密级:_____

中国科学院研究生院

硕士学位论文

Yield Explorer 故障诊断软件的设计与开发

作者姓名: 郭浩

指导教师: 胡瑜 研究员

中国科学院计算技术研究所

学位类别: _____

学科专业: _____

培养单位: 中国科学院计算技术研究所

2012 年 5 月

Design and Implementation of the Yield Explorer Fault Diagnosis Software

By

Hao Guo

A Dissertation Submitted to
Graduate University of Chinese Academy of Sciences
In partial fulfillment of the requirement
For the degree of

Master of Engineering

Institute of Computing Technology Chinese Academy of Sciences

May, 2012

声　　明

我声明本论文是我本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，本论文中不包含其他人已经发表或撰写过的研究成果。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

作者签名：郭浩　　日期：2012.5.17

论文版权使用授权书

本人授权中国科学院计算技术研究所可以保留并向国家有关部门或机构送交本论文的复印件和电子文档，允许本论文被查阅和借阅，可以将本论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编本论文。

（保密论文在解密后适用本授权书。）

作者签名：郭浩　导师签名：胡瑜　日期：2012.5.17

摘要

随着集成电路制造工艺步入超深亚微米(very deep submicron,, VDSM)和纳米(nanometer)阶段，芯片的缺陷密度不断上升，快速的成品率学习(yield learning)有助于迅速提高芯片的成品率，而成品率学习过程中的关键步骤就是故障诊断(fault diagnosis) 。故障诊断的目的是确定电路中故障的位置并判定故障类型，为后续的物理失效分析（physical failure analysis）提供信息。因此，故障诊断一直是研究领域的热点。相应的故障诊断软件开发，也就具有重要的实践意义。本文设计开发的故障诊断软件，集成了新的故障诊断算法，具备较好的扩展性，利于升级，且方便用户的操作，也可以作为新的诊断算法的评估平台。

本文分析了现有的故障诊断方法，对现有的成熟的故障诊断软件进行了对比分析，设计开发了 Yield Explorer 故障诊断软件，编写了软件的代码。本文的主要贡献包括：

1. 开发了一种友好的界面设计和操作流程。本文对工业界提出的常用软件进行了深入分析，结合这些软件的优点和特性制定了一种故障诊断软件的界面结构和交互方法。该方法使得用快速有效的软件半自动地进行故障诊断成为可能。整体软件流程比较简单，不易出错。用户可以自定义相关输入文件，灵活性较高。而且诊断结果有存档，易于查询和分析，并能通过统计图显示。

2. 设计了一种诊断软件的模块结构。本文中的诊断软件使用了模块化的设计。主要包括操作接口模块、文本编辑模块、输入解析模块、核心算法模块、结果展现模块。每个模块用主要的类来进行更细致的功能划分。各个模块之间的藕合性较低，利于对模块进行替换。同时由于用类进行功能的对应，逻辑关系清晰，新功能的添加可通过添加对应类的类函数来实现。对每种模块，提出了具体的特性要求。

3. 对软件的设计进行了完整的 C++代码实现。对于每个模块，由于特点不同，使用了不同的开发工具并利用相关函数库实现了实用的具体功能。对故障诊断核心算法也进行了不同地实现。既开发实现了经典的故障诊断算法——注入与评估（inject and evaluate）方法，也在 Yield Explorer 中集成了面向多故障诊断的新算法——基于故障元组等价树的诊断方法。全软件用 C++语言进行了完整实现，可以在 Linux 平台下进行安装使用。同时本软件接受商业领域较为常用的输入文件格式。

关键字： 故障诊断，软件设计，故障模拟，多故障假设，软件开发

Abstract

Design and Implementation of the Yield Explorer Fault Diagnosis Software

Hao Guo (Computer Architecture)

Directed By Prof. Hu Yu

As the IC manufacturing process steps into the very deep submicron (VDSM) and nanometer stage, the defect density of chips is rising fast. So effective yield learning would contribute to the rapid increase of the rate of good chips. Moreover, fault diagnosis is a critical step in the process of yield learning. The purpose of fault diagnosis is to determine the location of the fault in the circuit and determine the fault type to provide information for subsequent physical failure analysis. Therefore, continuously developing the digital IC fault diagnosis software is necessary to improve the quality of fault diagnosis. The software in this paper has good scalability and integrates a newly proposed diagnosis algorithm, which can also be used as an evaluation platform of new diagnosis algorithms.

This paper analyzes the existing fault diagnosis methods, and conducts a comparative analysis of the existing commercial fault diagnosis software. A practical fault diagnosis software design was proposed and the software was implemented. The main contributions of this paper include:

1. Propose a friendly interface design and a interactive method

Through the in-depth analysis of the software used in industry, the author develops a fault diagnosis software interface and a interaction method combining the advantages of those software tools. This interaction method allows a quick and efficient semi-automatic professional fault diagnosis by software. The overall flow of process is relatively simple and less error-prone. Input files can be modified at any time, which creates higher flexibility. And diagnostic results will be archived, which can be easy to query, and can be displayed by charts.

2. Design an architecture of fault diagnosis software.

Diagnosis software in this article uses a modular design, including interface module, text editor module, input analysis module, core algorithm module, and results showing module. Each module consists of several classes to carry out a more detailed division of features.

Lower coupling between the various modules will help to replace the module. The logical relationships between classes and features are clear due to effective division of features into classes. New features can be added by filling class functions into corresponding classes. Features for every module were designed.

3. Implement the whole software in C++.

The author implemented each module specifically with different methods according to its features. Some of the algorithms used in this software are classical algorithms (such as inject and evaluate method) and could be used widely. Some of the algorithms (such as fault tuple equivalence tree-based method) are proposed by the author's research group. The software in this article is implemented in C++ language, which can be installed in Linux platform. It accepts the commonly used input file formats.

Keywords: Fault diagnosis, software design, fault simulation, multiple faults, software development

目 录

图目录

表目录

算法目录

第1章 引言

集成电路的工艺在不停地进步中，芯片的特征尺寸随着时间持续地缩小，芯片的规模和复杂度不断增加，片上晶体管数目已达到十亿量级[1]，故障诊断（fault diagnosis）已经成为集成电路设计链中极具挑战性的一环。同时，电路的故障诊断在数字电路设计和生产过程中具有重要意义，它有助于有助于生产工艺的改进，有助于分析故障测试方法的效果，为芯片修复提供信息等，并最终提高芯片的产量、质量以及可靠性。故障诊断软件作为一种电子设计自动化（Electronic Design Automatic，EDA）工具，可有效地减轻相关从业人员的负担，快速准确地自动完成一部分功能，所以，研究和开发友好易用的故障诊断软件就具有较高的现实意义。

1.1 课题的选题背景及意义

受到生产工艺以及制造过程的影响，在制造中难免会引入一些物理缺陷，导致芯片内部存在故障。其中某些种类的故障可能导致芯片不能正常工作。芯片厂商为了使出产的芯片满足正常可使用的标准，需要对芯片进行测试以筛选出有故障的芯片。芯片测试之后，解决了辨别故障芯片的问题，但没有指出具体的故障类型和故障原因。因此芯片测试之后需要进行故障诊断，判定具体的故障类型和故障原因。传统的故障诊断往往是单故障假设，通过对比实际失效响应和单故障状态下的失效响应来判断芯片的故障在什么位置[2]。

大约从 1960 年代开始，由美国领导潮流，进行系统的故障诊断技术的研究，之后，故障诊断的研究在世界上广泛地传播开来。故障诊断技术如今已成为芯片制造的关键因素。芯片的故障诊断技术与制造技术受到关注的时间大致一样，但故障诊断技术的进步不如制造技术显著。最初，故障诊断主要是工程人员利用有限的工具通过自己的经验来人工进行。这种方法对工程人员的实际工作经验与相关知识积累的程度有很大程度的依赖，同时人工进行的工作速度慢，较易因疲劳和枯燥导致出错。现代电路规模不断加大，人工出错的现象更不可避免。为解决这个问题，只能进行方法的升级，研究如何用机器自动化地代替人类进行如此庞复的工作。近年不断升级的高速计算机系统让研究人员有了可供利用的物质平台，使得研究利用它运算的算法与软件有了现实可能。在微电子产业发展的过程中，模拟电路比数字电路更早被提出和使用，不过数字电路的发展与普及明显更迅速，最容易说明这个现象的例子就是数字计算机对模拟计算机的取代和它如今越来越高速地发展。这种现象的主要原因除了大型工业生产部门中有对数字集成电路的需求外，制造简单，容易提高集成度，可靠性高，诊断技术自动化程度高等因素都是其

发展的重要原因。故障诊断算法的电路模型是一个黑箱，即被测对象是一个无法打开的箱子，只可以利用输入输出来进行检测。这种情况下，唯一的方法就是输入一组精心选择的输入，根据输出信号来推断电路中存在的故障。伴随诊断方法的推陈出新，高速计算机处理速度的快速升级还有硬盘容量的不断扩大，数字电路诊断软件的实用性快速发展，诊断问题被极大地降低了难度。数字电路诊断软件的发展进入快车道。

随着现代集成电路工艺的改进，芯片的特征尺寸逐渐缩小，电路的规模和复杂度也增加，故障的密度快速增加，研究人员也提出了很多故障诊断算法[3-8]。但为了保证芯片的质量，依然需要对故障模型和诊断算法进行改进。

传统的基于单故障假设的故障诊断算法，运行快速，给出的候选故障也较少，有利于失效分析人员对芯片进行检查。但是，对于现代设备来说，芯片故障多呈现出聚集和影响范围广的特点。所以单故障假设不能描述现代芯片的故障特点。因此基于复合故障和多故障假设的诊断算法成为近年的研究热点。在这些诊断算法中，一部分限定了故障模型[9]，只能基于这些故障模型进行诊断；另一部分算法则不基于特定的故障模型[10]。本人所在研究组开发的基于多故障假设的基于故障元组等价树的诊断算法[11]没有假定故障模型，所以适用范围较广。在此基础上，利用主流的软件开发工具和方法，就有可能开发出友好易用的面向多故障的故障诊断软件。

本文开展的研究工作受到了国家自然科学基金面上课题“面向多个物理缺陷的纳米尺度极大规模数字电路故障诊断” （61076018）的资助，取得的主要成果和贡献是设计开发了一款故障诊断软件 Yield Explorer。

1.2 本文的主要内容和章节安排

本文所涉及的工作主要针对故障诊断工作设计实用的故障诊断软件。解决故障诊断软件涉及的主要功能的取舍问题以及操作流程制定问题。并考虑软件的界面设计和交互设计，同时实现有效的编辑功能。利用已有代码或自行编写诊断相关功能，解析文件信息。

本文的具体结构说明如下：

第一章是本文的引言，介绍了文中研究工作的背景和意义。

第二章是故障诊断算法的分类与方法介绍。介绍了故障诊断软件中的基本概念，组合电路和扫描链的诊断算法。对其中的有代表性的算法进行了详细阐述。

第三章针对故障诊断软件的发展趋势，提出了要研发的软件的目标与需求。并将软

件分为几个模块——操作接口模块、文本编辑模块、输入解析模块、核心算法模块、结果展现模块进行具体设计。对每一个模块，提出了具体方案和可行的方法。

第四章详细介绍了本软件的具体实现细节。先进行总体架构的讲解。接着，针对各个子模块进行有针对性的关键算法、组织结构或实现方法的讲述。

第五章总结全文所做的工作，并对以后的工作做出展望。

第2章 故障诊断方法与相应软件概述

随着数字集成电路工艺的不断提高，以及对芯片功能和性能要求的不断提高，芯片设计也越来越复杂。受到生产工艺以及制造过程的影响，在制造中难免会引入一些物理缺陷，导致芯片内部存在故障。为了能够防止由于制造过程中的缺陷而导致故障芯片流入市场，并且减少故障芯片在后续生产中出现的比例，需要对生产后的芯片进行高质量的测试和诊断。

数字集成电路中常见的逻辑故障模型包括固定型故障(stuck-at fault, SAF)[12]、桥接故障(bridging fault)、开路故障(open fault)、标准单元内故障(intra-cell fault)、跳变故障(transition fault)等[13]。

固定型故障包括固定 0(stuck-at 0)故障和固定 1(stuck-at 1)故障。如果某条线发生了固定型故障，那么该条线的逻辑值将固定为 0 或者 1。该故障模型描述的缺陷包括与地线或者电源线发生短路等。在早期的组合逻辑故障诊断方法研究中，常常将固定型故障作为诊断对象，但是它并不足以描述所有缺陷类型，因此还需要建立其它故障模型。

桥接故障描述了两根线发生桥接的缺陷，根据桥接程度的不同，这类缺陷所导致的故障逻辑值也不同，因此该故障模型包含诸多类型，如线与桥接故障(wire-and bridge fault)、线或桥接故障(wire-or bridge fault)、主导桥接故障(dominant bridge fault)、主导与桥接故障(dominant-and bridge fault)、主导或桥接故障 (dominant-or bridge fault)等等。开路故障描述了一根线发生的断路缺陷。当一根线发生断路后，该线的逻辑值将不受其扇入控制，而由其邻近线的逻辑值、与邻近线之间的耦合电容等等因素所决定。标准单元内故障描述了发生在标准单元内部晶体管或晶体管间的缺陷。例如，在一个与非门的晶体管上发生永久性短路或者永久性断路。

跳变故障描述了一种时延故障(delay fault)。如果一根线发生了跳变故障，指该线或输出为该线的门具有较大的延迟，以至于当这条线的逻辑值应该发生跳变时，其来不及跳变而反映出原有的逻辑值。

对于扫描链，比较常见的故障可以分为两类[14]，一类是建立时间(setup time)不满足导致的故障，故障行为表现为慢下降或者慢上升；另一类是保持时间(hold time)不满足带来的故障，故障行为的表现是快下降或者快上升。

本章首先介绍了数字电路的故障诊断的基本概念；其次介绍了对故障芯片的故障诊断方法；然后分析了现有故障诊断软件的发展现状；最后是对本章内容的小结。

2.1 数字电路故障诊断的基本概念

故障诊断的过程就是针对未通过测试的芯片，找到其中缺陷所在的位置并分析其原因，在 IC 设计和生产流程中，系统性地检测出芯片不合格原因的过程。故障诊断往往被与侦探过程和医疗过程相对比。在这两种情况下，工作人员都要根据观察到的症状来判断根本原因是什么。在故障诊断中，症状可以认为是在测试过程中哪个输出产生了错误的信号。故障诊断的主要目的是找出芯片失效的原因，这些原因对于提升成品率和客户的满意度有重要作用。

芯片是由各种元器件构成的，由于制造工艺、使用寿命、工作条件等的作用，故障的产生是难以避免的。对于电子系统，处理故障有两种基本的方案：第一种是利用冗余等可靠性技术，将故障暂时屏蔽起来，等到以后再进行修理。这种策略主要适用于需要长时间可靠工作，或无法停止系统工作的场合。另一种是及时处理，及时维修，也就是一旦系统出现故障，就停止系统运行，进行故障诊断和修复。无论哪种策略，其中的故障原因的分析都要涉及到电路的故障诊断。故障诊断是一项相当复杂的工作，主要原因是待诊断电路的规模通常很大，例如：待测电路的输入与输出端口可能多达数十个甚至上百个；电路的响应不仅是组合的而且经常包含时序的响应；集成电路中的各种基本门及记忆元件都集成在芯片内部，不可能直接测量它们的电平，而只能通过测试电路外部引线来间接测试。

数字系统故障诊断的基本办法是在输入端施加激励信号，在输出端获取响应，根据激励和响应的对应关系以及电路的拓扑关系确定故障位置。故障诊断在芯片生产中扮演重要作用。通常来说，一个典型的 IC 产品生产流程包括两个生产阶段：原型（prototype）生产阶段和量产（high-volume）阶段。

在原型生产阶段，少量的芯片被生产出来以便验证芯片功能的正确性。在这一阶段，芯片的成品率可能非常的低，这主要是源于设计错误或工艺偏差。这一阶段的诊断工作必须有助于分析出大部分降低成品率的原因。以下是其中一些常见原因：

（1） 时序错误（timing failure）和电压波动（circuit marginality）。生产出的实际芯片极有可能和时序仿真时的频率不一样，有可能因为没有在正常的电压或温度下工作。这种问题的出现概率很高，主要由于工艺偏差和仿真工具对真实情况的近似模拟所致。

（2） 对功能描述出现了误解。一个复杂的产品往往是由多个工程师合作进行生产的。因为功能说明是用英语写成的，所以往往存在二义性、前后矛盾、表达不完整等人工书写不可避免的错误。根据这种错误的功能说明所写成的 RTL 级代码可能

在某些情况下的行为不符合设计者的本意。因为功能测试生成和模拟非常耗时，工作人员不可能在流片（tape-out）之前对代码进行全面的验证。但是，通过严格的功能验证方法，这类问题是可以尽量减少的。这类问题属于设计错误。

（3） 不合理的版图设计。对于先进的纳米工艺来说，实际的设备和连线的尺寸会与版图所规定的尺寸有偏差。这是由于光刻工艺的光学效应所导致的。这些不匹配导致了可能的短路和开路，因而导致了电路的错误。受这一现象的启发，某些 DFM（design-for-manufacturability）规则可以被添加进设计规则中来保证可制造性。这种问题主要是工艺偏差的问题，但诊断过程会对导致错误的版图特征提供有效的启发信息，进而指导 DFM 规则的添加。

在原型生产阶段，同样的错误可能会在相当多的原型芯片上不断重现。工作人员可以较准确地通过降低时钟频率的方式确定错误类型是时序错误还是功能错误。有时，某些电压波动问题也可以通过更改电源电压来实现。在上述较易检测或修正的问题解决之后，诊断过程就会由设计师（designer），版图设计师（layout editor），测试工程师（testing engineer）所组成的团队开始进行，以便检测故障位置并指导成品率的大幅提升。

当一个设计通过了原型阶段，缺陷和电压波动问题等被大部分解决，生产过程就可以进入量产阶段。在量产阶段，产量可能会很低，也可能频繁波动。所以产量提升过程是很有必要的，这通常是由调整制造工艺来完成的。不过，即使芯片产量达到了最高的生产率，每个晶圆的成品率可能还有较大差别。所以持续的产量监测是很有必要的，以便应对特别的情况。在这一阶段，芯片的错误往往是由于工艺的不完美所致。已发表的研究中已经揭示出一些产生错误的机制，包括通孔错位（via misalignment），金属线的部分缺失（mouse-bites），晶体管源极和漏极的短路。这个阶段的诊断工作应该给出大量故障芯片的故障数据，以便后续统计分析，这样的分析能够揭示出限制成品率的关键机制。

诊断过程之后，失效分析工程师必须通过现有的手段尽最大可能检查硅片，最终发现导致芯片失效的机制。这些手段包括蚀刻掉某些金属层，用 SEM（scanning electronic microscopy）或 FIB（focused ion beam）来检查硅片表面。这些过程非常耗时。现代芯片往往含有几亿晶体管，如果没有高效的诊断工具来指导进行，失效分析工作将很难在合理的时间内判定导致电路失效的机制。可以看出，在芯片生产的不同阶段，诊断工作有不同的意义，但都对成品率的提升有重大作用。

故障诊断工作就是比对无故障电路和失效芯片响应的过程。无故障电路被叫做 CUD（circuit under diagnosis）。在某些测试向量的作用下，失效芯片和 CUD 的输出不完全相

同。类似于可测试性设计，我们可以插入可诊断性设计（design-for-diagnosability， DFD）电路来减少诊断的复杂性。举例来说，可使用扫描链旁路法[15]，增加一条扫描链来倒出另一条扫描链的信息。还可以在扫描链的单元前增加异或门，进行异步置位[16]。

大部分现有的诊断工具在逻辑部分进行诊断。在分析之后，报告故障可用两种方法，一是报告一系列候选故障，每个候选故障代表一个逻辑门或信号线。所有这些候选故障都被认为具有相同的成为真实故障的概率。一是报告一个候选故障的有序列表，排名最高的候选故障被认为最有可能成为真实故障。

故障诊断工具的质量可以通过多方面指标来考量。最重要的标准是它能不能精确地找出故障位置。具体包括以下指标：

（1） 第一次命中序号（first-hit index）：在提供一个有序结果列表，而不是一组成为真实故障概率相等的诊断结果的诊断工具上，无法定义诊断分辨率。在这种情况下，准确率可以被第一个真实故障的序号所描述。也就是说，候选故障列表中第一个被证明为真实故障的候选故障的列表中的序数叫做第一次命中序号。这个值越小，诊断结果越精确。

（2） 诊断分辨率（diagnostic resolution）：诊断工具报告的所有候选故障的数量与真实故障的数量之间的比叫做诊断分辨率。理想情况下，诊断分辨率应该是 1。在某种意义下，这个指标反映的是这个工具提供的结果够不够集中。然而，一个工具可能有很好的诊断分辨率，却没有报告出故障的真实位置。所以，分辨率够高并不意味着诊断效果够好。

（3） 前十命中率（top-10 hit）：芯片中有可能包含多个缺陷。因为在失效分析过程中，通常不会分析过多的逻辑门和信号线，候选故障列表中十位以后的故障往往就不会被分析了。在这十个故障中，最好是有多于一个的真实故障。因此，前十命中率被定义为故障列表中前十个故障中真实故障的比例。这是一个对多故障诊断很有意义的指标。这个指标的值越高，诊断效果越好。

2.2 故障诊断方法

芯片的故障可发生在任何位置，例如发生在时钟树、电压网络、组合电路或者扫描链等。已提出的故障诊断方法，一部分方法专门针对组合逻辑部分，一部分方法则专门针对扫描链。在本节中，我们就这两个部分，介绍故障诊断的基本方法。

2.2.1 组合电路故障诊断

在组合电路故障诊断中，我们只关注组合逻辑中的故障，也就是进行逻辑运算的部分的故障。在这种诊断情况下，我们假定触发器和扫描链都是无故障的。在这种诊断过程中，主要有两类方法：因果分析法[17][18]和果因分析法[19-24]。

2.2.1.1 因果分析法（cause-effect analysis）

因果分析法的第一步是制定故障模型（通常是固定型故障）。之后进行大量的故障模拟，为每一个测试向量和故障位置的组合创建故障字典。一旦这个故障字典被创建，就可以用查表的方法利用失效芯片的失效响应在故障字典中进行查找。在故障字典中查找到的失效输出最接近实际情况的故障往往可能是真实故障。这种方法也常常被叫做基于故障字典的方法。

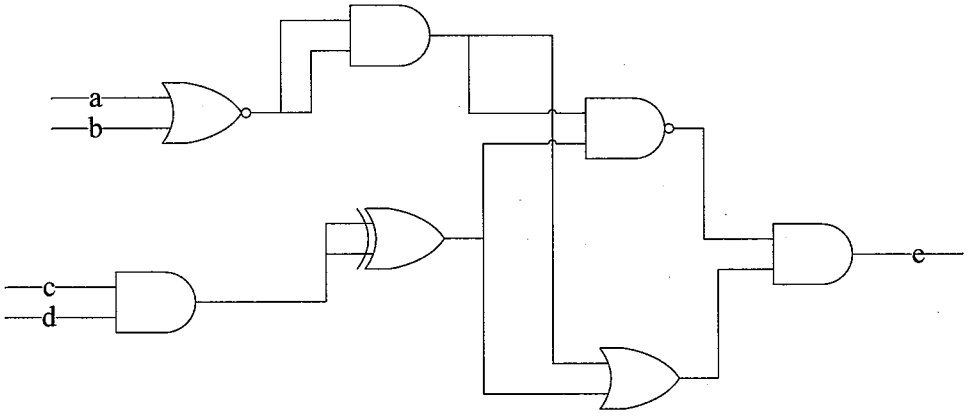


图 2.1 待诊断电路

考虑如图 2.1 的电路。这个电路有四个输入{a，b，c，d}，一个输出{e}。假设有事先生成的五个向量{v_1，v_2，v_3，v_4，v_5}。因为是基于单故障假设，我们假定故障空间是{f_1，f_2，f_3，f_4，f_5}。

我们列出各种故障在向量序列 v_1，v_2，v_3，v_4，v_5 下的响应序列：

（1） 无故障：00001

（2） f_1： 11111

（3） f_2： 01101

（4） f_3： 00011

（5） f_4： 10100

（6） f_5： 11101

以上说明了在所有故障和向量组合情况下输出信号 e 的响应，包括了无故障状态和五个故障状态下的响应。每一行对应着一种状态下的电路，每一列对应着一种测试向量的响应。接下来，我们可以构造一种简单的故障列表。

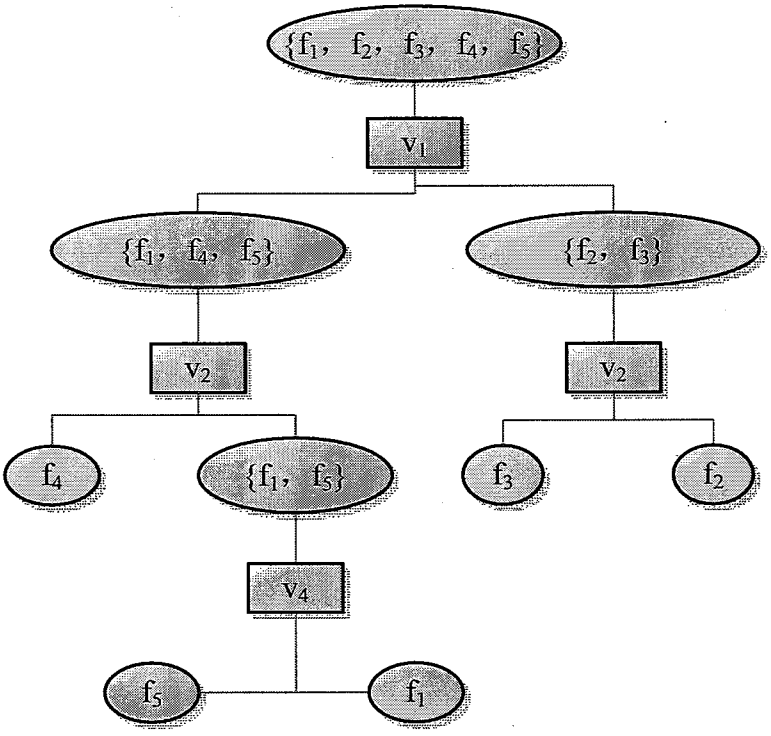


图 2.2 诊断树

图 2.3 展示了一个可能的故障字典，这种故障字典通常被称为诊断树。这个诊断树不一定是最小尺寸的。我们只是用它来展示诊断过程。基本的思想就是不断地缩小可能的故障集合。最初，这个集合包括了所有可能的故障。检查了在第一个向量下的响应后，我们就可把故障集合缩小为一个只有几个故障的集合。在这个例子中是{f_1，f_4，f_5}和{f_2，f_3}。因为 CUD 只有一个输出，我们就只用划分两个集合。不过，一般情况下，划分速度会更快。在这个例子中，细化过程一直持续到检查 v_1，v_2，v_4 的响应之后。诊断过程就是一个从树根到叶子的过程。

构建故障字典可能耗费很大的时间和空间，不过，一旦故障字典建立好了，进行诊断就相当容易了。因为故障字典只用建立一次，其后的诊断工作都是基于其上进行。在实际应用中，这个方法可能会受到一些因素的限制：

（1） 故障字典的大小。故障字典记录了所有向量和故障的组合下的输出响应。如果没有通过很好地压缩，大小是跟三个因素有关的：（F·V·O），F是故障数量，V是向量数量，O是输出端口数。在一个有一百万个逻辑门，一万个触发器，一万个测试向量的芯片中，这个大小会达到 10^{12} 比特位，需要非常大的存储空间。同时，如果电路有较小的修改，故障字典就得重新生成。

（2） 未经模型化的故障。故障字典用的是固定型单故障假设。如果 CUD 确实含有一个固定型故障，那么结果就会非常准确。然而，现实中的缺陷不一定是固定型故障，有可能是桥接故障等其他的故障。这些问题也加重了存储空间的负担，也可使诊断工作产生错误。所以，因果分析法的诊断效果并不非常理想。

2.2.1.2 果因分析法（cause-effect analysis）

与因果分析法不同的是，果因分析法直接检查芯片的失效响应，并由此推断故障位置。因果分析法相对于因果分析法有很多优点：

（1） 不用事先假定故障模型，所以能检测更多故障。

（2） 可以适用于芯片的多故障检测，特别是当这些故障结构相关的时候。

（3） 能够用于部分扫描链结构中。

果因分析法的一个缺点是运行时间长，因为对每个故障芯片都要进行一次全面的检测过程。相比于故障字典方法，这是个动态的方法。由于故障诊断是用来指导物理失效分析的，所以诊断时间往往不是一个大问题。

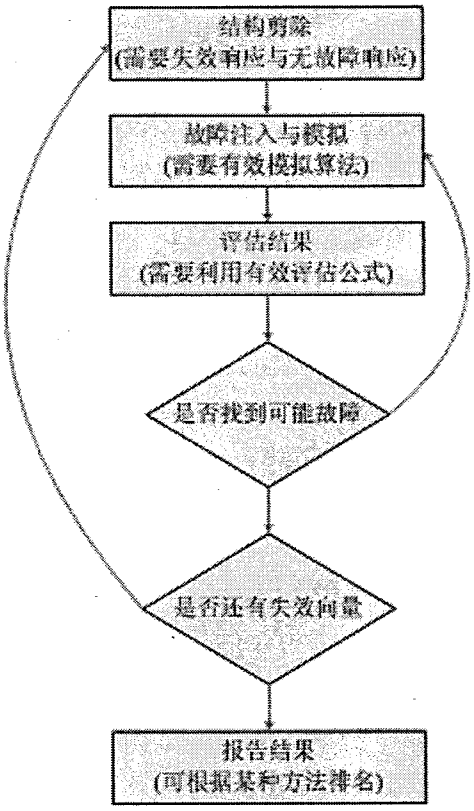


图 2.3 果因分析法流程

上图 2.3 是此方法的流程。首先利用 CUD 进行无故障模拟，得到无故障的响应。之后开始诊断流程。第一步是剪除过程，对比无故障输出和故障输出，对电路中不可能出现故障的部分进行剪除。之后是模拟过程，在剩余电路结构中一个一个注入故障进行模拟，得到结果后进行评估。之后再用另一条测试向量重复上述过程。再之后就是评分并给出结果，主要利用一些指标对可能的故障进行排名。简单而言，果因分析法包括三步：结构剪除，故障模拟，评估故障。

结构剪除

CUD 中一个输出信号的扇入逻辑锥指的是所有可到达此输出信号的逻辑门和互连线。利用电路中的故障假设，我们可以利用逻辑锥的并集和交集操作进行结构剪除，删掉不可能产生错误的区域[25]。

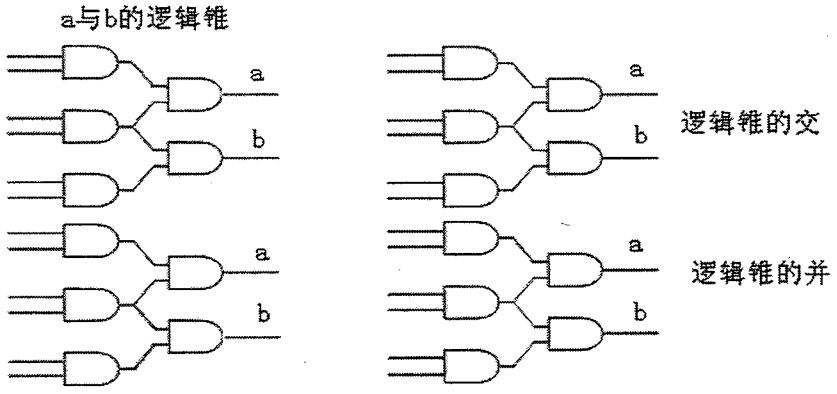


图 2.4 逻辑锥的交与并

如图 2.4 所示，如果只有一个故障，最好采用逻辑锥的交操作，用每个故障输出的逻辑锥进行求交。所得出的区域是唯一一块可能存在故障的区域，叫做故障候选区域（fault candidate area）。不过如果故障芯片中不止有一个故障，我们就应该采取求并集的方法。因为有可能扇入逻辑锥中的逻辑门并没有影响其他的故障输出。两者对比的话，显然求交运算比求并运算有效得多。不过，由于在诊断之前我们并不知道故障芯片中有多少故障，所以对逻辑锥进行求并运算是一个相对保守和有效的方法。如果在有多故障的芯片上用求交的方法选取故障区域，往往会得到空集。

之后，我们利用回溯算法进行处理。这种算法起初用在故障模拟中，后来被用到故障诊断之中。这种算法的具体过程是，从错误的输出开始，一步步根据逻辑门和输入信号的特点进行推断并找出可能的故障集合。这里面用到了控制值和非控制值的概念。控制值是指不论其他输入信号是何值，此输入信号可以导致逻辑门的输出必然是一种确定的值。反之，则叫非控制值。我们用与非门举例。如果一个与非门的输入是 0，那它的输出就是 1，这个输入就叫控制值。如果一个输入是 1，那它的输出还要取决于其他的输入，那这个输入就叫非控制值。那么回溯算法的过程就是从失效输出开始，先向上包括这个输出所连逻辑门的控制值，如果没有控制值就把所有输入都包括。然后再递归到新的输入上重新开始。整个过程如图 2.5 所示：

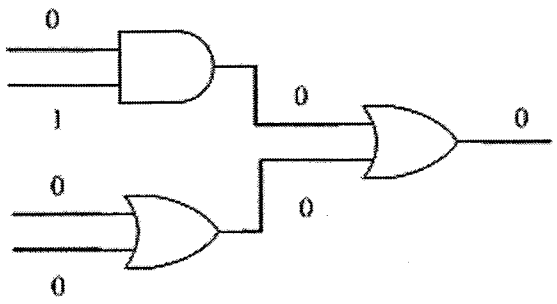


图 2.5 回溯过程

图 2.5 的过程为从输出 0 开始，因为或门有两个非控制值，就把两个输入都包括。对于回溯到的与门，输入中有一个控制值 0，就只包括这个 0 值。对于回溯到的或门，要把两个非控制值都包括。

故障模拟

结构剪除可以有效地降低需要考虑的故障位置，但还不够精确。为了使得结果更精确，我们需要在这些候选位置上进行故障模拟。通常使用的方法是注入和评估过程（inject-and-evaluate paradigm）。简单地讲，这种过程就是先假定一个故障位置，再在这个故障的基础上进行电路上输出信号的模拟。图 2.6 是注入和评估过程的展示：

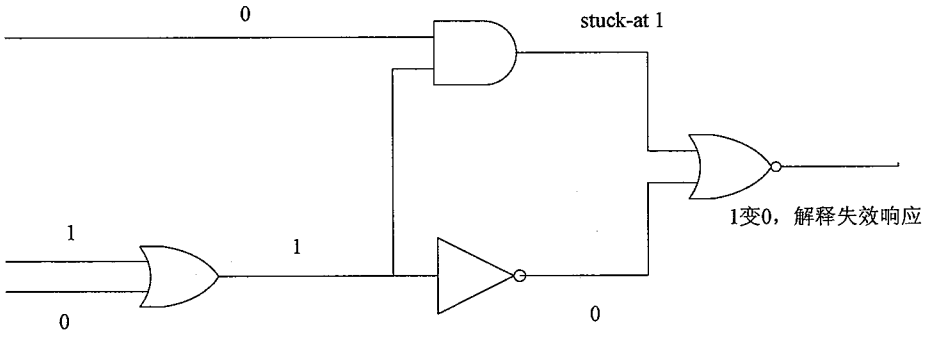


图 2.6 故障模拟

结构剪除过程是注入评估过程的预处理过程。之后，开始运行两层的循环过程。最外层循环对每一个失效向量进行循环，最内层循环对每一个可能的故障位置进行循环。具体地说，过程如下：

1. 先对每一个失效向量进行无故障模拟，然后记录下这些无故障状态下的输出值。以备下一步使用。

2. 对每一个可能的故障位置 f，我们用三个小步骤进行处理：

2.1 在 f 位置反转无故障模拟下的值，创建一个信号转变事件。这个就是故障注入过程。

2.2 进行事件驱动的模拟，得出有故障时电路的响应。这个就是故障的模拟过程。

2.3 对每次模拟所得出的结果进行打分。评估这次模拟的结果和实际的失效输出有什么不同，这个是打分过程。

图 2.7 是模拟过程的流程图：

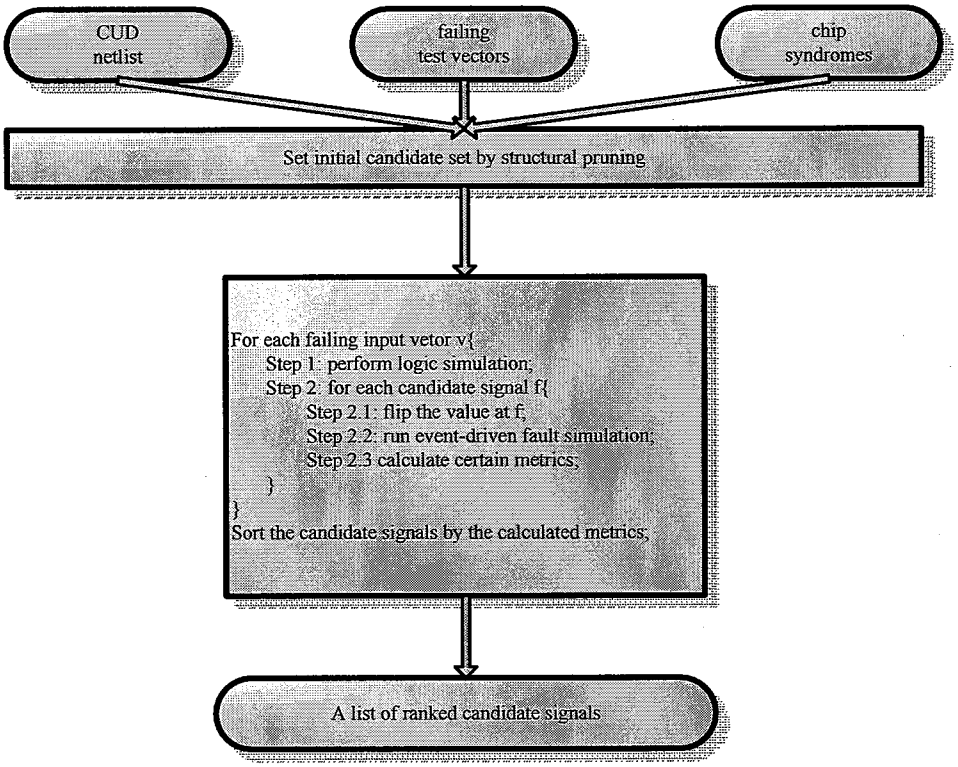


图 2.7 注入评估过程框图

评估策略

在进行了以上两步后，就进入到果因分析法的最后一步。对这些候选故障进行评估后，就可以报告结果了。

当在 CUD 上进行故障模拟时，有的故障可以完全解释故障输出，也就是说故障输出和模拟所得结果一模一样。这种情况非常难得，也就是说有极大的概率这次注入的故障就是实际存在的故障。不过，更有可能的情况是，有一部分失效输出被解释了，一部分失效输出和模拟结果不一致。那么那些与失效输出相匹配的输出叫做可治愈的输出。在这种情况下，我们就需要为这个模拟得到的故障进行打分。有一种打分方法如下式所示：

分值 = 可治愈输出的数量 – 不匹配的输出的数量 × 0.5 (2.1)

通过这个公式我们看出结果和治愈输出以及不匹配输出都有关系。这种方法在处理单故障时特别有效，但在处理多故障时，需要检测的情况就非常多。为了处理多故障的情况，有人提出了 SLAT 的方法[26-30]。

SLAT 是指一种测试向量的属性。具有这个属性的向量有至少一个故障位置可以单独解释所有的失效输出，同时又不影响其他正确的输出。我们用以下例子来进行说明。假设有六个 SLAT 性质的向量，同时有六个可能的故障位置。这六个向量我们用 V1，V2，V3，V4，V5，V6 来表示，故障用 F1，F2，F3，F4，F5，F6 来表示，我们把能够解释这六个 SLAT 向量的故障列举如下：

（1） V1：F1，F3

（2） V2：F2

（3） V3：F2，F3，F4

（4） V4：F1，F3

（5） V5：F6

（6） V6：F5，F6

当然，在某些情况下，能覆盖所有失效向量的故障集合并不唯一。比如，上例中 F1，F2，F6 或 F2，F3，F6 都有可能是故障的最小覆盖。那么在这种情况下，我们可以这样来产生结果。要么，我们把这两个故障集都作为最终结果，要么我们把两个故障集结合起来，把 F1，F2，F3，F6 作为最终结果。以上两种方法都不是较令人满意的方法。所以，有其他的研究人员提出了一些另外的方法。有人利用额外的 SLAT 向量的结果来指导当前的结果[31]。用这种方法的原因是多次检测到的故障往往更有可能是实际缺陷。还有一些研究人员利用版图信息来指导结果的判断[32]。因为有些金属线互连的缺陷无法在网表中反映出来。比如，两根金属线在版图上是相邻近的线时，这两根金属线的桥接故障的概率就大于其他的信号线对。所以，这两根线的桥接故障就会被赋予更高的权重。我们用图 2.8 进行说明。网表中的 s1，s2 桥接故障是可能的故障。在版图上，这两根线距离很近，那么这个故障就被优先考虑。

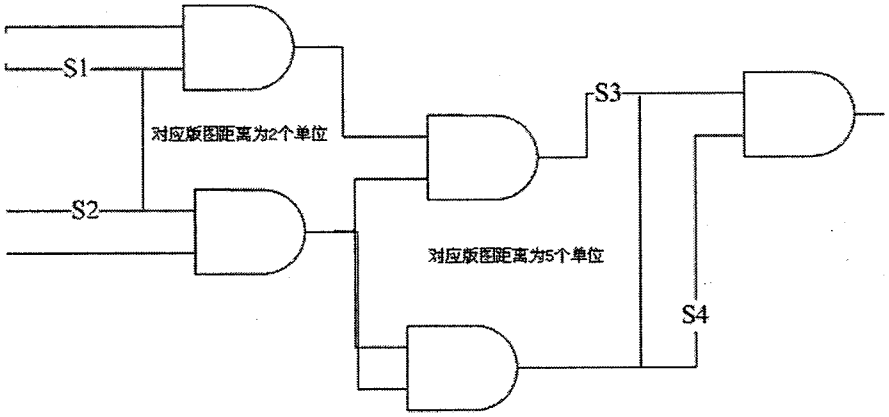


图 2.8 桥接故障的网表表示和版图信息

2.2.2 扫描链故障诊断

对于扫描链的故障诊断，一种方法是利用故障模拟来进行诊断[33-36]，一种方法是生成确定型诊断向量来进行诊断[37-40]。我们先说明扫描链故障模型，再介绍相应诊断方法。

2.2.2.1 故障模型

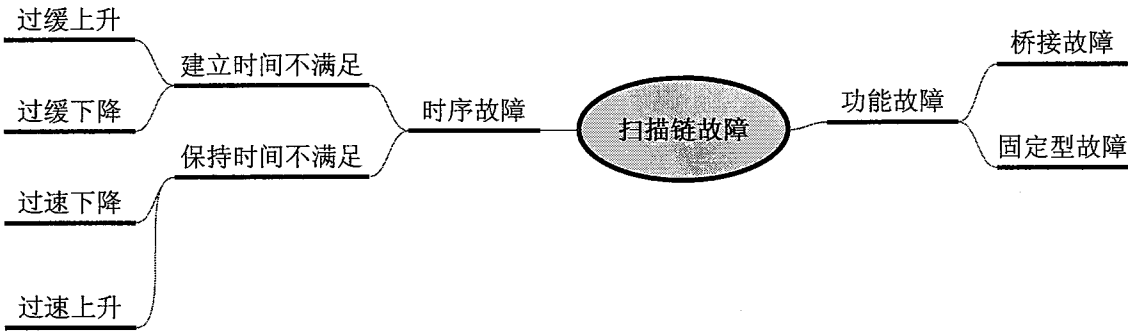
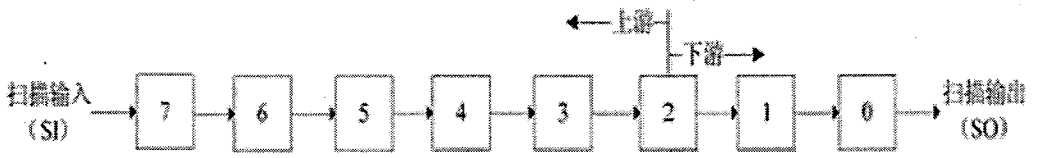


图 2.9 故障模型

扫描链故障主要是两种。时序故障和功能故障。如图 2.9 所示，功能故障主要包括固定型故障和桥接故障。固定型故障是说信号线上的值恒为 1 或 0。这种故障可能是电源、地线与金属线的短接。桥接故障是指金属线的短路。时序故障是由于电路行为没有满足一些时序要求所导致的故障。这种故障主要有两种情况，一是由建立时间不满足而导致过缓下降（slow to fall），过缓上升（slow to rise），另一类是由保持时间不满足导致的过速下降（fast to fall）和过速上升（fast to rise）。

通常，我们把扫描链上的扫描单元按照从输出到输入的顺序进行从小到大的顺序

进行编号。这样，这些扫描单元就被赋予了一个唯一编号。这些单元的个数就被称为扫描链的长度。对于具体的扫描单元来说，索引值大于这个扫描单元的单元就被称为这个扫描单元的上游。同理，小于这个单元的单元就被称为下游。图 2.10 展示了在不同的故障下，扫描向量的输出和无故障输出有什么不同。



(a) 扫描链模型

向量	向量 1	向量 2
无故障扫描输出与输入	11110000	00001111
存在 SA0 的扫描输出	**0000**0000	0000**0000**
存在 SA1 的扫描输出	1111**1111**	**1111**1111
存在 STR 的扫描输出	111**0**0000	00001111
存在 STF 的扫描输出	11110000	0000**0**111
存在 FTR 的扫描输出	1111**1**000	00001111
存在 FTF 的扫描输出	11110000	0000**0**111

(b) 扫描链故障效果

图 2.10 扫描链模型与故障输出[41]

2.2.2.2 基于故障模拟的扫描链故障诊断

扫描链故障诊断也可以利用故障模拟的诊断方式[42]。这种方法主要是先进行故障注入，再进行故障模拟。这种方法把移入的扫描链向量当成输入，把移出的向量当作输出。如果输出和预期不符，就认为是失效输出，然后利用失效向量和失效输出进行类似组合电路的注入与模拟过程。我们用图 2.11 进行举例。扫描链有四个单元，我们移入向量 1011，经过捕获后，假设正确的预期输出为 0111，而实际输出为 0010。于是我们开始诊断过程。我们在第二个单元之后注入固定为 0 的故障。第一步，移入 1011 向量。由于故障的存在，接近扫描链输出端的两个单元被污染为 0。第二步，在时钟的作用下进行捕获过程，结果为 0110。第三步，移出向量，故障把第二个单元的值污染为 0。最终输出为 0010，与实际输出相符，我们认为这次注入的故障是实际故障。

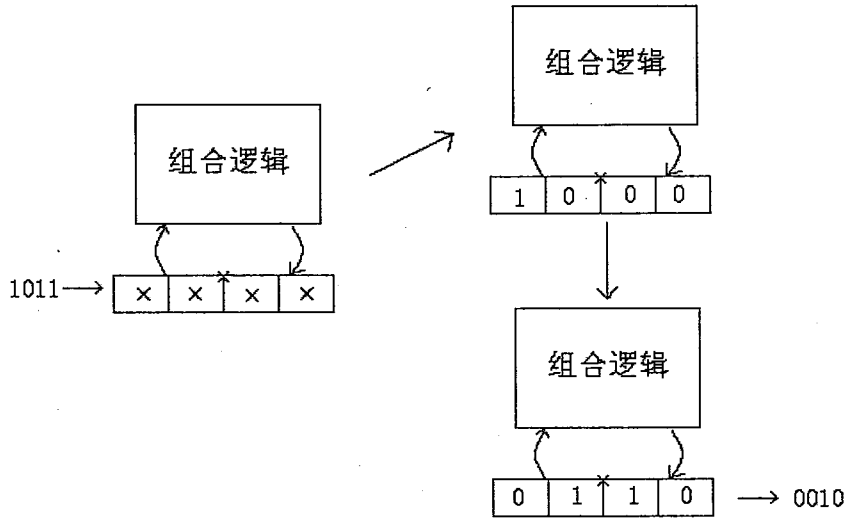


图 2.11 基于故障模拟的扫描链故障诊断方法

2.2.2.3 基于确定性诊断向量的扫描链故障诊断方法

因为故障模拟的方法的精确度依赖于测试电路和测试向量的情况，所以为了得到更好的效果，有研究人员提出了确定性诊断向量的方法（determination diagnosis pattern generation，DDPG）。

确定性向量的生成方法主要是两种。一种方法是让候选扫描单元在激励下得到期望的逻辑值[43]。这种情况下，此单元在设备上的响应和期望响应相同，就认为是无故障扫描单元。如果不同，就认为是故障单元的上游。另一种方法是，将候选单元的逻辑值传导到可靠观察点[44]。这种观察点包括电路主输出，无故障扫描链伪输出，故障扫描链中故障单元的下游组成的伪输出。通过观察响应来推断候选单元实际值。

2.3 本章小结

本章首先对故障诊断的概况进行了简要的介绍。然后分别介绍了故障诊断中组合逻辑的诊断方法，并具体介绍了每种方法的具体步骤和特点。接着，对电路中的扫描链的故障模型进行了介绍，之后总结了对扫描链进行故障诊断的方法，提供了具体思路和原理。

故障诊断是失效分析的前提，对芯片的量产有重要意义。近年来的诊断方法主要集中在组合逻辑和扫描链上。在这些研究方向上，许多学者提出了多种方法，并做出了重要贡献。

第3章 Yield Explorer 故障诊断软件设计与实现要求

本章在进一步分析故障诊断软件的需求和发展现状上，给出 Yield Explorer 故障诊断软件的总体设计框架。我们将软件分为操作接口模块、文本编辑模块、输入解析模块、核心算法模块、结果展现模块。对各个子模块的设计提出有特色的要求，对模块间的整合与协作给出具体的方案。切实实现有效的故障诊断软件设计，以期提高故障诊断人员的相关工作效率。

本章的组织结构为：3.1 节讲述故障诊断软件的发展现状以及 Yield Explorer 的发展目标；3.2 节讲述故障诊断软件的总体设计方案；3.3 节介绍各个子模块的具体设计；最后对本章进行总结。

3.1 诊断软件发展现状与 Yield Explorer 设计目标

要对故障诊断软件进行总体设计，就必须了解市场上现有的故障诊断工具。了解这些软件的具体特点，在借鉴这些软件的特点的基础上加入自己的特色，形成自己的风格和优势。经过调研，现有的故障诊断软件主要集中在国外的大公司中，国内的厂商和研究机构少有类似的故障诊断软件。

具体而言，主流的故障诊断软件的具体情况如下：

1. Mentor Graphics 公司

此公司有故障诊断工具 Tessent™ Diagnosis[45]。此软件能够提供高解析度的判别故障形成机制、物理位置、逻辑位置的诊断。有单元内部故障的诊断能力（cell-internal diagnosis capability），扫描链故障的诊断能力（advanced scan chain diagnosis capability）。同时利用版图信息（layout-aware）的性质使它能通过分析网络拓扑来提高诊断解析度。诊断结果可通过大多数版图查看器查看。版图级别的诊断需要 LEF/DEF 格式的物理信息。为了加速失效分析的进程，Tessent Diagnosis 还提供了针对诊断结果的 ATPG。此工具通过初步结果生成更多的向量来提高诊断解析度。Tessent™ Diagnosis 还可与该公司测试工具相配套。同时还有服务器模式利用网络资源处理大量数据。

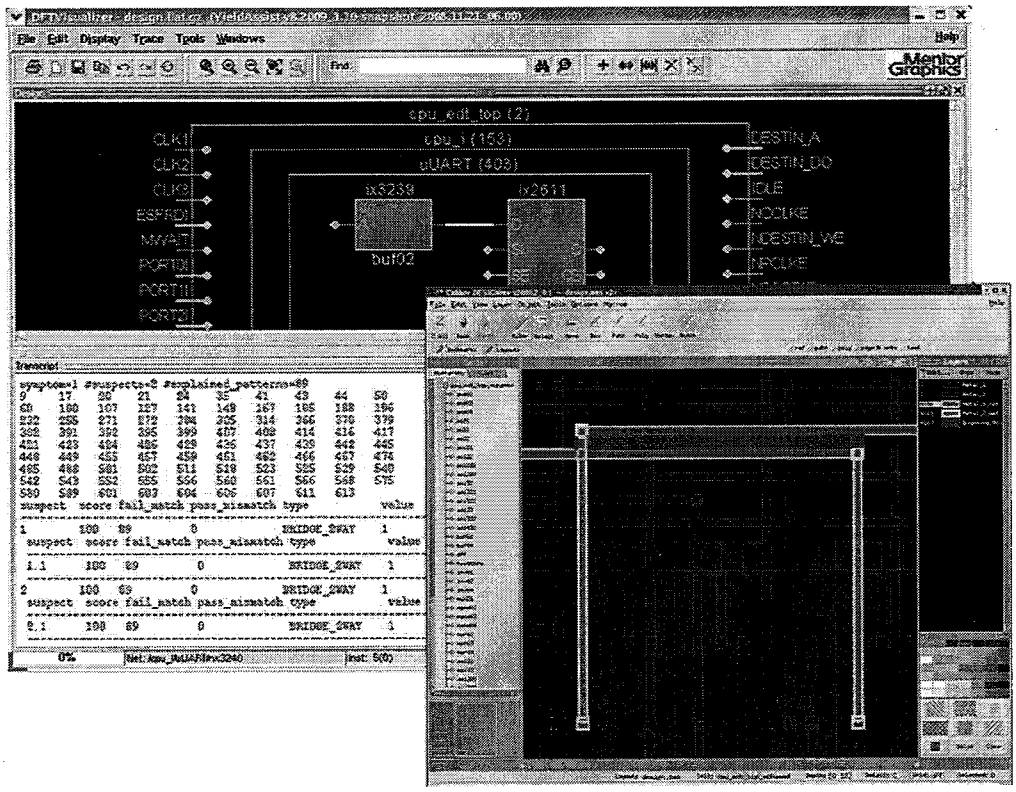


图 3.1 Tessent™ Diagnosis 使用效果

此软件具有如下的具体优点：

- 通过有效的识别导致测试错误的缺陷来加速失效分析
- 支持以诊断工作为主导的成品率分析流程
- 可检测时序错误
- 利用版图的能力可以提高诊断质量
- 自动日志记录和分布式的处理
- 多种扫描链故障的诊断能力
- 可直接利用 Tessent TestKompress® 软件的结果进行诊断
- 故障按可能性大小报告为列表

2. Synopsys 公司

此公司有 TetraMAX ATPG[46]。此工具是 ATPG 工具，但它有故障诊断功能，可以在版图上指出故障位置。为了提高诊断效率，它采用了启发式算法和高效的故障模拟算法。

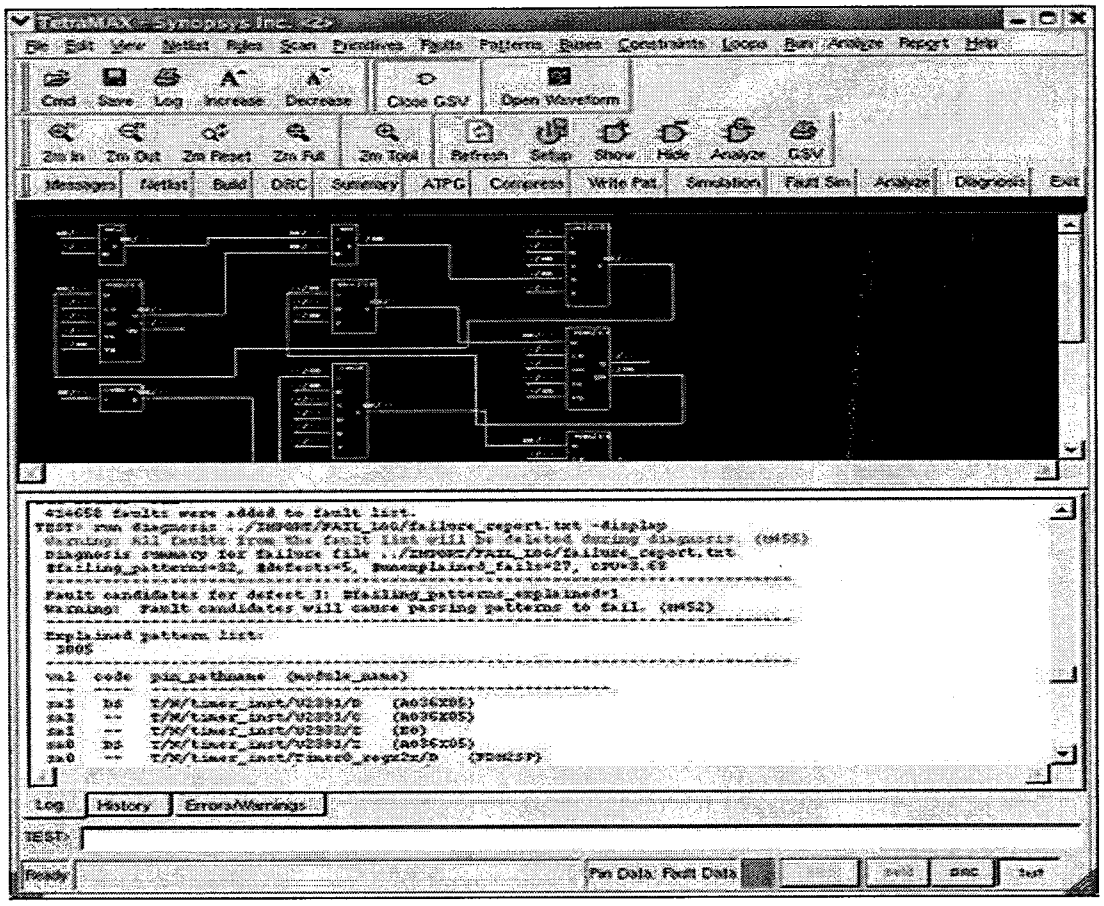


图 3.2 TetraMAX ATPG 使用效果

3. Cadence 公司

此公司有故障诊断工具 Encounter Diagnostics[47]。Encounter Diagnostics 分析大规模测试的结果来寻找稳定出现的错误并能够在版图上指出故障位置。Encounter Diagnostics 支持大规模数据检测模式和小规模精确诊断模式，静态与动态的故障类型，故障建模，逻辑模型和物理模型的显示，各种常用的测试向量格式。Encounter Diagnostics 的高级特性有扫描链故障诊断，时序错误诊断，逻辑锥划分，针对诊断的 ATPG。

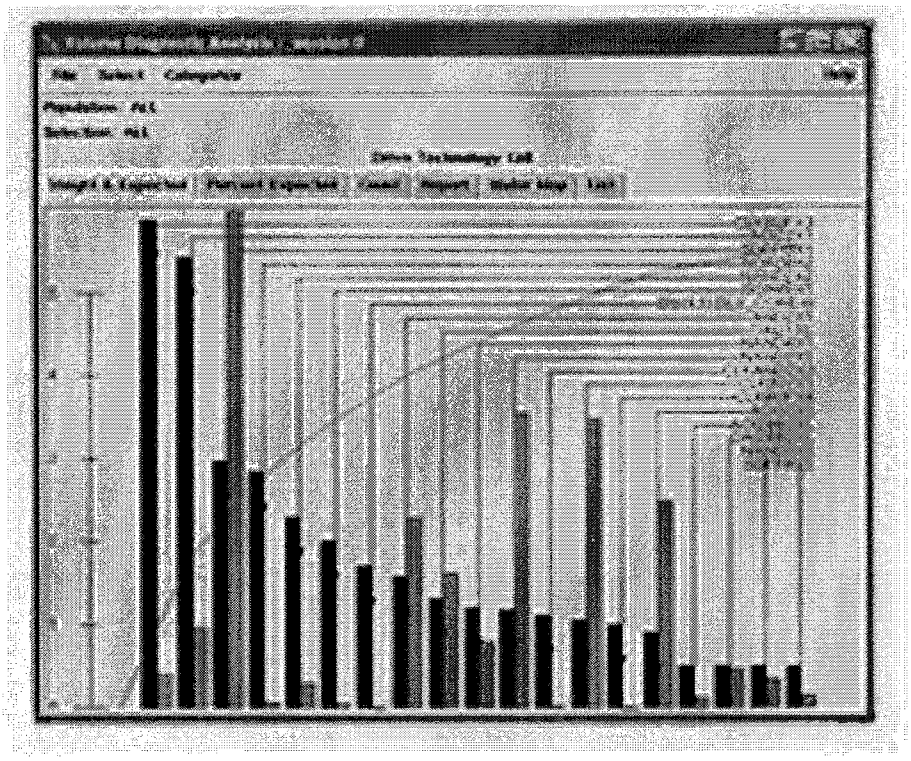


图 3.3 Encounter Diagnostics 使用效果

此软件具有以下特点：

- 可检测出限制量产的最关键因素
- 对最关键故障的检测准确率为 80%
- 可利用逻辑模型和物理模型来检测故障
- 可检测扫描链故障
- 专有的故障模型对 65nm 和 45nm 的故障检测很有效
- 支持多种 ATPG 向量格式
- 可利用多个处理器进行分布式计算

经过以上调研可以看出现在比较成熟的故障诊断工具有 Mentor Graphics 公司的 Tessent™ Diagnosis 和 Cadence 公司的 Encounter Diagnostics。从接受的诊断向量来说，Encounter Diagnostics 做的比较好，它可接受所有的主流测试向量格式。对于这一点，我们决定本软件也应接受主流的输入文件格式。对于主流软件共有的功能，我们的工具应在第一版以至今后的工作中尽量做到包含。比如，能够利用版图信息进行诊断、诊断能力多（比如组合电路的诊断、扫描链的诊断等）等。不过，上述软件有几点不足有待加强。比如，软件的使用流程复杂化或不清晰，软件缺乏对多故障屏蔽与加强作用的考

虑，诊断相关的辅助功能少，软件体积大等等。经过学习和思考，我们认为本软件应在上述其他软件的不足处进行加强，另外可扩展性要强，以便未来有功能还可以扩充。同时，因为本软件还要作为今后研究的工具，那么本软件应该可以保留历次的诊断结果并可以进行图形化的统计显示。还有，我们发现成熟的工业软件缺少源文件的编辑功能，为了保证修改和调试的有效性，我们对故障诊断软件增加了编辑文件功能。本软件还集成了电路信息查询功能、故障模拟功能和生成故障功能等。针对这些功能，本软件支持有效的、顺序化的工作流程。以上是我们决定的软件的大体特点。

3.2 Yield Explorer 软件总体设计方案

在我们的研究中，有几个关键问题。如何设计软件的操作接口以及提供哪些功能是关键问题之一，如何将所提故障诊断算法集成到软件之中是关键问题之二。由于我们希望所开发的故障诊断软件界面友好易用并具有完整故障诊断流程，因此在总结归纳国外相关软件特点的基础上，形成了 Yield Explorer 故障诊断软件的总体设计。

1．Yield Explorer 软件功能设置

该软件提供了完整流程的故障诊断功能。首先，本软件有对电路信息的查询功能。比如单连线分析和双连线分析功能。用户可以利用这些功能对单连线两端所连接的逻辑门或两条信号线是否有交汇点等信息进行查询。第二，本软件提供故障生成的功能。对于故障生成，我们提供三种生成模式。用户可单选任一种模式，输入数据后进行生成。同时还有一些附加选项可微调生成方法，比如参考版图信息选项，生成可诊断故障选项等。第三，软件具有故障模拟的功能。所需输入数据是故障生成功能中生成的故障。输出是电路的失效响应。同时也具备一些选项可以对模拟结果进行微调。第四，故障诊断功能。有两种故障诊断算法。一种是利用本小组已经研究出的诊断算法进行故障诊断。一种是利用经典的故障诊断算法进行诊断。第四，诊断结果评估。对于诊断结果评估，我们输入诊断结果文件和真实的故障信息文件进行比对。结果通过柱状图进行显示。对于历次的结果评估，软件会进行历史记录，可以通过相应菜单项进行统计图查看。

值得一提的是本软件集成了较全面的文本编辑功能。例如多标签浏览与编辑、查找替换、关闭提示等。集成这些特性的目的主要是方便测试诊断人员对相关源文件进行实时的查看编辑，加快工作速度。这些文本编辑功能与工程管理特性相结合，非常便于操作。在导入工程之后，在工程列表框中双击相应文件就可打开文件。

对于输入文件，本软件接受主流的文件格式，例如 Verilog、STIL、LEF/DEF 等。同时还接受本小组自己制定的文件格式。

输出方面，我们力求直接易懂，同时还要作到历史数据可查询。所以大部分的输出结果会在软件界面直接显示出来，同时保存在相应文件夹里以备今后查询。对于统计数据也作到了图形化显示。

2．Yield Explorer 软件框架

对于故障诊断软件来说，用户需要读入网表文件，向量文件，版图文件等，之后进一步进行诊断流程的操作。本文中的 Yield Explorer 软件除了这些基本功能之外，对软件结构还实现了模块化，增强了可扩展性，便于以后的功能扩充。

本软件可以与其他 EDA 工具进行配合使用。能够与其他 IC 测试与诊断相关工具联合使用，也是我们的设计初衷之一。作到这一目标能够使整个诊断工作规范化，系统化，增强整个团队的工作效率。为了达到这个目的，输入输出文件的格式就必须标准化，需要使用其他 EDA 软件广泛接受的文件格式。例如，对于网表文件，本软件就可以接受 Verilog 文件。当然，本软件还不完善，未来还会在现有的基础上进行扩充。所以，在实现本软件的过程中，模块化和可扩展性就得到了较高的关注。

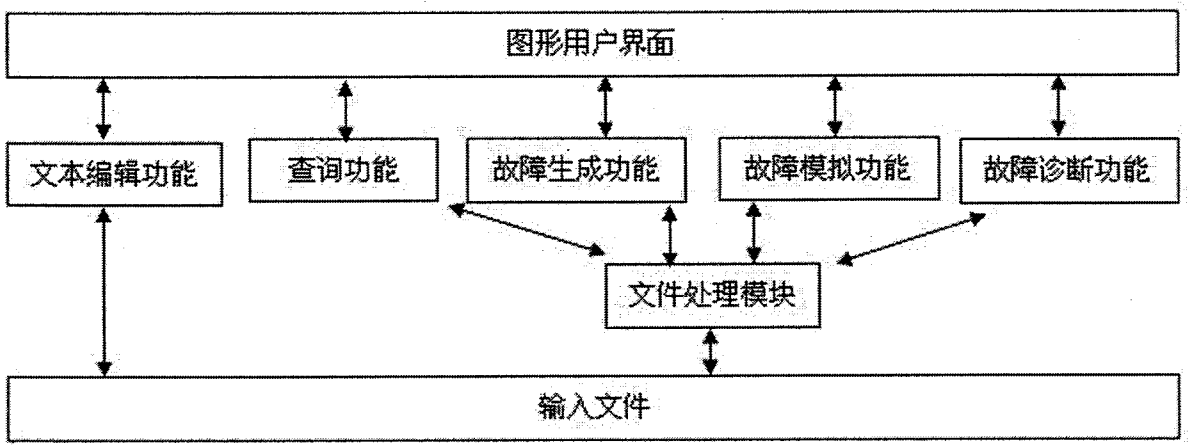


图 3.4 Yield Explorer 模块框图

图 3.4 描绘了 Yield Explorer 的结构框图。在此结构框图中，各个模块的相互关系也得到了充分的展现。我们可以看到，软件的模块化的目标得到了较好的实现。每个模块之间都是相对独立的，这样便于以后增删功能模块，可维护性较强。

软件通过图形用户界面进行操作和显示。通过调用相应的功能菜单项来执行需要的功能。之后，软件调用相应的模块进行处理。通常，需要先经过文件的解析，再用解析得到的内部数据进行处理。得到的结果会通过图形界面进行实时的显示。

如何进行输入文件的内部处理是本软件所要解决的重大问题之一。对于网表文件，软件接受的标准输入格式是 Verilog。对于版图文件，软件接受的标准输入格式是

LEF/DEF。这两种输入格式在 Open Access 函数库中都有现成的处理程序，所以本软件对这两种格式使用 Open Access 函数库中的程序进行转化。利用 Open Access 函数库自带的 verilog2oa、lef2oa 和 def2oa 三个程序把输入文件转化成 Open Access 自有的格式。软件的其他功能读入这些自有格式的数据并转化为软件内部的数据结构进行储存和处理。不过，值得一提的是，verilog2oa 只能转化不含有赋值等功能性语句的 Verilog 文件，只能通过 Verilog 文件中的模块的嵌套结构来建构电路结构。不过这对于我们现阶段的软件来说已经够用了。对于诊断向量文件，软件接受的标准格式是 STIL 格式。对于这种文件格式，本软件利用相关开源软件的 STIL 解析模块进行处理。转化后的数据会在内部的数据结构中进行保存。利用内部的数据结构保存输入数据也是为了保证功能模块之间的相互独立。这样，功能模块就不需要直接访问磁盘文件，未来更换输入文件格式或更新文件解析功能时就会更加方便。

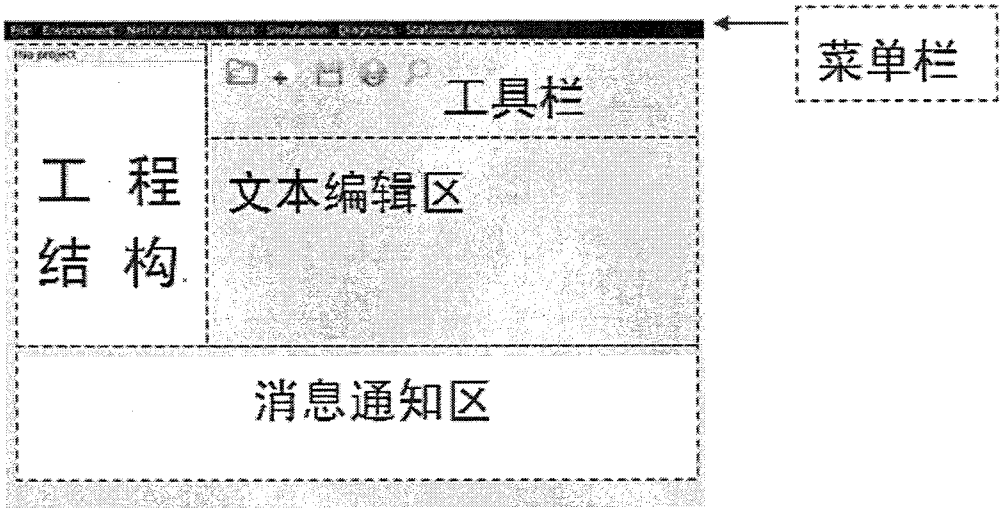


图 3.5 Yield Explorer 界面组成

图 3.5 展示的是 Yield Explorer 的初始界面。

用户界面由 Qt 开发工具开发完成。使用 Qt 开发工具的优点是界面控件图形艳丽、易于安排控件的位置、编程方法简单。所开发软件的菜单项以及工具栏图标在不该被使用时会显示为灰色，可以使用时会被激活。这些细节的要求也可用 Qt 较简单的实现。界面的结构主要通过容器构件来进行组织。对菜单项和按钮，连接上回调函数，这样点击这些元素时，就会调用相应函数执行功能。这些功能包括改变界面某些元素的显示，执行诊断相关功能和在界面上输出信息等。

用户界面大致分为五个区域。每个区域负责一种相对独立的任务。菜单栏主要用来负责功能的激发。File 菜单中主要包括对工程结构和源文件的相关操作。工程与文件的

创建与打开等功能都可在这个菜单项中找到。Environment 菜单主要涵盖对输入文件的读入功能。例如网表文件、向量文件、版图文件等需要用到此菜单中的菜单项进行读取，之后才可以在读取的文件的基础上进行后续的操作。Fault 菜单主要用来进行故障生成。Simulation 菜单包括故障模拟的相关功能。Diagnosis 菜单中包括的是诊断的相关功能。Statistical Analysis 菜单中涵盖对故障诊断结果的统计分析功能。界面的左边是工程结构区域。这个区域用来展现所打开的工程文件夹中目录与文件的组织结构。最上方显示工程目录的名字，没打开工程的时候显示 No project. 工具栏处于界面的右上方，主要是对文件操作的快捷按钮。工具栏下面是文本编辑区。通过菜单或工程列表打开的文件都会显示在这个区域。未打开文件时，整个区域为灰色。最下方是消息通知区域。此区域主要用来显示运行结果。

五个功能区域负责不同的任务，既相对独立又互相协作。通过菜单与工程结构目录调度任务，再通过文本区域修改文件，最后通过消息通知区域查看结果是主要的工作模式。

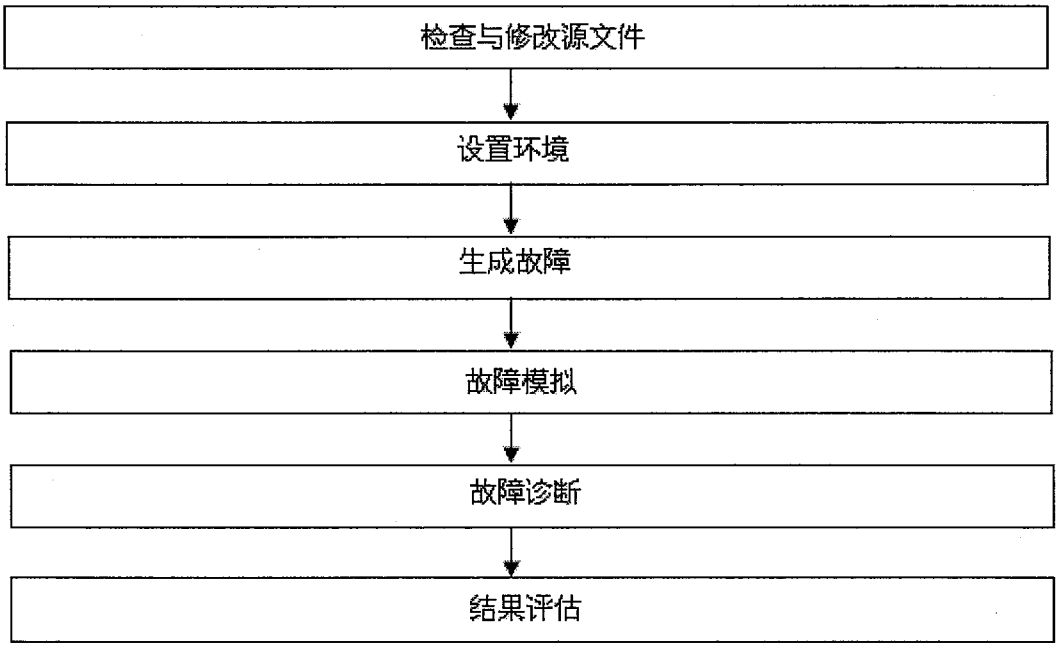


图 3.6　Yield Explorer 使用流程

图 3.6 展示的是软件的经典使用流程。这个流程是能较好发挥软件功能的较为推荐的流程。此流程最大的特点是顺序化，操作思路清晰，有利于尽快得到结果。如果需要根据结果调整输入文件，可以循环使用此流程。所以此流程也提供了逐步完善细化诊断工作的可能。在此流程中，第一步是打开工程，所有的单个工程的源文件都应该存储在一个工程目录中，便于未来的操作和管理。操作软件生成的结果文件也会存储在此目录

中。第二步打开需要查看和修改的原文件，例如网表文件。检查这是否是我们所需要的源文件，是否有错误，语法语义上是否表达了我们意愿中的电路结构。第三步设置环境。修改和检查完所需源文件后，就应该把他们设置成实验环境，这样在进行下一步的工作中就不用来回填写输入文件了。第四步生成故障。在这一步中软件会根据用户的要求进行故障描述文件的生成。第五步是故障模拟，软件利用上一步生成的故障描述文件进行故障模拟，生成失效响应文件。第六步故障诊断功能利用失效响应文件结合所设置的环境进行故障的诊断和定位。第七步利用故障诊断结果和真实故障给出对诊断效果的评估，以便于今后对算法的改进。

3.3 Yield Explorer 软件子模块设计与实现要求

下面对 Yield Explorer 故障诊断软件的各个子模块的设计进行详细的介绍。

3.3.1 操作接口模块设计与实现要求

对于操作接口，我们主要采取以菜单操作为主，工具栏操作为辅的操作方式。同时还有少许的其他操作方式。目的就是要使操作方式清晰化。同属一大类的操作方式要在空间位置上接近。

大体的初试界面组成已在图 3.5 中进行了展示，以下进行具体的介绍。

菜单栏：

菜单栏包括了所有软件的主要功能。主要分为七个大类。File 类主要负责工程与文件的操作，Environment 包括了输入文件的设置。Netlist Analysis 包括了对电路的查询功能。Fault 包括故障的生成。Simulation 包括故障模拟功能。Diagnosis 包括所有故障诊断的功能。Statistical Analysis 包括了所有诊断结果的统计显示。

对于 File 菜单，我们采取了两层菜单的形式。因为这一菜单中的功能较多，且可以形成两个大类，所以，在下表中，我们把菜单的组织结构列出。在第一个菜单中，有“+”号的菜单项是第一层菜单项。如果以下有不带“+”的菜单项，则这些菜单项从属于上一层菜单项。

菜单栏的具体结构如表 3.1：

表 3.1 菜单组织结构

File	Environment	Netlist Analysis	Fault	Simulation	Diagnosis	Statistical Analysis
+Project	Netlist	Single Net Analysis	Random Fault Generation	Fault Simulation	Fault Diagnosis	Diagnosis Result Evaluation
Open	oaNetlist	Double Nets Analysis			General Fault Diagnosis	Diagnosis Results Statistics
Close	Verilog					
Create	Pattern					
Copy	STIL					
Rename	Layout					
+File	oaLayout					
Import	Lef/Def					
Create	Compactor					
Copy						
Rename						
Save						
+Quit						

在以上菜单项中，我们用点击后弹出对话框的方式来进行输入。针对每种功能的不同特点我们设计了不同的对话框。针对这些功能，我们对其中一些典型菜单项的功能和对话框进行重点介绍。

（1） File→Project→Open：打开一个 Project。由用户使用 Choose 选择需要打开的 Project 文件夹或者直接输入 Project 文件夹的路径。输入完成后，打开 Project，在程序左侧显示 Project 文件树。

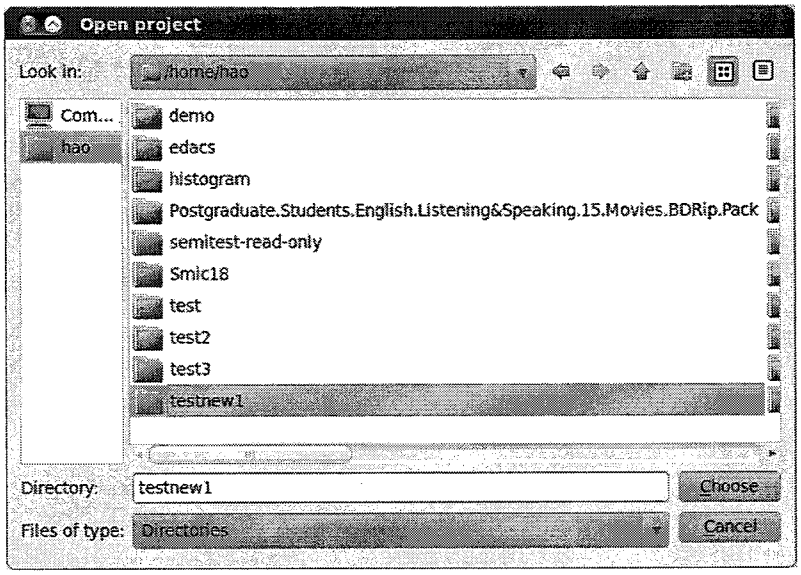


图 3.7 打开工程对话框

（2） File→File→Import：导入一个文件。由用户输入 a）使用 Choose 选择需要导入的文件或者直接输入该文件路径，b）使用上方树形列表选择需要导入到的 Project 文件夹。输入完成后，该文件被复制到指定文件夹下。

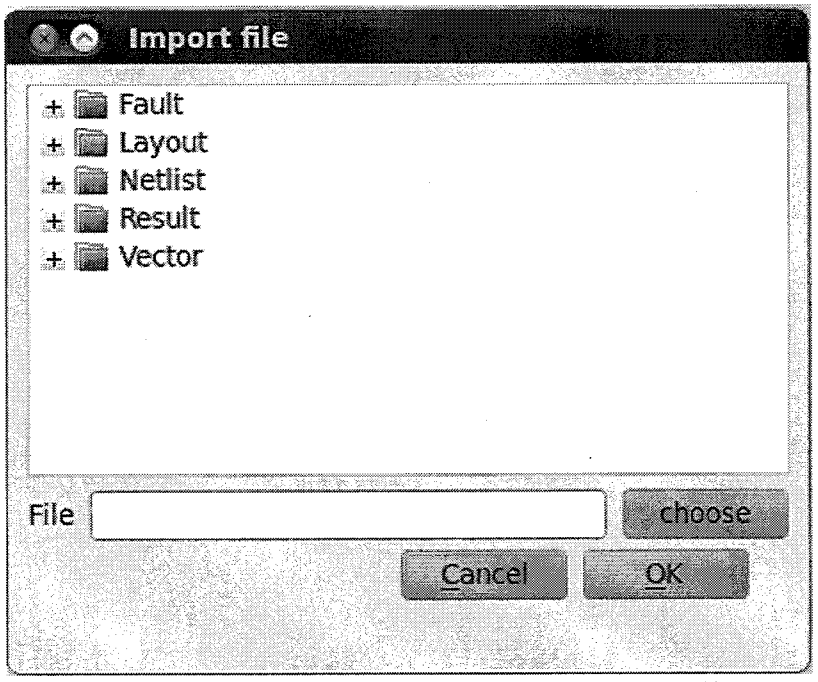


图 3.8 导入文件对话框

（3） Environment→Netlist：读入本小组自有格式的网表文件，可以通过按钮选择文件，也可以直接输入文件路径。

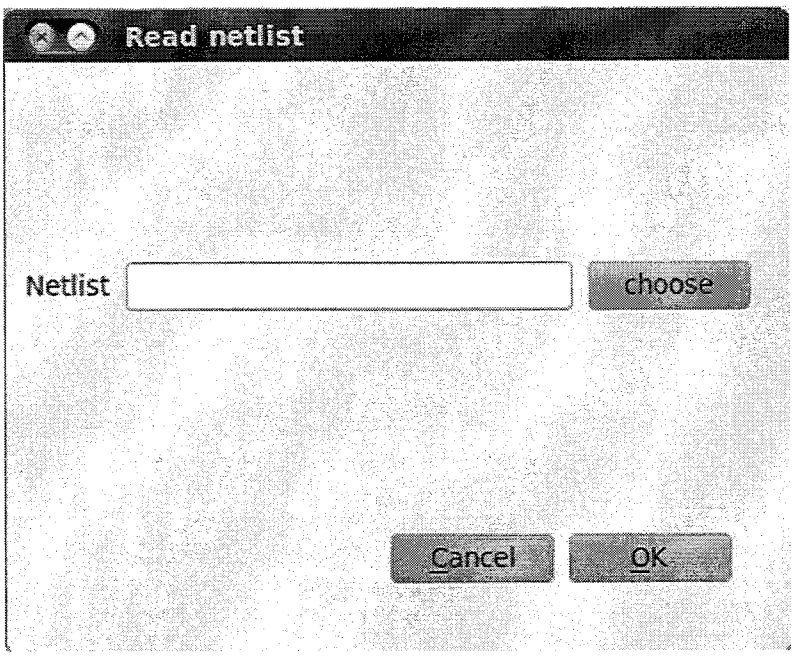


图 3.9 导入文件对话框

（4） Net Analysis→Single Net Analysis：分析一根线的连接关系。用户可以自定义临时环境，即临时改变输入文件，由“Temporary Environment Setup”完成。用户输入 Net 并选择需要分析的内容，一共有四种内容，以多选列表形式组织。默认情况下都进行分析。输出结果写入到 Project 的 Result/Netlist Analysis/Single Net Analysis.log。

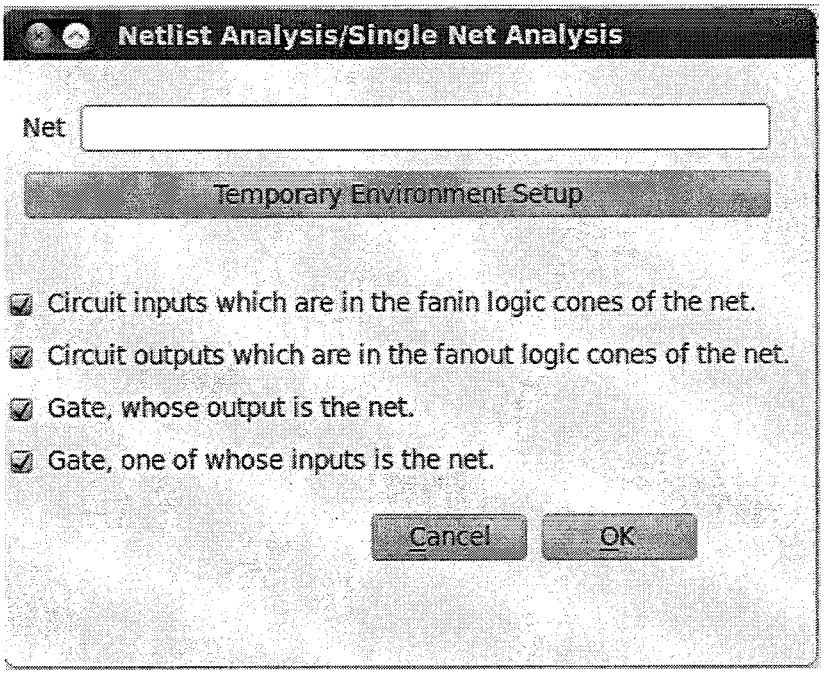


图 3.10 单连线分析对话框

（5） Random Fault Generation：随机产生故障。最上部的选项控制故障生成的特点。下方有三个单选按钮，分别对应按数量生成故障，按类型生成故障，按数量和类型生成故障。当其中一种情况被选择时，其他区域会变成灰色未激活状态。产生的故障保存在 Project 的 Fault/flt 文件夹下，一个故障对应一个文件。

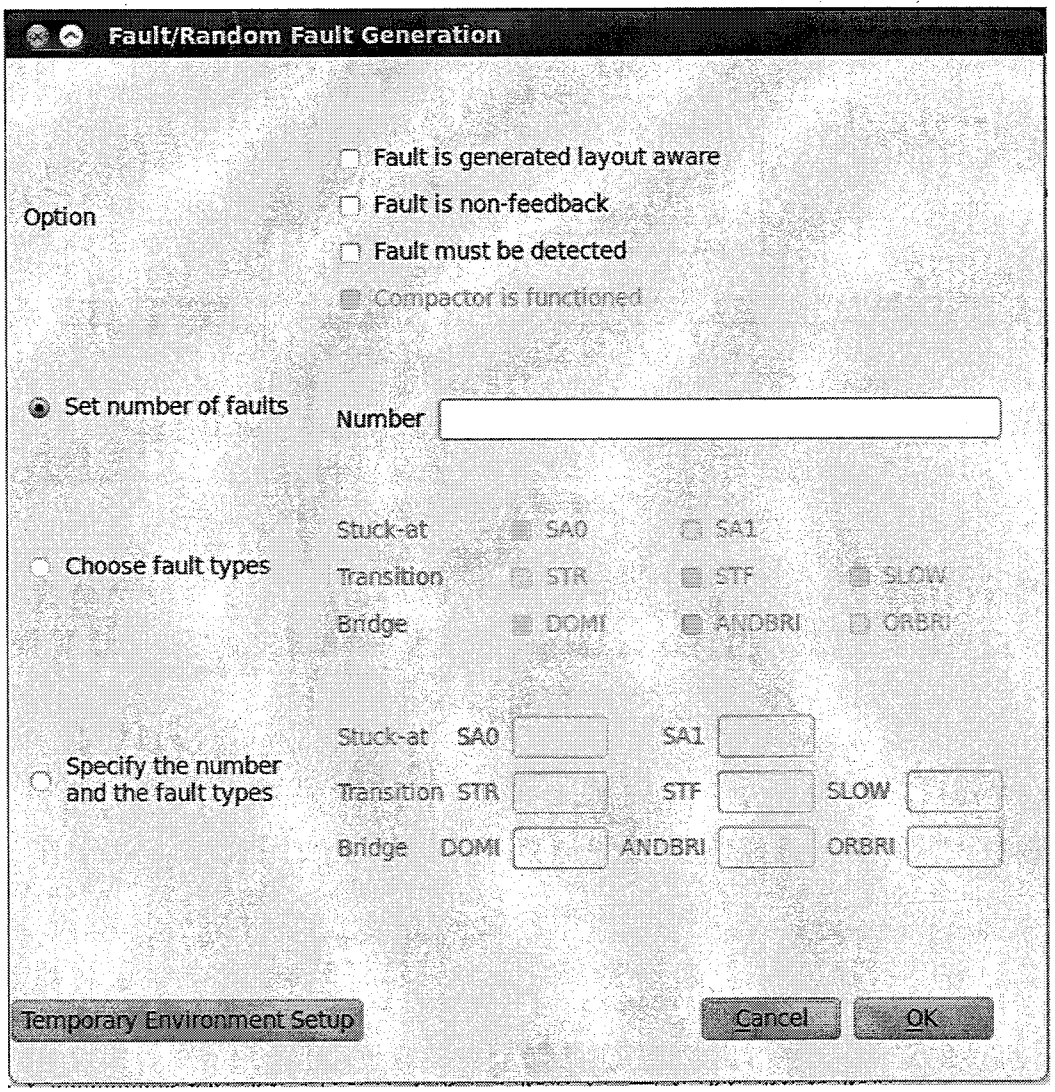


图 3.11 生成故障对话框

操作接口模块小结：采用统一有续的操作方式有益于操作人员快速上手软件的使用。同时针对每种功能设计不同的对话框，满足了不同操作的不同特点，使各项功能能够发挥应有的作用。

3.3.2 文本编辑模块设计与实现要求

此模块是我们在研究相关软件之后决定要增加的模块，也是本软件体现出用户友好

特色的模块之一。如果电路网表文件和向量文件能够在故障诊断软件环境中直接编辑，则可提高工作效率和灵活性，因此，本软件增加了文本编辑模块。与文本编辑相关的区域大致包括工程结构区域，工具栏区域，文本编辑框区域。

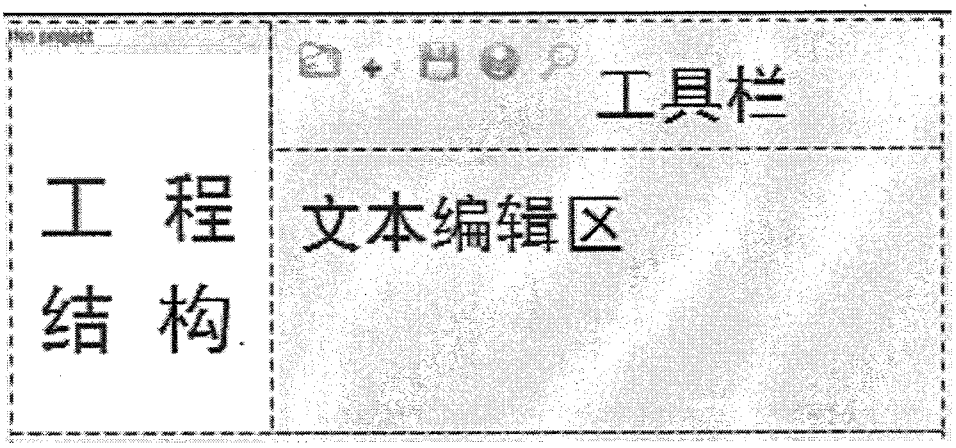


图 3.12 文本编辑相关区域

在打开工程后，工程结构区域就会显示打开的工程文件夹的树形结构。

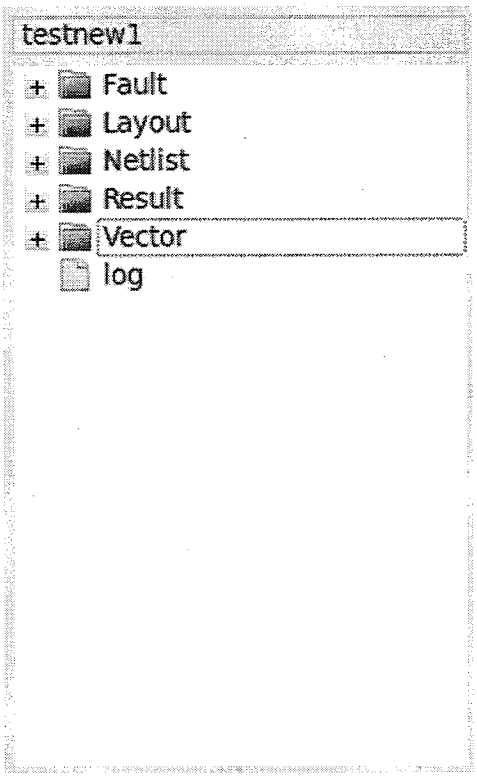


图 3.13 树形结构

对于以上树形结构，文件夹左侧都有相应的加号，点击就可显示此文件夹的内部文件。对于文件编辑，第一步是打开相应的文件。打开的方式是双击要打开的文件。我们以 log 文件为例。双击打开此文件。

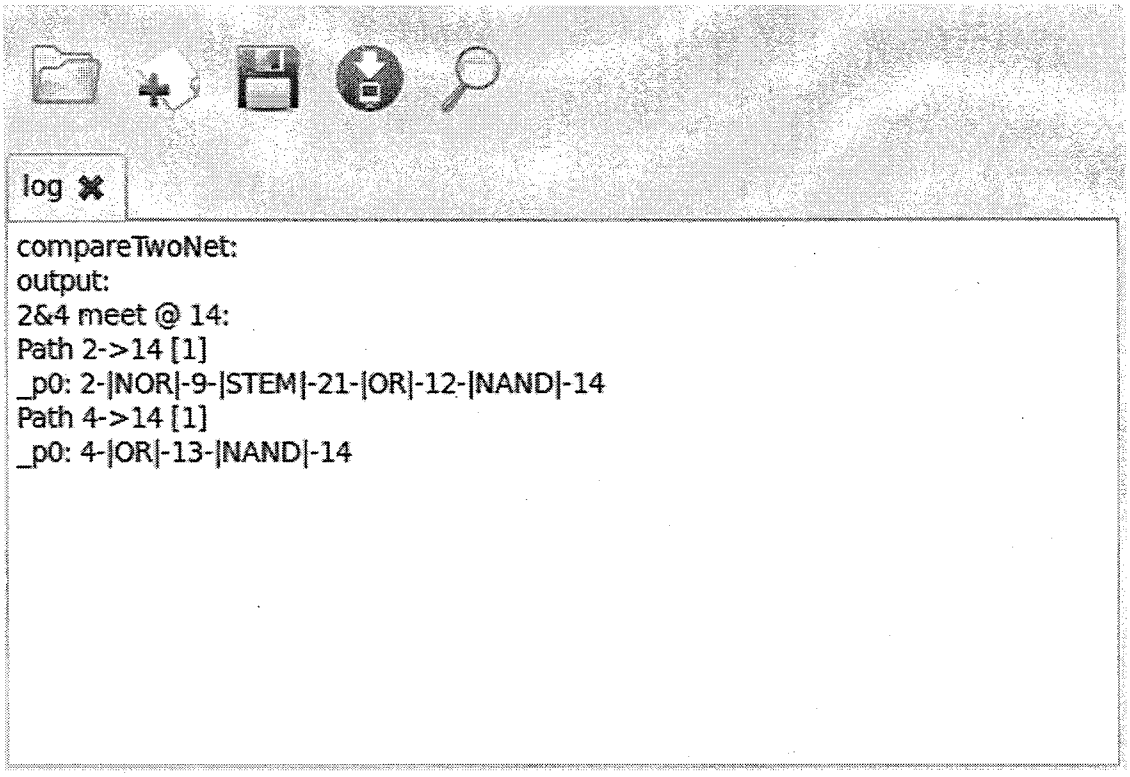


图 3.14 文件多标签显示

可以看到，对于打开的文件。我们用多标签的方式进行显示。文本显示之后，上方的工具栏图标也被相应的激活。原先不可以被使用的图标在当前情况下可以使用后，会变为彩色显示。

我们把工具栏图标按自左到右的方式排列如下：

- 打开图标：点击后弹出对话框询问要打开的文件，确认后在新标签中显示文件。
- 新建图标：点击后弹出对话框询问文件的创建目录，确认后创建文件并在新标签中显示文件。
- 保存文件图标：点击后把在文件显示区域的修改内容保存在磁盘文件上。
- 另存为图标：点击后弹出对话框询问新文件的目录，确认后在此目录生成和当前文件一样的新文件。
- 查找替换图标：点击后弹出对话框询问要查找替换的内容，确认后会高亮显示文本中要查找的内容，并可替换。

另外，在右击文本编辑框的时候还有编辑相关临时菜单。如图 3.15。

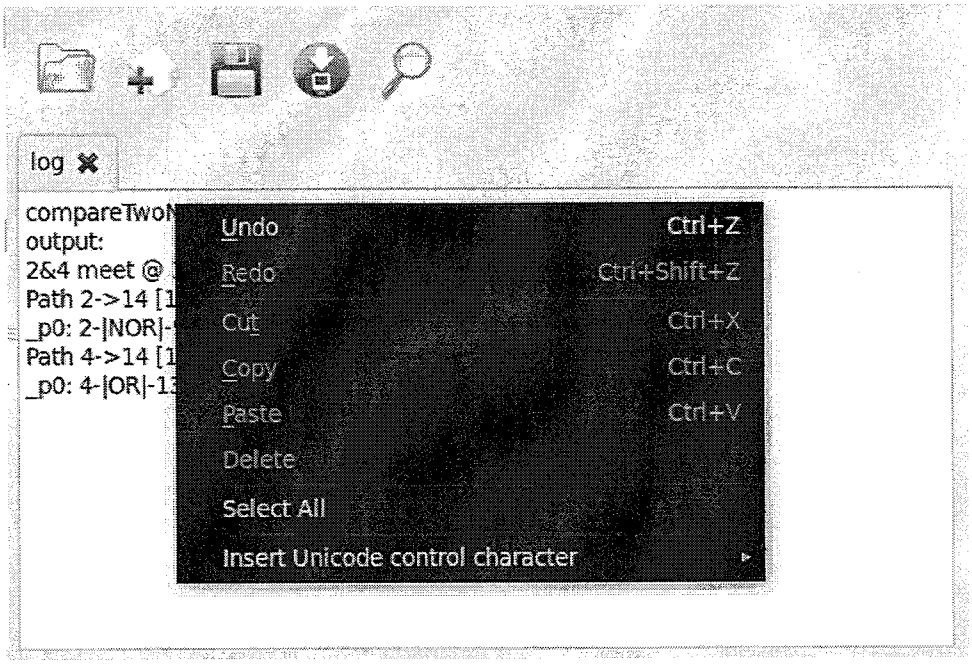


图 3.15 右键菜单

从以上菜单可以看出，菜单上的功能补充了工具栏上提供的功能。比如撤消和重做功能。在可以使用上述功能时选项为白色激活状态。另外，在关闭文本文件所在标签时，软件会检测文件是否有改动，若有的话，会提示保存，询问对话框如下：

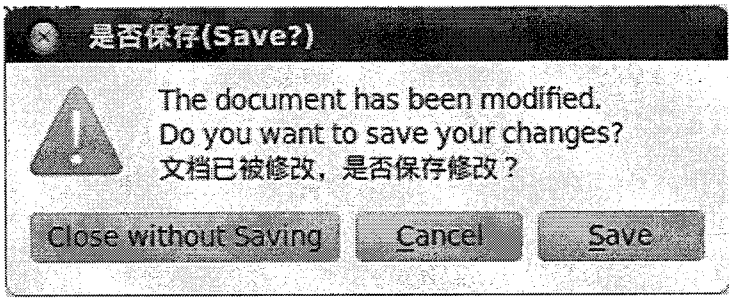


图 3.16 保存提示对话框

文本编辑模块小结：此模块注重用户体验，以快捷和多功能为主。同时还考虑利用多标签方式一次性展示所要操作的文件，提高工作效率。另外考虑了文件未保存的情况，加强了工作的安全性。

3.3.3 输入解析模块设计与实现要求

对于输入解析模块，我们主要设计针对自有格式的网表文件和向量文件，Verilog 文件，LEF/DEF 文件，STIL 文件，Open Access 专有格式的网表文件和版图文件的解析。

对于自有格式网表文件和向量文件的解析，Yield Explorer 直接集成了原有算法代

码。对于这种格式的支持主要是便于本人所在研究小组开展相关实验。因为格式简单，从此类文件提取相关信息非常迅速和方便。

我们对自有格式的文件进行简单介绍，用网表文件进行举例说明。

```
1 NumNet: 3
2 NumPI: 4
3 PI: 1 2 3 4
4 NumPPI: 3
5 PPI: 5 6 7
6 NumPO: 1
7 PO: 18
8 NumPPO: 3
9 PPO: 17 28 11
10 NumScanChain: 1
11 ScanChain: [NumCell Cell]
12 3 5 6 7
13 Net: [NetName Logic NumPred Pred NumSucc Succ]
14 1 1 0 1 8
15 2 1 0 1 9
16 3 1 0 1 11
```

图 3.17 网表自有格式

第一行为所有互连线的数目，第二行为伪输入的数目，第三行为伪输入的具体编号。以下四到十二行类似。第十三行以后记录的是互连线的信息。第一个数字为互连线编号，第二个数字为互连线靠近输入端所连接的逻辑门类型。第三个数字为互连线的前驱数，接下来是具体编号，第五个数为后继数，接下来为具体后继编号列表。由以上格式可以看出，只要用读取数字的程序语句对数字逐个读取，就能实现自有格式的信息提取。

对于 Verilog 文件，我们采取利用 Open Access 自带的转换程序来进行解析。对于 LEF/DEF 文件，Open Access 也带有相应的转换程序。所以这两种文件的解析和 Open Access 专有格式的解析方法差不多。Verilog 文件和 LEF/DEF 文件的解析有一个转化为 Open Access 数据库格式的步骤。

我们以 Open Access 格式的网表文件的解析来进行说明。首先相应菜单项调用回调函数，回调函数调用读取网表文件的函数，此函数利用 Open Access 的函数接口提取所要信息并保存在临时数据结构中。下一步把提取后的信息转移至后续功能操作所用的 C++类中。

我们用图 3.18 来表示：

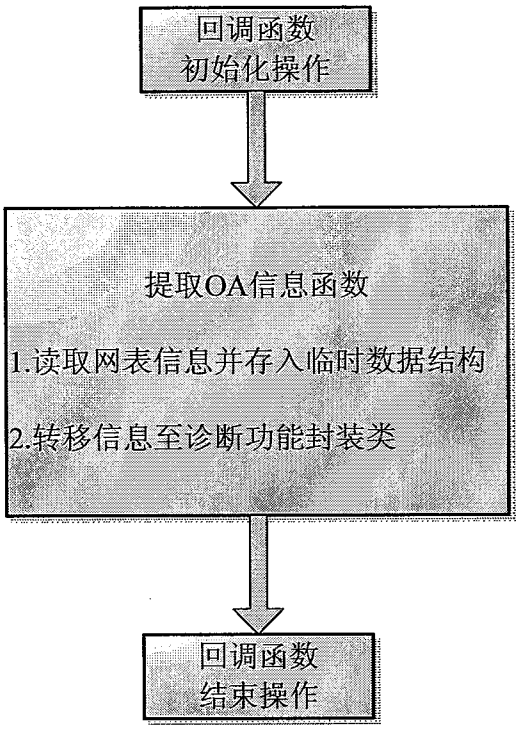


图 3.18 OA 网表解析流程

以上是 OA 网表解析的大致思路。具体的解析过程我们留待下一章再进行讲解。

对于 STIL 文件，需要利用 Lex/Yacc 工具和相关开源代码完成解析工作。

Lex 是生成扫描器的一种语言，而扫描器是一种分辨原代码中符合定义的字符串的程序。这些定义主要用正则表达式来进行规范描述。Yacc 能将任何编程语言的语法定义转化为解析此种语言书写的程序的语法解析器。它用巴科斯范式(BNF, Backus Naur Form)来书写。

对于 Lex/Yacc 工具来书写 STIL 文件解析器的大致方法如下。利用 Lex 工具用正则表达式来定义例如关键字和变量名等语法元素。然后利用 Yacc 工具来定义语句、类、结构体等语法结构。之后再在这些语法结构定义之后书写动作。也就是检测到这类结构时要做的操作。那么在 STIL 文件解析中最关键的动作就是检测到输入端口名字的描述语句以及描述端口对应信号的语句要做对应的保存操作。把这些语句中的名字和信号按顺序存起来以后，再把这些信息对应起来存入相应数据结构。

读取 STIL 文件流程如图 3.19：

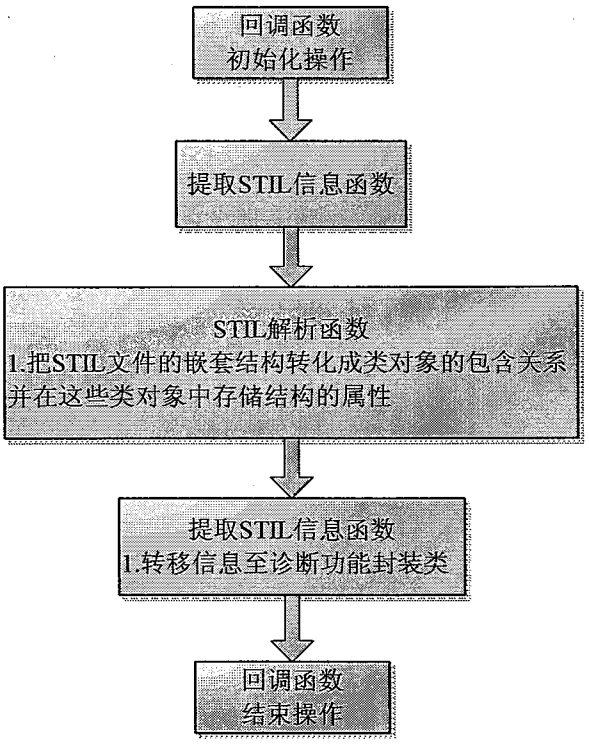


图 3.19 STIL 解析过程

图 3.19 展示了解析 STIL 文件过程中函数的调用过程和所包含操作。对于具体的实现细节我们在下一章再进行说明。

输入解析模块小结：此模块是正确进行主要诊断功能的前提。由于需要支持多种文件格式，同时每种格式可利用的程序和代码又有所不同，所以每种格式所用的解析方法各有不同。但优先利用已有资源进行编写，这样可最大限度地保证解析过程的正确性和缩减开发时间。

3.3.4 核心算法模块设计与实现要求

对于这一模块，是软件的最重要部分。软件功能的丰富性和有效性主要取决于这个模块。我们的核心功能包括电路信息查询、生成故障、故障模拟、多故障假设的诊断、单故障假设的诊断、诊断结果显示。

由于电路信息查询、生成故障、故障模拟、多故障假设的诊断四项功能已有师兄写好的基于控制台的 C++代码，所以，对这些功能的处理过程为，在回调函数中传递参数调用这些功能处理函数，同时重定向输出，从输出文件中提取结果再进行显示。

对于单故障假设诊断，没有现成程序可以利用。大致过程如下：

（1） 回调函数中先接收 STIL 用户输入。

（2） 处理 STIL 文件。

（3） 对互连线名进行数字化处理。

（4） 对每条向量循环模拟可能的故障位置并针对原来的结果进行取交集操作。

具体操作算法在下一章再进行介绍。

对于诊断结果显示，我们有两种显示功能，一种是评估此次诊断结果并图形化显示历次诊断结果，一种是只图形化显示以往历次结果。

评估此次诊断结果并图形化显示历次结果的大致过程如下：

（1） 回调函数中接收用户输入。

（2） 生成输入参数并调用诊断评估函数。

（3） 将评估函数结果写入历史记录文件。

（4） 对历史记录文件进行统计显示。

图形化显示以往结果的功能只有一个函数调用，就是对历史记录文件进行统计显示。这一步和上一功能的第四步是一样的。我们用同一个函数做这一项功能。这一个功能所用的历史记录文件是我自己制订的格式，对于这个格式和具体算法我们在下一章再进行介绍。

核心算法模块小结：核心算法主要注重功能，既要满足学习研究的需求，所以添加多故障诊断和结果评估功能；又要满足实际诊断工作的需要，所以添加基于 STIL 文件的单故障假设诊断功能。

3.3.5 结果展现模块设计与实现要求

结果展现承担着传递有效信息给用户的责任。有的结果不详细报告就可以，有的结果需要保存在日志文件中以便今后查看，有的结果只需文字显示，有的结果需要图形显示。

要不要保存运算结果在日志文件中，我们根据输出的情况进行讨论。我们发现，根据输出分类，输出分为两类。一类是即时通知性输出；一类是目标文件输出。下面详细地解释一下这两类输出。即时通知性输出：例如查看电路信息的功能。如果我们只想知道某根信号线附近的一些信号线。那么我们就应该把这些信号线的名字代号按照整洁的格式用文本输出出来。此种输出的特点就是利用自然语言快速通知用户结果。我们把这

类功能的结果在界面的消息通知区域中进行报告，同时把结果存在记录文件中，在软件的文本编辑框中就可查看。目标文件输出：有的功能并不是要通知用户结果，而是要生成可进一步处理的文件。例如生成故障功能。它生成的故障文件，其中保存着对故障的描述信息。此文件要在故障模拟功能中使用。对于这种输出，我们把目标文件存在工程目录的相应子目录中。此时通过软件左侧的工程目录就可查看这些文件。

对于图形显示的决定，我们发现，不用报告评价指标的结果只需用文字就可说清结果，所以不用图形化的显示。而报告指标的结果同时也是展示历次数据的结果，所以适合用柱状图来展示历次结果，这也便于结果的对比。

结果展现模块小结：由于对功能的特点进行了分类，所以结果呈现的问题就显得清晰化了许多。为每类功能的结果显示进行了有针对性的设计后，用户对结果的理解吸收和备份查阅这两项要求都得到了较好的满足。

3.4 本章小结

本章首先具体化了 Yield Explorer 故障诊断软件的各项功能需求，之后通过分析成熟 EDA 软件的特点给出了 Yield Explorer 故障诊断软件的具体目标。同时，根据目标设计了 Yield Explorer 故障诊断软件的总体结构，将系统分为了操作接口模块、文本编辑模块、输入解析模块、核心算法模块、结果展现模块。对各个子模块进行了功能划分，也详细给出了各子模块的设计哲学、方案与特点。同时对各子模块的大体结构进行了概括的介绍。

第4章 Yield Explorer 故障诊断软件实现

基于上章 Yield Explorer 故障诊断软件的具体设计，本章介绍故障诊断软件的具体实现方案，具体到类与函数的组织方式。同时对牵涉到的实现工具与语言进行简单介绍。对于各个子模块的实现，我们分别进行说明

本章的章节安排如下：4.1 节大体给出软件开发的必备工具，比如开发环境、开发工具的选取。4.2 节介绍软件的总体架构，包括各个 C++类的封装，重要函数的功用以及重要变量的作用等。4.3 节从子模块的角度对重要的类和函数从结构和算法的角度进行详细说明。最后是对本章的总结。

4.1 Yield Explorer 软件开发基础

4.1.1 开发环境

大部分 EDA 专业工具主要是运行在 Linux 操作系统下。为了与这些专业工具兼容，也必须使我们的软件支持 Linux 操作系统。所以 Yield Explorer 故障诊断软件的开发环境就选择了 Ubuntu 操作系统。一是在其上编写与编译的软件必然是支持 Linux 系统的，二是考虑到此操作系统的易用性与普遍性。开发语言采用 C++语言，此语言特性丰富，同时效率也较高。在编译工具的选择上，我们使用了 g++。此工具选项丰富，编译效果好。

4.1.2 开发工具的选择

为了加快开发的速度和降低开发的成本。我们在实际开发的过程中运用了一些开发工具。主要有：Qt、Lex/Yacc、Open Access、Qwt。

（1） Qt：Qt 是一个适用于多种系统的 C++ 图形用户界面库，生产厂商为挪威 TrollTech 公司，此工具包中包括 Qt，界面绘画工具 Qt Designer，基于 Framebuffer 的 Qt Embedded，语言多国显示工具 Qt Linguist 等。Unix 平台支持 Qt。我们主要看重它美观的开发效果和对 C++语言的良好支持。

（2） Lex/Yacc：Lex/Yacc 主要是开发编译器的工具。Lex 是生成扫描器的一种语言，而扫描器是一种分辨原代码中符合定义的字符串的程序。Yacc 能将任何编程语言的语法定义转化为解析此种语言书写的程序的语法解析器。我们利用它进行 STIL 文件的解析。

（3） Open Access：Open Access 是 Cadence 开发的辅助 EDA 工具开发的函数库和数据库。因为它提供了一个可共享开发数据的平台，所以我们选择支持它的数据格式进行各项功能操作。于是自然利用到了它的函数库。还有，我们利用了它的转化工具进行某些文件的转化。

（4） Qwt：Qwt 是 Qt 的一个辅助函数库。主要是用来进行图形绘画操作。我们利用此工具进行统计图显示的算法开发。

4.2 Yield Explorer 软件总体架构

在各子系统实现过程中，各子系统需要进行频繁的交互。对此，各子系统需要相对独立又要相互协作。同时为有效地实现代码重用，同时便于系统的维护和升级，软件的源代码采用对功能进行划分的封装，同时利用全局变量传递重要消息的结构。在软件中有以下几种重要的 C++类。

现在对重要的类介绍如下：

界面类：主要负责界面的显示和跟用户的互动行为。在这种类中，主要分为两个大的类型：主界面与对话框。在界面类中最主要的数据部分是界面元素。最主要的函数是回调函数。通过把回调函数与界面元素进行连接，对界面的交互动作如点击等就会调用回调函数运行。我们用 MainWindow 类来举例。MainWindow 的大体结构如图 4.1：

MainWindow：类名
数据对象：
ui::MainWindow*ui：包括工具栏、菜单栏、文本框的元素集合的类对象的指针。
HTextEditor* tabtext：包含文本编辑框的类对象的指针。
……
回调函数：
void create_project()：创建工程的回调函数。其中调用其他界面类进行对话框的显示。
void show_menu(const QModelIndex &)：右键菜单的回调函数。针对工程区域的右击位置显示右键菜单。
……
公共函数：
void getinput(Command_class &YE,int netnum,QString &res)：其他回调函数调用的函数，此函数的具体作用是计算选定的信号线的前驱。
……

图 4.1 MainWindow 类结构

以上介绍了 MainWindow 类的大体结构，用……代替了未写出的函数与数据。此类中包含了大概 40 个函数，是最大的界面类。对于其他的对话框界面类，规模要小很多，不过对话框类数量多，大概有接近 30 个，具体结构与 MainWindow 类似，就不具体介绍了。

主体功能类：此大类主要利用师兄的现有代码。主要包括基础操作类，诊断功能类。基础操作主要包括整型数组类，浮点数组类。基础操作类主要是针对软件的具体功能要求设计的有利诊断功能开发的数据结构以及其上的运算操作。诊断功能类就是利用基础操作类来进行诊断算法实现的类，用多层继承的方式简化代码的逻辑结构，便于维护。

ArrayInt：类名
数据对象：
int MaxSize：此数据定义数组的最大长度。
int *Ptr：这是数组的具体指针。
公共函数：
int &operator[](int)：此公共函数允许直接对数组的某一位进行赋值，如果数组长度不够，会自动分配空间进行加长。
……

对 ArrayInt 进行操作的普通函数：
extern int FindInArray (int length, int *array, int target)：根据数值在 array 所指向的数组中查找此数据所在的数组序号。
extern void PrintArray (int length, ArrayInt& array)：对数组的元素进行格式化的打印。
……

图 4.2 ArrayInt 类实现

图 4.2 是一个典型的基础操作类。在 ArrayInt 类中，类中的函数主要进行赋值操作。还有大概 20 个普通函数对 ArrayInt 类对象进行其他操作。比如查找操作，算术操作，打印操作等。在上图的横线下给出了两个实例说明。对于 ArrayFloat 类，大体结构与 ArrayInt 类相似，在此不做具体介绍。

对于诊断功能类，主要有四个主要的类：FileAnalyst、CircuitMap、FaultSimulation、FaultDiagnosis。其中 CircuitMap 类继承 FileAnalyst 类。FaultSimulation、FaultDiagnosis 类继承 CircuitMap 类。

对于 FileAnalyst 主要保存网表、版图、向量等输入文件的具体信息。大体结构如图 4.3：

FileAnalyst：类名
数据对象：
主要利用基础操作类来保存文件信息。
公共函数：
void ImportCC(const char *filepath=FILEPATH_NETLIST_CC)：导入网表文件的函数。
void ExportWFFV(const char *filepath=FILEPATH_VECTOR_WFFV)：导入向量文件的函数。
……

图 4.3 FileAnalyst 类结构

FileAnlyst 中的公共函数主要进行文件的输入输出操作，以及输入输出相关操作所需要的其他操作。具体有大概 50 个函数，在此不一一介绍。

CircuitMap 主要负责网表层次的信息的查询工作，网表信息利用 FileAnalyst 进行保存。

CircuitMap：类名，继承 FileAnalyst。
数据对象：
无，主要利用 FileAnalyst 类来存储信息。
公共函数：
void GetOutput(int net, int &numoutput, ArrayInt &output)：获得电路的输出端口。
void GetInput(int net, int &numinput, ArrayInt &input)：获得电路的输入端口。
……

图 4.4 CircuitMap 类结构

CircuitMap 的公共函数大概有 14 个。

FaultSimulation、FaultDiagnosis 都属于主体功能类，我们以 FaultDiagnosis 进行介绍。

FaultDiagnosis：类名，继承 CircuitMap。
数据对象：
无，主要利用 FileAnalyst 类来存储信息。
公共函数：
void FaultDictionaryConstruction (char *filepath=FILEPATH_FAULT_DICT,
int cratio=0)：构建故障字典的函数。
void FaultDiagnosis(int &numcandidate, Candidate_class *candidate,
int optionlayout=0, int numdominate=0,
int optiondict=0, char *filepath=FILEPATH_FAULT_DICT,
int optionreportiter=0)：对故障进行诊断的函数。
……

图 4.5 FaultDiagnosis 类结构

FaultDiagnosis 的类函数主要包括故障诊断功能的运行，诊断结果的报告等。大概有 5 个可调用的公共函数和 20 个内部私有函数。

值得一提的是 FaultDiagnosis 类中的诊断算法是面向多故障假设的。下面把这种诊断算法的大概过程说明如下：

第一步先进行初始故障元组的建立。通过初始故障元组推导故障逻辑锥，如图 4.6。

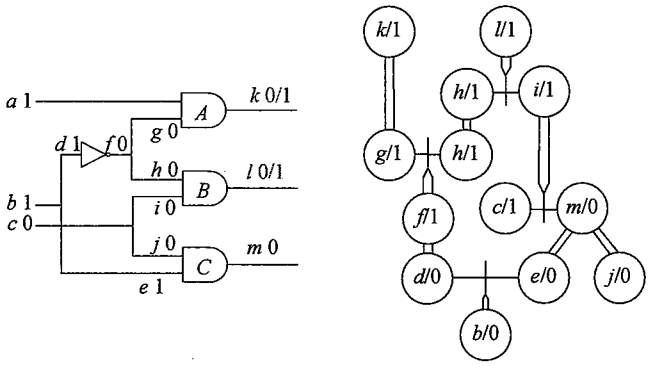


图 4.6 故障逻辑锥建立

第二步对这些潜在故障进行评分。如果此故障能单独解释一个失效响应，分数 1。如果 N 个故障联合起来等价于一个已得分的故障，则这 N 个故障都得到已评分故障的得分的 N 分之一，；评分过程如图 4.7。

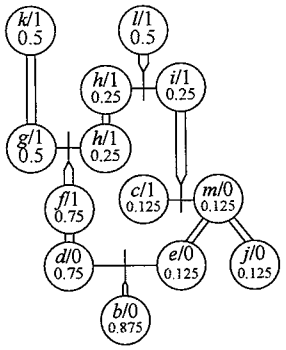


图 4.7 故障打分过程

下一步是候选故障位置选择，这一步选择得分最高的故障，这个是被认可的故障，然后是根据候选故障进行剪枝，递归选择故障。

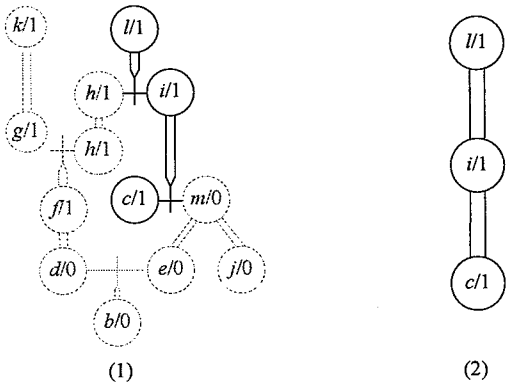


图 4.8 故障选择与剪枝

接下来是基于 OA 的诊断算法类。这个大类包括存储网表信息的结构体、存储版图信息的结构体以及基于这两个结构体的诊断算法类。

我们把网表信息结构体用图 4.9 表示：

netdata：类名。
数据对象：
oaString　　name：字符串名字。
oaUInt4　　dname：数字名字。
oaUInt4　　logic：信号线输入端所连逻辑门类型。
oaUInt4　　numpred：信号线的前驱信号线的个数。
oaString　　*pred：前驱信号线字符串数组。
……
公共函数：
构造函数和赋值函数等。

图 4.9 网表信息结构体结构

网表信息的结构体中的数据元素类型使用的是 OA 函数库提供的类型，便于利用 OA 函数进行后续操作。后续的利用 OA 读取信息的操作使用 opnWalkHier 类。

opnWalkHier：类名。
数据对象：
netdata* nets：存储网表中每条线的信息。
oaString* pi：输入端口数组。
map<oaString,oaUInt4> mapname：线名到数字的映射。
……
公共函数：
void iterateInstance()：将 OA 数据库的信息转入内部。
FaultDiagnosis 中的数据元素中。
……

图 4.10 opnWalkHier 类结构

对于 opnWalkHier 类数据对象存储从 OA 数据库得到的信息。在 iterateInstance 函数中提取 OA 信息到数据对象后，再把这些信息转存到 Faultdiagnosis 中的数据对象中，因为大部分主体功能是基于 Faultdiagnosis 中的数据对象进行运算的。存储版图信息的结构体与提取版图信息的类的结构与以上情况类似，不再做具体介绍。

此外还有一个MainWindow中的回调函数专门负责基于以上结构体的单故障假设的诊断，具体算法在后面的小节再进行介绍。

文件解析类：主要是 STIL 文件解析的类定义。STIL 文件解析主要是通过 Lex/Yacc 工具进行生成的。类的结构较为复杂。STIL 解析类基本按照 STIL 文件的结构，对 SignalGroup、Procedure、Timing 等 STIL 语法定义的结构相应地创建一个 C++类。并且按语法结构的包含关系创建层次化的嵌套的 C++类对象来表示 STIL 文件。

以上是各个主要的 C++类的类型。他们的类函数的调用关系如图 4.11：

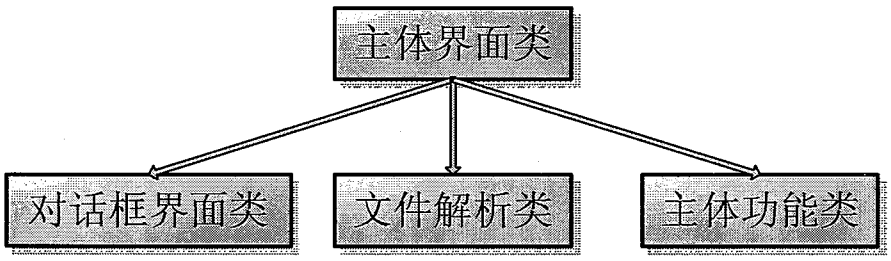


图 4.11 类对象调用关系

软件由主界面的回调函数调用其他各种类对象的公共函数。主从关系清晰，具体进行运算的类便于升级时替换。

4.3 Yield Explorer 软件子模块关键实现

本节将对上章的各个逻辑子模块进行关键函数的详细阐述。重点对各子模块中的核心功能的实现方式进行阐释，包括关键数据结构、核心功能流程的算法描述以及其他实现中的重要元素等。因为对子模块的具体实现细节进行完全阐述会非常冗赘，故此，在本节中会以各子模块为基本单位，利用有限篇幅对核心功能函数的算法进行充分的描述。

4.3.1 操作接口模块关键实现

软件实现要求的功能的第一步就是与用户交互。交互所接受的信息是否全面准确就非常关键。同时交互后所得到的信息要交给具体功能函数进行使用，而这些原始信息可能不能直接利用，那么对操作接口还要求能够对输入信息进行转化。

通过以上要求，我们对操作接口的调用过程总结为算法 4.1：

```
变量定义;
if(对话框返回值)
   临时数据=对话框中输入值;
else   return;
正式输入=con(临时数据);
func(正式输入);
```

算法 4.1 接受数据与转化

以上算法过程需要说明的是第二步，if 语句中的对话框返回值是用来判定用户是否进行输入的。第四步在无输入的情况下直接返回主循环中重新等待事件。第五步调整数据格式。其他步骤不用再详细说明。

以上接受信息的过程由菜单项或按钮等元素的事件进行调用，在总体设计中已经讲过，所以接受数据的具体实现相对简单，不必再继续详述。

4.3.2 文本编辑模块关键实现

文本编辑模块在界面上包括工具栏和文本编辑框。另外，由工程区域的双击也可开启文件进行编辑。所以此模块封装了代表工具栏图标和文本编辑框的元素。而文本编辑框用多标签框作为框架，用文本控件添加在此框架中。以上两项是数据元素。对于函数元素来说，主要有打开，保存，搜索等。用此类定义的类对象被包含在主界面对应的类

对象中。

Editor：类名。
数据对象：
QCommandLinkButton *commandLinkButtonHTextEditorOpen：打开按钮指针。
QCommandLinkButton *commandLinkButtonHTextEditorNew：新建按钮指针。
……
回调函数：
void HTextEditorOpenText()：打开按钮回调函数。
void HTextEditorNewText()：新建按钮回调函数。
……

图 4.12 文本编辑类结构

图 4.12 根据前述内容而言，意义明了，不再详述。下面针对一些重要回调函数的具体实现做算法上的介绍。

打开函数：

```
变量定义;
if(对话框返回值)
    文件名数组=对话框输入值;
else return;
for(i=1; i<=文件名数组元素数量; i++)
    当前编辑框数组= this->findTextChildren(文件名数组[i]);
    if(当前编辑框数组数量=0){
        编辑框=new QTextEdit(文件名数组[i]);
        新标签.setchildren(编辑框);
        This->tabWidget->setcurrentindex(新标签); }
    else this->tabWidget->setcurrentindex(当前编辑框数组[0]);
```

算法 4.2 打开函数

算法 4.2 一次可打开多个文件，并设置最后打开的文档为当前文档。程序第五步对文件名进行循环。第六步检查当前文本框中有没有含有此文件的标签，若有，直接转到第十一步，把此标签设置为当前标签。如果没有，则在第八步创建新的包含此文件的文本显示控件。在第九步创建包含此文本控件的标签。第十步把此标签设为当前标签。

查找函数，算法 4.3：

```
变量定义;
当前编辑框= this->findcurrentText();
光标位置=find(查找文本, 当前位置);
if(!光标位置)当前编辑框.setcursor(光标位置);
else {
    光标位置=find(查找文本, 0);
    if(!光标位置)当前编辑框.setcursor(光标位置); }
```

算法 4.3 设置查找字词高亮算法

在算法 4.3 中，第二步找到当前文本。第三步从当前文本开始查找文中是否有要查找的文本。第四步检测是否找到文本，若找到则设置高亮，否则，在第六步从文件开头查找文本。第七步检查是否找到，若找到则设置高亮。

以上讲述了两个函数的算法，这些算法的特点是与 Qt 的内置函数和变量关系密切，要求熟练掌握 Qt。

4.3.3 输入解析模块关键实现

在总体架构设计中已经讲述了输入解析模块包括的类以及与语法结构的对应关系。本节主要讲述如何用 Lex/Yacc 来编写输入解析模块。

在 Lex 文件中主要是定义 STIL 文件中的 token。对于 token 的定义，Lex 使用了正则表达式来表达。具体的正则表达式的介绍见表 4.1。

字符	含义
A-Z, 0-9, a-z	构成了部分模式的字符和数字。
.	匹配任意字符，除了 \n。
-	用来指定范围。例如：A-Z 指从 A 到 Z 之间的所有字符。
[]	一个字符集合。匹配括号内的 *任意* 字符。如果第一个字符是 ^ 那么它表示否定模式。例如: [abC] 匹配 a, b, 和 C中的任何一个。
*	匹配 *0*个或者多个上述的模式。
+	匹配 *1*个或者多个上述模式。
?	匹配 *0*个或*1*个上述模式。
$	作为模式的最后一个字符匹配一行的结尾。
{}	指出一个模式可能出现的次数。 例如: A{1,3} 表示 A 可能出现1次或3次。
\	用来转义元字符。同样用来覆盖字符在此表中定义的特殊意义，只取字符的本意。
^	否定。
|	表达式间的逻辑或。
"<一些符号>"	字符的字面含义。元字符具有。

表 4.1 正则表达式中一些字符的含义

我们举一个 Lex 文件中的实例来说明：

DIGIT [0-9]

HEXDIGIT {DIGIT}|[aAbBcCdDeEfF]

以上定义把 DIGIT 标识符定义为可代表 0 至 9 中的任意数字。HEXDIGIT 定义为数字或 a 至 f 中的任意字母。他们是十六进制中可能出现的字符。

Lex 文件的主体部分是对已用正则表达式定义的标识符定义动作，一般是返回可被 Yacc 文件识别的 token 名称。例如对 HEXDIGITS 返回“_STIL_HEXDIGITS”。“_STIL_HEXDIGITS”会被 Yacc 文件继续利用来匹配 Yacc 文件中定义的语法结构。并在匹配后执行定义的动作。

Yacc 文件用来定义语法的 BNF 范式如下：

term-> term*factor

|term/factor

|factor

expr-> expr+term

|expr-term

|term

Factor-> digit|(expr)

以上 BNF 中的“|”是“或”的意思，是指凡是满足“->”右式的表达式都可以用“->”左式来称呼。而右式中没有出现在左式的表达式就是终止符，也就是 token。这些 token 被 Lex 文件中定义的动作所返回，由 Yacc 文件中的 BNF 表达式来匹配，匹配到定义的结构以后进行相应的动作。这些动作一般是保存这些语法结构的相应信息到对应的类对象中。

4.3.4 核心算法模块关键实现

对于核心算法模块的关键函数，我们进行以下介绍。

基于 OA 数据库的提取电路信息算法：

```
变量定义;
利用 OA 函数库获取 oaNet 类型数据的集合;
创建 netdata 型数组存储这些信号线的名字;
for(每一个 oaNet)
    计算每根线的前驱后继并存入 netdata 数组;
对每个 netdata 型变量数字化，并存入 map 型 STL 容器;
for(每一个 netdata) {
    for(netdata 的每一个前驱字符串名)
        从 map 型容器中查找此字符串名的对应数字名并储存到 netdata 中;
    for(netdata 的每一个后继字符串名)
      从 map 型容器中查找此字符串名的对应数字名并储存到 netdata 中;}
```

算法 4.4 提取电路信息算法

算法 4.4 把 OA 电路信息存储到 opnWalkHier 的私有数据中，并进行了数字化。因为本软件所利用的一部分师兄编写的函数功能对信号线是用数字表示的，所以进行数字化就是希望能够在这些功能中使用。

对于单故障假设算法，大体过程如算法 4.5。

```
变量定义;
利用向量文件生成向量与错误输出的映射 it;
for(每一个 it 中的映射) {
    用故障模拟的方法对每个可能故障模拟，存入临时 map 容器 tempres;
    if(第一个映射)最终结果容器 ultres=tempres;
    else ulres 和 tempres 取交集; }
```

算法 4.5 单故障假设算法

4.3.5 结果展现模块关键实现

我们对此模块介绍画统计图的函数：

```
变量定义;
for(每行在历史日志中的数据) {
    for(i=1;i<2;i++) {
        对第一个指标的一次实验结果的两个数据设置方柱高度，区间长度设置为 0.5;
        记录下区间的最右横坐标和柱子个数以供下次使用; }
    对第二、三、四个指标重复类似第一个指标的循环;
    提取第五个指标的实验数据的个数 c;
    while(c--) {
        设置实验数据 c 的区间长度为 1 除以数据个数;
        根据实验数据设置方柱高度;
        记录下区间的最右横坐标和柱子个数以供下次循环使用; }
    对第六个指标重复类似第五个指标的循环; }
```

算法 4.6 统计图数据设置算法

图 4.14 是统计图效果：

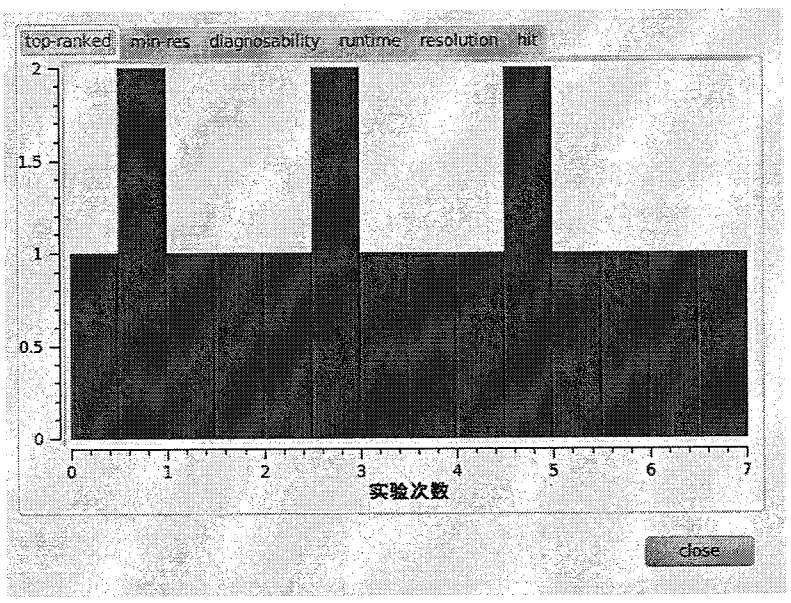


图 4.13 统计图效果

4.4 本章小结

本章首先介绍了 Yield Explorer 故障诊断软件在实现过程中使用的开发环境和开发工具。之后，对软件的总体架构进行了详细的介绍，包括 C++类的类型以及这些类从属于几种类型。对这些类型中的一些关键的 C++类的封装作了较完整的介绍，让读者了解到这些类中的数据和函数都有什么作用。介绍完总体情况后，阐述了各个子模块中关键问题的实现。从实现方法或算法的角度详细说明各个子模块的重点难点。对于算法还给出了伪代码的具体描述。实现过程中专业开发工具和函数库的使用使得开发速度加快，开发质量提高；代码按照不同的功能划分到不同的类封装使代码易于维护和升级。

第5章 结束语

随着集成电路工艺的发展，芯片所使用的工艺特征尺寸不断缩小，电路的规模不断增大，系统工作的时钟不断提升，这种芯片产业的发展趋势会带来更高的缺陷密度。在这种情况下，故障诊断对于检测故障和提升成品率就具有重要意义。故障诊断软件作为诊断人员的辅助工具在新的行业发展趋势下需要有新的特点。传统的诊断方法主要针对单故障假设，在新的形势下，多故障假设和快速有效的流程是诊断软件的方向。本文主要利用现有工具和算法，在现有基础上提出了一种故障诊断软件的设计，具有单故障诊断和多故障诊断功能，相关功能完善，编辑功能较好。同时将这种设计实现了出来，可以进行实际使用。

本章接下来总结全文的主要工作，同时对未来的更深入工作提出研究的方向。

5.1 本文主要工作

本文用概略系统的语言说明了故障诊断的基本概念。包括组合电路的故障诊断，扫描链的故障诊断以及主流的一些算法。在参考流行 EDA 软件的基础上，通过对故障诊断工作的思考提出了一种具体的故障诊断软件的设计。在此设计的基础上，利用开发工具对软件进行了完整的实现。此软件能够实际进行完整的诊断流程，诊断算法适应性广。本文的工作主要有以下三点：

（1） 开发了一种友好的界面设计和操作流程。本文参考了 Cadence、Mentor Graphics 等公司的诊断软件，结合这些软件的优点和其他有用的特性制定了一种故障诊断软件的界面结构、交互方法。界面设计为主要利用菜单项来选择功能，利用对话框来输入信息。同时考虑了工程文件夹的展现方式，也把工程文件夹的展现与文件的打开相结合，加快了操作速度。操作流程使用顺序化方法。整体软件流程比较简单，不易出错。同时可以循环操作，重新对原有工作稍做调整就进入下一工作。并且可以随时在流程中改变源文件，临时利用其他输入文件进行功能操作，灵活性较高。

（2） 设计了一种诊断软件的模块化方案。主要模块列举如下：文本编辑模块、操作接口模块、输入解析模块、核心算法模块、结果展现模块。各个模块之间耦合性较低，利于替换。各个模块都集成了多种特性。操作接口主要负责交互，利用菜单、工具栏、列表框给用户提供对软件的全方位控制。文本编辑模块中，主要以多标签的方式实现文本的编辑浏览功能，辅以撤

消重做、查找替换等实用功能增强编辑的能力。输入解析模块利用各种方法提取输入文件中的有效信息，提高后续功能使用数据的便利性。核心算法模块包括电路信息查询、故障模拟、故障诊断等支持有效诊断工作的功能，考虑了通用性和高效性。结果展现模块，我们根据结果的不同特点，决定结果是直接显示在界面上、保存在日志中、生成特别文件还是图形显示。

（3） 对于软件的设计，我们用 C++代码实现了一个可用的软件。对每种模块，又具体划分了多个 C++的类进行细致的功能划分。界面利用 Qt 进行开发，考虑了美观和开发效率。输入文件解析利用了 Open Access 和 Lex/Yacc，以便支持更通用的文件。作为一个 IDE，我们着重实现了工程与文件管理功能，工程文件夹的操作实现了创建、打开、重命名等。文件管理包括打开、导入、新建等。同时注意了这些操作的相互联系。在代码结构的实现上，每个对话框的相关操作都封装到一个类中，每个主体功能用一个主界面的回调函数来实现。输入信息在软件中有专门设计的结构体来进行储存。

5.2 下一步工作方向

本文中实现的故障诊断软件具有上节提出的优点，同时也具有可以改进的方向。

（1） 核心功能优化：由于本软件是第一版软件的开发，着重点主要放在可使用性上，而不是高效性上，所以对于核心功能并没有进行特别的优化。在今后的研究中可以把优化重点放在改进功能所使用的算法上，降低算法的复杂度从而提高软件的高效性。比如可以加入作者所在研究组开发的其余各种较有特色，较高效的诊断算法[48-56]。

（2） 提升输入输出性能：在输入方面，未来还可以增加可支持的文件类型使软件兼容性更强。还可以在已支持的文件类型上开发新的解析模块。因为现有解析模块对文件的格式有一定要求，所以在支持文件的普遍性上还有提高的空间。在输出方面，可以在今后考虑更高端的显示方式，例如版图的图形显示。这些高端的显示方法能更好地让用户理解有效信息，实用价值很大。