布方式大都采用时钟树的架构，即在时钟源与时钟端之间插入若干缓冲器，形成树架结构。图 6.15 和图 6.16 分别给出了时钟 BUFFER 插入前后的时钟网络图。在时钟树综合后的时钟网络中，由于 BUFFER 的插入，使时钟源到时钟端的延迟（Clock Latency）和时钟波形的偏差（Skew）大大减小，并且时钟源到每个时钟端的时间差很小，这样，实际的时钟波形能够尽量和理想时钟接近，电路能够同步运作。所以，时钟树综合的主要目的是尽量缩小时钟源到每个时钟端的时间差异，提高电路同步功能。

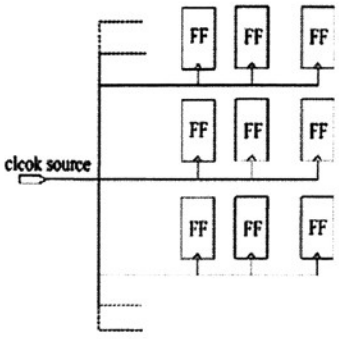


图 6.15 时钟 BUFFER 插入前的时钟网络

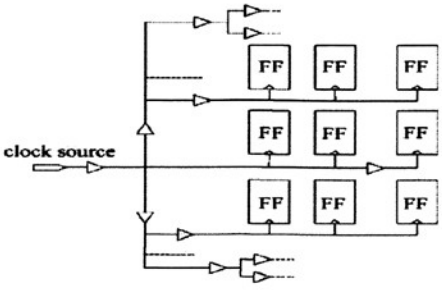


图 6.16 时钟 BUFFER 插入后的时钟网络

6.4.4 布线

布线时首先留出模拟信号用的走线和隔离空间。然后考虑时钟树的布线，可以选择用传输速度较快的金属层作时钟树布线。最后是数字信号布线。对主要关键路径的布线，可以适当地给此线加权。

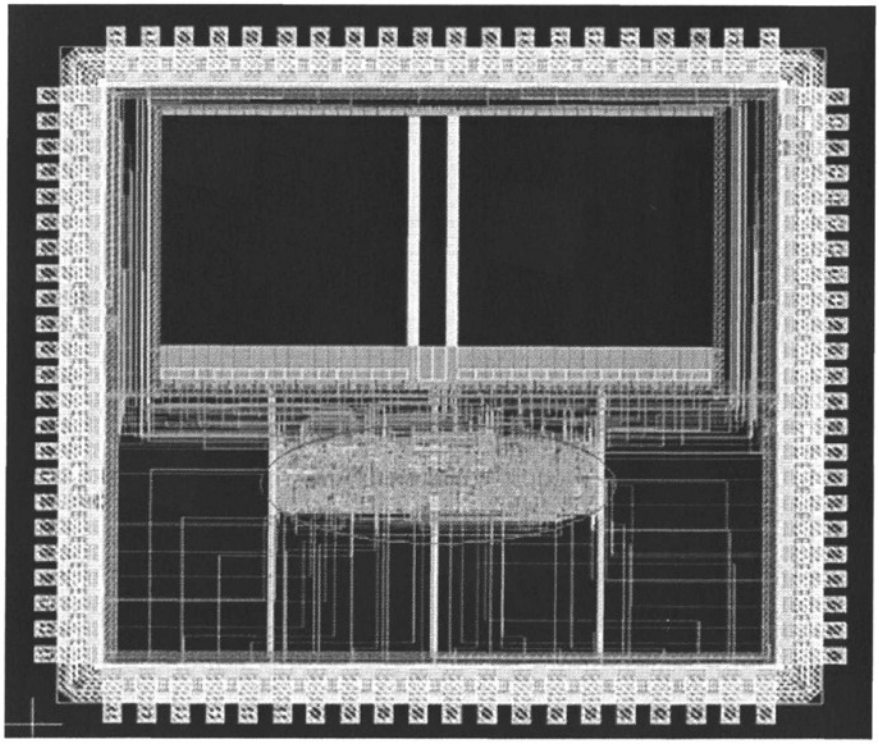


图 6.17 memoryBIST 芯片版图

布线完成之后，我们还需要进行 DRC（design rule check）和 LVS（layout vs schematic）检查，以满足工艺厂商流片的要求，为最后流片成功提供保证。

布局布线完成之后，会生成新的网表文件和标准延时信息文件。如我们在上文中所介绍的在 P&R 之后仍需进行 post-STA 和后仿真。若静态时序分析和动态仿真的结果都满足设计要求，就可以把最后生成的 GDS II 文件送交工艺厂进行流片。如果在 post-STA 或者后仿真中出现错误，那么我们就必须重新进行布局布线，或者返回 DC 进行重新综合，必要时甚至需要修改 RTL 代码，以满足版图后仿真的需求。

图 6.17 为最后已经完成布局布线工作的版图，图中上半部分两个黑框是放置 SRAM 的位置，下部分空白处放置 PLL，图中展现的是本次工作所完成的 BIST 部分的设计，如图中红线所圈出来的部分。至此，整个基于 ASIC 流程的 memoryBIST 芯片的设计阶段已经全部完成，接下来就可以等待芯片流片返回，最后进入芯片的测试阶段。

第七章 结论

随着半导体工艺尺寸不断缩小，IC 设计的规模越来越大，高度复杂的 IC 产品正面临着高可靠性、高质量、低成本以及更短的产品上市周期等日益严峻的挑战。一方面随着半导体工艺尺寸的缩小，嵌入式存储器可能存在的缺陷类型越来越多；另一方面，随着 IC 产品的复杂度的提高，嵌入式 SRAM 在 IC 产品中的比重越来越大。嵌入式 SRAM 大量使用显著提高了 SOC 的性能，但也给芯片的设计、制造，特别是测试，带来了巨大的挑战。为了解决测试问题，存储器内建自测试（memoryBIST）得到了广泛使用。

本文针设计了一个对四颗不同容量的SRAM进行测试的memoryBIST电路。具体来说：

1. 在深入分析了 SRAM 工作时序，存储器故障模型以及测试算法的基础上，选定了了 March C+和 March D2PF 算法组合来进行测试，提高了最终测试的故障覆盖率。

2. 分析比较了两种传统 BIST 时序设计在测试时间上的劣势，针对本次设计中 SRAM 可以在同一个周期中进行读写操作的特殊性，提出了一种全新的方案作为本次 BIST 系统的时序，最大限度保证了全速测试。

3. 由于设计本 IP 核的目的是为做测试出更多的 SRAM 出错信息，反馈给 SRAM 设计人员，为 SRAM 设计改进提供更多帮助。因此在传统 BIST 架构的基础上细分了功能模块，增加了内建自分析（BISA）模块，支持故障信息在较慢的时钟频率下输出，方便流片后的 FPGA 片上测试时通过逻辑分析仪抓取错误信息。

4. 基于 Verilog 语法编写了各模块的代码，并通过了 FPGA 验证、综合静态时序分析、版图设计等一系列设计流程。最终 BIST 版图增加的面积不超过原嵌入式存储器面积的 10%。

由于作者水平有限，同时又受到开发时间、需求等各种条件的限制，本文的设计仍有一些不足之处。主要表现为

1. 双口 SRAM 故障产生机制非常复杂，现有的 March D2PF 算法不能覆盖所有的双端口故障。

2．March C+和 March D2PF 算法组合测试提供了较高的故障覆盖率，但是这种组合以牺牲测试时间为代价。

3．BISA 部分慢速输出时使用的是强制 BIST 探测模块处于等待状态，这样虽然可以支持慢速输出错误信息，可是大大牺牲了测试时间。

基于以上的不足，在以后的改进中，可以主要围绕以下几个方面展开：

1．集成电路向深亚微米发展的趋势，存储器失效机制将越来越复杂，特别是双端口故障。提出新的算法或在现有算法上改进的 BIST 设计，满足故障覆盖率的要求。

2．对于双端口测试来说，需要几种算法联合测试，导致有些故障重复测试，这样增加了测试时间和设计的复杂度。如果在深入分析故障模型的机理上，将几种算法有效的结合起来，剔除故障的重复测试，可以有效减少测试时间，降低设计的复杂度。

3．如何寻找一种更加有效的方式支持慢速输出，是否可以在支持 BISA 慢速输出的同时，并不需要 BIST 模块处于等待状态。

4．尝试在设计中加入内建自修复（BISR Build-In-Self-Repair）的功能，在系统定位到错误故障的具体位置之后，能自动地进行故障修复，提高自测试系统的智能化。

参考文献

[1] SIA.ITRS.2002.

[2] Said Hamdioui and Georgi Gaydadjiev. Future Challenges in Memory Testing. Delft university of Technology, Faculty of EEMCS Computer Engineering Laboratory, Mekelweg 4, 2628 CD Delft, The Netherlands. 2004.

[3] Alfred L.Crouch. Design-For-Test For Digital IC and Embedded Core Systems. 北京：中国电力出版社，2004. 175－231

[4] D.Adams, High Performance Memory Testing, Kluwer Academic Publisher, ISBN: 1-4020-7255-4, 2003.

[5] P. Camurati, et.al, "Industrial BIST for Embedded RAMs", IEEE Design & Test of Computers, Vo. 12, No. 3, pp. 86-95, 1995

[6] 杨士元. 数字系统的故障诊断与可靠性设计. 北京：清华大学出版社，2004. 145-152

[7] 雷绍充 邵志标 梁峰. VLSI 测试方法学和可测性设计. 北京：电力工业出版社，2005.146－177

[8] A.J.Van de Goor. Testing Semiconductor Memories: Theory and Practice. Chichester, UK：John Wiley&Sons, Inc. 1991.

[9] 向东. 数字系统测试及可测性设计. 北京：科学出版社. 1997

[10] R. Dekker, et al., "A Realistic Fault Model and Test Algorithms for Static Random Access Memories", IEEE Trans. on Computers CAD, Vol. C9, No. 6, pp. 567-572, 1990.

[11] V.C. Alves and M. Nicolaidis, "Detecting Complex Coupling Faults in Multi-Port RAMs," IMAG Research Report No. RR978, Feb. 1991

[12] M. Nicolaidis, V.C. Alves, and H. Bedrr, "Testing Complex Coupling Faults in Multi-PORT Memories," IEEE Trans. VLSI Systims, vol.3, no. 1,pp.59-71, Mar. 1995

[13] A.J. van de Goor and S.Hamdioui,"Fault Models and Tests for Two-Port Memories," Proc.16th IEEE VLSI Test Symp.,pp. 401-410, April 1998.

[14] Rei-Fu Huang，Yung-Fa Chou and Cheng-Wen Wu. Defect Oriented Fault Analysis for SRAM. IEEE 12th Asian Test Symposium. 2003

[15] S.Hamdioui and A.J.van de Goor. An experimental analysis of spot defects in SRAMs：realistic fault models and tests，in Proc.Ninth IEEE Asian Test Symp. (ATS)，Taipei.2000

[16] Hamdioui，Said and A.J.Van de Goor. Efficient Test for Realistic Faults in Dual-Port SRAMs. IEEE Transaciions on Computers. 2002, VOL.51.NO.5

[17] Hamdioui Said，RodgersMike，van de Goor A.J.et. March tests for realist ic faults in two-port memories. Recoders of the 2000 IEEE International Work shop on Memory Technology，Design and Testing. 2000

[18] A.J.Van de Goor and C.A.Verruijt. An Overview of Deterministic Functional RAM Chip Testing. ACM Computing Surveys. Mar, 1996. vol 22, no. 1.

[19] A.Jee，J.E.Colburn，V.S.Irrinki and M.Puri. Optimizing memory tests by analyzing defect coverage.in Proc IEEE Int.Workshop on Memory Technology, Design and Testing (MTDT). San Jose.2000

[20] A.J. van de Goor. Using March Tests to Test SRAMs. IEEE Design & Test of Computers，vol.10，no. 1. Mar，1993

[21] Daniel Milton. An Efficient Built-in Self Test (BIST) Methodology for Testing Configurable Embedded Memories in FPGAs：A Case Study. Auburn: Auburn University

[22] S. Hamdioui, A.J. van de Goor and M. Rodgers, "March SS: A Test for All Static Simple RAM Faults", In Proc. of IEEE International Workshop on Memory Technology, Design, and Testing, pp. 95-100, Bendor, France, 2002

[23] Said Hamdioui, Associate Member, IEEE, and Ad J.van de Goor, Fellow, IEEE. Efficient Tests for Realistic Faults in Dual-Port SRAMs. IEEE TRANSACTIONS ON COMPUTERS, VOL.51, NO.5, MAY 2002

[24] Peter Alfke. Efficient Shift Registers, LFSR Counters, and Long seudo-Random Sequence Generators XAPP 052 July 7,1996 (Version 1.1)

[25] Huang Der-Cheng，Jone Wen-Ben. A parallel built-in self-diagnostic method for embedded memory arrays. IEEE Trans. onComputer-Aided Design of Integrated Circuits and Systems. 2002-04. 21(4)

[26] 王晓琴，黑用，吴斌，乔树山. 嵌入式存储器 MBIST 设计中内建自诊断功能究. 2005.12，电子器件，28(4).

[27] Lei S C，Hou X Y，Shao Z B，et al. A class of SIC circuits：theory and application in BIST design[J]. IEEE Trans Circuits Syst II，2008，55(2)：161—165.

[28] David R，Girard P，Landrault C，et al. On using efficient test sequences for BIST [c]. IEEE VLSI Test Symposium 2002. Monterey，California，USA ：IEEE CS Press，2002：145. 150.

[29] Corno F，Rebaudengo M，Reorda M S，d al. Low power BIST via non·linear hybrid cellular automata[c]. IEEE VLSI Test Symposium 2000. Los Alamitos，California，USA ：IEEE CS Press，2000：29-34.

[30] 陈新武. 集成电路测试方法研究，华中科技大学内部培训材料.

[31] Sankaralingam R，Oruganti RR，Touba NA. Static compaction techniques to control scan vector power dissipation Ec]. IEEE VLSI Test Symposium 2000. Los Alamitos，California，USA ：IEEE CS Press，2000：35-40.

[32] 姚其爽. 高速低功耗嵌入式 SRAM 研究与设计. 西北工业大学硕士学位论文.

[33] 夏宇闻，Verilog 数字系统设计教程.北京：北京航空航天大学出版社，2003 年

[34] Design Compiler User Guide. by Synopsys. Version Y-2006.06.

[35] PrimeTime User Guide: Fundamentals. by Synopsys. Version Z-2006.12.

致谢

本论文是在导师俞梅副教授以及苏州秉亮科技公司存储器部门经理郑坚斌先生的指导下完成的。两位导师精严谨踏实的治学态度、循循善诱的导师作风、平易近人的做人风范，将使我终生受益。在我三年的研究生生活期间，俞老师在学习、工作、生活等各方面给予了无微不至的关怀和细致入微的指导。在此特向导师俞梅副教授致以最崇高的敬意和最真诚的感谢。在苏州工作实习的一年时间，部门经理郑坚斌先生在我从事 IC 设计工作的道路上给予了导师般的指引，两位导师丰富的理论知识以及实践经验给了我很大的帮助，在此表示衷心的感谢。

特别感谢秉亮科技公司为本论文的完成提供良好的工作条件与环境，在这里，我学到了很多宝贵的知识，对 IC 设计有了更清晰的认识，为我将来从事 IC 设计行业奠定了坚实的基础。

还要感谢公司同事资深 SRAM 设计师于跃，姚其爽在 SRAM 设计知识方面给予的大力帮助。感谢公司存储器内建自测试设计的前辈程沁，本次设计是在他的设计基础上进行改良和完善的。本论文的 FPGA 验证工作在秉亮科技测试部门韩惠方的大力协助下完成的，设计的后端布局布线部分由秉亮科技数字后端部门江利民完成。衷心感谢所有在本次设计中付出努力的老师和同事，他们无私耐心的指导和帮助使本论文得以顺利完成，在此深表谢意。

由于水平有限，论文中肯定存在许多不妥和错误之处，恳请指正。

区晓云

2009 年 5 月

个人简历

区晓云，1982年2月21日生于广西省河池市。
1999年9月至2003年6月就读于南开大学信息技术科学学院微电子科学系，获得理学学士学位。
2006年9月至2009年6月就读于南开大学信息技术科学学院微电子与固体电子学专业，方向为超大规模集成电路设计，主攻数字集成电路设计，师从俞梅副教授。

在学期间发表的学术论文和研究成果

2008年3月至2008年10月参与苏州秉亮科技公司基于memoryBIST的SRAM TestChip设计，该设计是基于March C+/March D2PF 算法的memoryBIST，以及包含memoryBIST的SRAM TestChip芯片设计。 芯片是基于FSA0A_C 0.18 标准单元库，待测SRAM为4个同步双端口SRAM（FSA0A_C_SJ），芯片使用ASIC的标准流程，实现了全速测试（200MHz）SRAM的基本读写功能，自动侦测错误地址和数据并向外输出。该设计作为本人毕业论文课题，已通过系统后仿真，功能正确。

2007年8月至2008年1月参与苏州秉亮科技公司全定制memoryBIST设计，参与设计基于UMC 45 nm SP RVT 1P10M low-K process的全定制memoryBIST设计，芯片包含16个单端口同步SRAM MACRO， 4个BIST电路， 和1个 ADPLL。 芯片实现功能包括测试SRAM的Set-up Time, Hold-Time，和TA（Access Time），使用ALL0/ALL1，checkboard/icheckboard算法实现全速测试（1GHz）SRAM的基本读写功能。负责设计用于生成不同延迟CLK的时钟生成模块，用于调option和选择MACRO的MUX模块，以及BIST部分的电路。芯片后仿真通过(ss corner > 800MHz)，已经交付流片。