分类号_____ 密级_____

UDC [注1] _____

学 位 论 文

基于子图同构的晶体管级电路门级模型抽取的研究

（题名和副题名）

徐玉婷

（作者姓名）

指导教师姓名 李绍荣 副教授

电子科技大学 成都

（职务、职称、学位、单位名称及地址）

申请学位级别 硕士 专业名称 电路与系统

论文提交日期 2007.4 论文答辩日期 2007.5

学位授予单位和日期 电子科技大学

答辩委员会主席_____

评阅人_____

2007 年 月 日

注 1：注明《国际十进分类法 UDC》的类号

独 创 性 声 明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。据我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得电子科技大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

签名：_____ 日期：　　年　　月　　日

关于论文使用授权的说明

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

（保密的学位论文在解密后应遵守此规定）

签名：_____　　导师签名：_____

日期：　　年　　月　　日

摘　要

随着 VLSI 的发展，电路日益增长的复杂性，使得自动从电路的晶体管级网表中抽取出门级模型变得愈发重要。在传统应用上，得到的门级模型可用于电路的功能仿真，与晶体管级仿真相比可节约大量的资源；而在形式化验证中，尤其是等价性检验中，门级模型的自动抽取也扮演着非常重要的角色。所以，必须研究开发一种抽取性能好，能处理大规模电路的门级模型抽取方案。基于子图同构的晶体管级电路门级模型抽取方法是目前较好的解决方案之一。该方法将电路转换为一个与之等价的图的结构，从而能明显降低时间复杂度，提高抽取能力。

本文结合具体的研究项目，对基于子图同构的晶体管级电路的门级模型抽取进行了系统的研究，并对 DECIDE 算法进行了改进和实现，取得了部分研究成果。主要内容为：

1．研究了晶体管级电路门级模型抽取的两大类主要方法，并对这两类方法进行了分析、比较。针对基于子图同构的抽取方法，进行了仔细的理论研究和分析，在此基础上，对经典的 DECIDE 算法进行了改进和实现。

2．根据项目实现的具体要求，针对实现中的技术难点进行了分析，提出了本系统的实现方案，并对该方案的实现进行了可行性分析，然后给出了系统实现框图。

3．根据系统要求，对系统中的每个功能模块进行了设计实现。主要包括输入处理模块、匹配模块和门级模型输出模块的设计与实现。并根据实际运算中的数据特性，对运算数据的存取进行了创新设计。

4．对晶体管级电路门级模型抽取软件进行了性能分析，包括时间复杂度和内存使用两方面，并进行了系统测试，对测试结果进行了分析。

5．总结了本文的主要工作，并对晶体管级电路门级模型抽取的后续工作进行了展望。

关键词：VLSI，晶体管级网表，门级模型，抽取，子图同构，DECIDE 算法

ABSTRACT

With the development of VLSI, the complexity of circuits makes automatally abstraction of a gate model from its transistor netlist more and more important. In traditional application, the gate model is used for functional simulation, which is much less time consuming than transistor level simulation; and in formal verification, especially in equivalence checking, the gate model plays a very important role. Therefore, a good way of abstraction should be researched. As we known, the method based on subgragh isomorphism is a promising way at present. It maps a circuit into a graph, which can reduce the time complexity and improve the abstraction ability.

In this thesis, the abstraction of a gate model from its transistor netlist is discussed systematically. Meanwhile, we make improvement and implementation on DECIDE algorithm and get some conclusions and achievements. The main content of the thesis is as follows:

1. Two kinds of main methods of abstraction are studied, analyzed and compared. And we carry on carefully the theoretical research and systematic analysis of the method based on subgraph isomorphism. In addition, we carry on improvement and implementation of DECIDE algorithm.

2. According to the project’s requirement, we analyze the technological difficult points while realizing it, and propose a new realization scheme. Meanwhile, we carry on feasibility analysis and provide systematic block diagram.

3. According to the system specification, the function modules of the system are analyzed and implemented, including input block, matching block and output block. Besides, the access and storage method of operands is redesigned in terms of data characteristic.

4. We carry on performance analysis of the abstraction, including the timing complexity and memory, then system testing is carried on and the experimental results are analyzed.

5. The general work of this thesis is summarized and directions for future investigation are outlined.

ABSTRACT

Keyword: VLSI, transistor netlist, gate model, abstraction, subgraph isomorphism, DECIDE algorithm

目录

第一章　绪论 1

1.1　课题背景 1

1.2　门级模型抽取的应用和发展现状 3

1.2.1 门级模型抽取的应用 3

1.2.2 国内外发展现状 3

1.3 本文主要工作 6

第二章　基于子图同构的子电路识别方法分析 7

2.1 子图同构 7

2.1.1 子图同构的定义 7

2.1.2 子图同构问题的复杂性 8

2.2 解决电路子图同构问题的主要算法 9

2.2.1 SubGemini 算法 9

2.2.2 DECIDE 算法 10

2.3 算法选择 10

2.4 基于 DECIDE 算法的子电路识别方法分析 11

2.4.1 电路图的建立 11

2.4.2 权值计算 12

2.4.3 子电路识别过程 13

2.4.4 对原算法的改进 15

2.4.4.1 改进一 15

2.4.4.2 改进二 16

第三章　基于 LINUX 的系统方案设计 18

3.1 系统平台的选择 18

3.1.1 LINUX 下 C 语言编程的特点 18

目录

3.1.2 LINUX 下 C 语言主要编程工具..... 18
3.2 系统方案总体设计..... 20
3.2.1 系统要求及分析..... 20
3.2.2 技术难点分析及解决方案..... 20
3.1.2.1 特殊类型电路的识别..... 21
3.1.2.2 图的存储..... 23
3.3 系统实现框图..... 24
第四章 模块设计及实现..... 26
4.1 功能模块分析..... 26
4.2 编译器模块的设计实现..... 26
4.2.1 编译器模块的功能分析..... 26
4.2.2 设计描述文件分析..... 26
4.2.3 系统内部数据结构..... 27
4.2.4 编译器模块实现..... 29
4.2.4.1 分析器的实现..... 29
4.2.4.2 检查器的实现..... 30
4.3 权值计算模块的设计实现..... 31
4.3.1 权值计算模块的功能分析..... 31
4.3.2 权值的存储..... 31
4.3.3 权值计算模块实现..... 32
4.4 同构判定模块的设计实现..... 33
4.4.1 同构判定模块功能分析..... 33
4.4.2 起始点和匹配候选集的确定..... 33
4.4.3 同构判定..... 34
4.4.4 同构判定模块实现..... 37
4.5 门级模型输出模块的设计实现..... 39
4.5.1 门级模型输出模块功能分析..... 39
4.5.2 门级输出格式确定..... 39

4.5.2.1 Verilog HDL 简介..... 39
4.5.2.2 Verilog HDL 基本语法..... 40
4.5.2.3 门级模型输出格式分析 41
4.5.4 门级模型输出模块实现 42
4.6 基于 LINUX 的系统实现..... 45
4.6.1 系统的自动化编译和链接 45
4.6.2 系统的运行和调试 45
4.7 系统工作流程..... 47
第五章 系统测试及实验结果..... 49
5.1 系统的编译和运行..... 49
5.2 系统资源使用情况分析..... 49
5.3 实验结果..... 50
第六章 全文总结..... 54
致谢 56
参考文献 57
攻硕期间取得的研究成果 60

第一章　绪论

1.1　课题背景

早在 1965 年，Gordon Moore 就曾经指出，硅片上晶体管的集成程度每 18 个月就会翻一番，这就是著名的摩尔定律[1]。几十年来，集成电路的发展不断印验了这一定律的正确性。随着当前半导体工艺水平步入深亚微米阶段，并正向 100nm 以下水平发展，芯片的复杂度和集成度空前提高，这使得芯片功能和逻辑出错的概率大大增加。为了保证芯片设计的准确性和 IP 的可重用性，验证[2]在设计中具有决定性的作用，因为设计中遗留的一点微小的错误将可能造成巨大的经济损失或者引起人员伤亡等灾难性后果[3]。例如 1994 年，奔腾处理器被发现在执行某个特定的浮点运算时出现错误，这种错误 27000 年才可能出现一次。对此，Intel 付出 4.75 亿美元的巨额代价回收有缺陷的奔腾处理器。1996 年 6 月 4 日，欧洲航天局研制的阿里亚娜五型火箭由于当一个很大的 64 位浮点数转换为 16 位带符号整数时出现异常在发射后不到 40 秒爆炸，造成直接经济损失五亿英镑。据统计，设计验证的时间占整个设计周期的 50%～80%，因此设计的正确性验证已经成为设计的瓶颈，被称为“验证危机”。

传统的验证方法[4]分为模拟和仿真。其中模拟验证仍然是目前主流的验证方法，它将激励信号施加于设计，进行计算并观察输出结果，判断该结果是否与预期一致。模拟验证的主要缺点是非完备性，即只能证明有错而不能证明无错。因此，模拟一般适用于在验证初期发现大量和明显的设计错误，而难以胜任设计中复杂和微妙的错误。模拟验证还严重依赖于测试向量的选取，而合理、充分地选取测试向量，达到高覆盖率是一个十分艰巨的课题。由于设计者不能预测所有错误的可能模式，所以尚未发现某个最好的覆盖率度量。即使选定了某个覆盖率度量，验证时间也是一个瓶颈。仿真验证在原理上与模拟验证类似，只是将模拟验证的三个主要部分即激励生成、监视器和覆盖率度量集成起来，构成测试基准(testbench)，用 FPGA 实现。仿真比模拟的验证速度快得多，其缺点是代价昂贵，灵活性差。

既然模拟和仿真验证都有如此大的局限性，自然就促使人们去研究更完善的验证方法，目前人们找到的新的验证方法是形式验证[5]。它用数学理论来证明所设计的系统满足系统的规范或具有所期望的性质。形式验证的主要优点是完备性，能够完全断定设计的正确性。其缺点是首先要对原始设计进行模型抽取，这对使用者有高的数学技能和经验上的要求。

形式验证包含等价性检验[5]、模型检验[6]和定理证明[7]，其中等价性检验使用最为广泛，而且已应用于大型复杂设计的验证。等价性检验主要用来验证两个设计模型的功能等价性，例如 RTL（register-transfer level）模型与门级（gate level）模型，RTL 模型与 RTL 模型，门级模型与门级模型。因此，等价性检验被广泛应用到设计流程中的不同阶段，如图 1-1 所示。

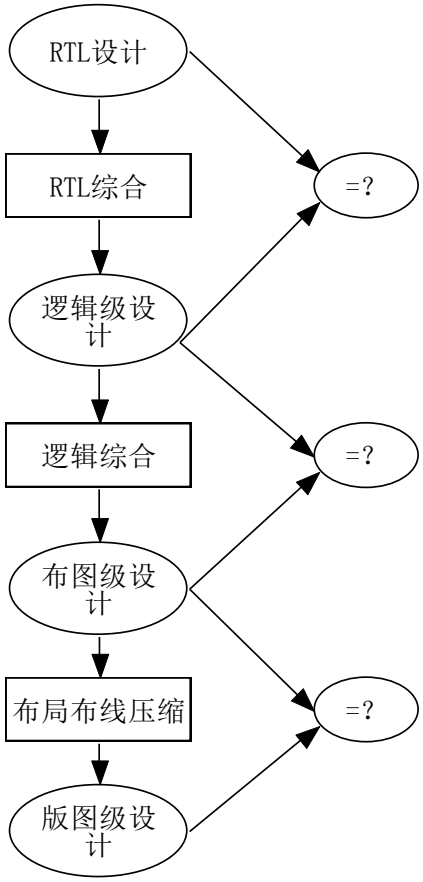


图 1-1 数字系统设计流程中的等价性问题

本文所讨论的晶体管级电路门级模型抽取，就是要从电路的晶体管级网表中抽取出它的门级模型，用于与参考的门级模型进行等价性比较。

1.2 门级模型抽取的应用和发展现状

1.2.1 门级模型抽取的应用

早期开关级仿真[8]是MOS技术中验证数字电路的一种有效方法。因为从电路的晶体管级描述出发，开关级的仿真器可以处理各种不同类型的电路设计，而且还能发现双向信号流，动态存储电荷及不同电阻的多信号源驱动等这些因素对电路的影响。然而随着集成电路的不断发展，电路的规模越来越大，在一个芯片上已经可以集成数百万个甚至数千万个晶体管构成的电路。而电路分析软件可以仿真的电路受到计算时间、计算机内存和计算收敛性的限制，在晶体管级完成这么大规模电路的仿真是不现实的。因此需要自动地从晶体管电路中产生一个门级模型，用于对电路进行功能仿真[9] [10]，与晶体管级仿真相比可节约大量的资源。

后来随着 VLSI 的进一步发展，电路日益增长的复杂性给计算机辅助设计 CAD 工具带来巨大挑战的同时，电路的层次化描述也愈发重要。一个数字电路可以通过多种不同的抽象层次来描述。例如晶体管级是通过晶体管的性质和它们之间的连接关系来描述电路，门级则从逻辑门组合及连接角度来描述整个电路。这也使得自动地从电路的晶体管级网表得到一个较高层次的描述如门级描述在 VLSI 设计中起着非常重要的作用。抽取得到的门级模型可直接套用固定型故障（Stuck-at-fault）模型[1]，对大规模集成电路做测试向量生成[11]和可测性分析[12]；可通过 LVS（版图与线路图对比工具）直接与晶体管电路图进行比对。

此外，采用形式化验证，尤其是等价性检验，门级模型的自动抽取也扮演着非常重要的角色。等价性检验的主要目的是在一个设计经过变换之后，穷尽地检验变换前后功能的一致性，即证明设计的变换没有产生功能的变换，比如逻辑最小化后与原电路等价，对电路结构作局部修改后需证明与原电路等价，从较高级别的描述综合为较低级别的描述，也需要证明其所实现的电路与原来的描述等价等。所以，将从晶体管级电路中抽取出来的门级模型与参考门级模型二者进行等价性检验是检验电路晶体管级设计正确性的一个重要方法。因此，研究晶体管级电路门级模型抽取技术，对于超大规模集成电路的验证具有重大意义。

1.2.2 国内外发展现状

早在 1983 年，学术界就出现了用于电路抽取的专门算法。但是早期的算法[13]-[16]大多依赖于特殊的电路工艺或类型，在应用上有很大的局限性。后来随着

VLSI 的发展，用于电路抽取的算法也越来越多，根据其基本思想主要分为两大类：规则识别[17]和模式匹配[18]。

规则识别采用基于规则的识别技术从沟道相连的 MOS 管集合[16] [19]中提取出逻辑门。它首先根据一些普遍的规则对晶体管级网表进行划分，得到的每一个部分都表示一个已抽取的门，然后这些门的具体功能描述可以通过推理得到。这类算法对于结构规则的电路，例如 P 和 N 互补的 CMOS 静态电路，识别速度非常快。但是它不能处理结构不规则的模块，如 DFF，锁存器和模拟模块等，因为这些模块的识别规则很难事先定义出来。

模式匹配是一种基于子图同构[20]的抽取算法，它的研究始于 1994 年。这类算法的主要思想是将电路转换为一个与之等价的图的结构，其中晶体管等器件表示为节点，内部的连接表示为边，然后采用子图同构技术判定两个图之间节点和边的一对一的关系, 从而在主电路中识别出与事先定义的子电路[21]结构相同的部分，并用定义好的相对应的门级模型替换识别出的子电路。但是子图同构问题实际上是一个 NP 完全(Non-deterministic Polynomial time complete)问题[22]，换句话说，在最坏情况下，判定两个图之间子图同构关系所需的时间与这些图的节点数量成指数增长关系。然而由于这类方法不依赖于电路的制造工艺与类型，所以相对于规则识别，它的应用依然更为广泛。

在学术界对抽取算法进行不断研究的同时，工业界也陆续出现了一系列用于门级抽取的 EDA 工具。例如 Avertec 公司开发的功能抽取器 YAGLE[23]可以自动地从电路的晶体管级描述中得到它的行为描述。Verplex 公司则将它的逻辑晶体管提取工具 Tuxedo LTX[24]嵌入在它的另一形式化验证工具 Tuxedo LEC 中，可以对几百万门的 IC 设计进行从 RTL、门级到晶体管网表的等价性比较。Cadence 公司的 Encounter Conformal Custom[25]则能自动地从晶体管电路中提取出门级模型，并将其与原始的 RTL 进行功能等价性比较。

当前芯片设计日趋复杂，设计验证工作越发繁重，花费在验证环节的费用要占经费投入的 70%[3]，集成电路验证技术已经成为了制约集成电路产业发展的瓶颈。但是在国内，这种情况尚未引起足够重视，普遍使用的验证技术还停留在传统的验证手段上，对新兴的各种验证技术了解不够，更谈不上应用到实际设计过程当中，甚至很多人对于形式化验证还感到陌生，EDA 设计过程中结合各种验证方法的技术和相应平台工具基本上还是空白。因此，具有自主知识产权的集成电路验证技术和平台的研发是我国集成电路产业发展的迫切需求，也是我国集成电路产业做强做大的必由之路。同时，我国集成电路产业的发展必须在核心技术的

掌握上重点突破，要建立“中国的集成电路芯片产业”，而不能仅仅停留在“集成电路芯片产业在中国”。从国家的安全考虑，不能没有自主的核心芯片技术和知识产权。网络与信息安全的基础是产业，要有自己的高水平的集成电路芯片和相关的软件产品，必须建立独立自主的信息安全产业，掌握核心技术才能不受制于人，才能有真正的国家安全。

2006 年，成都集成电路数字形式验证工程技术研究中心联合成都巨微集成电路验证技术工程有限公司开发了具有我国自主知识产权的集成电路芯片验证平台“巨微”等价性验证系统（如图 1-2 所示），填补了我国在集成电路验证领域的空白，为满足集成电路产业发展与国家信息安全的需要做出了贡献。

在“巨微”等价性验证系统中，应用层核心算法是研发的重点，除了包括输入文件分析器（Parser）、综合器（Synthesizer）、预处理器（pre-processor）、核心算法（匹配器、验证器、调试器和模拟器），还包含了抽取器（Abstractor）的研发，它的基本功能就是从物理版图中抽取门级模型描述。

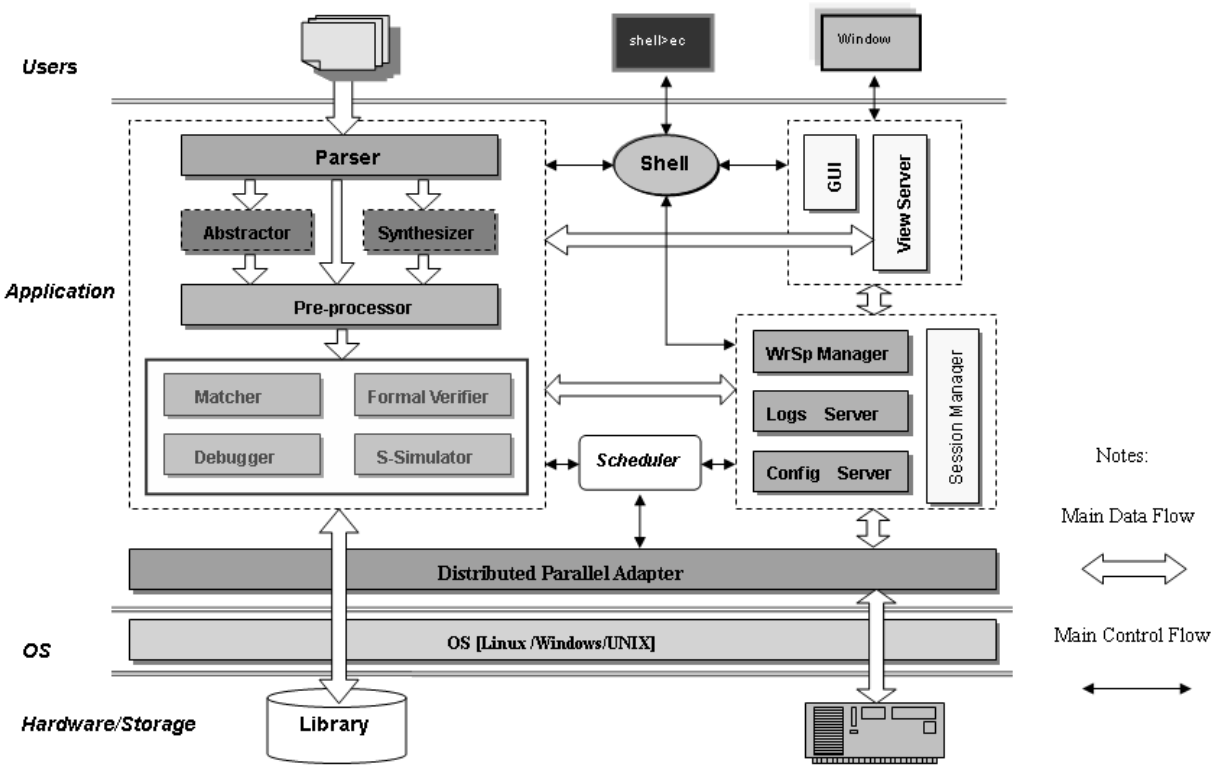


图 1-2 “巨微”等价性系统架构图

1.3 本文主要工作

本文所描述的系统，是作为“巨微”等价性检验系统中的前端处理模块，采用基于子图同构的晶体管级电路门级模型抽取算法，主要完成对电路晶体管级网表的处理、门级模型的抽取及输出，最后输出将用于等价性检验系统。实现功能的简单框图如图 1-3 所示。

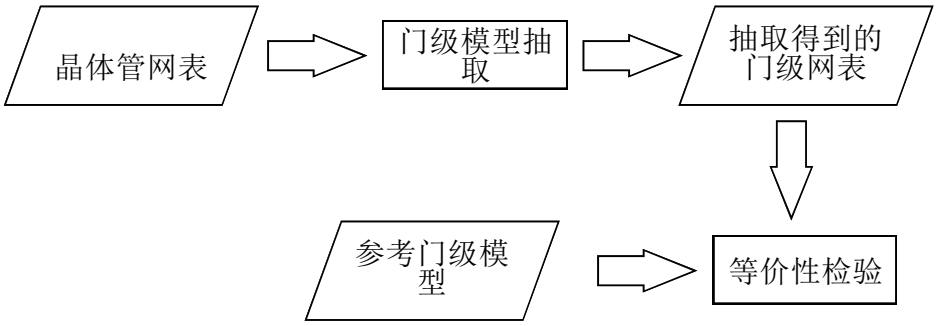


图 1-3 等价性检验系统框图

结合上述实际科研任务，本文对基于子图同构的电路晶体管级门级模型抽取算法进行了系统的研究，在实现方案上进行了大胆的探索，取得了部分研究成果，主要内容安排如下：

第一章首先论述了晶体管级电路门级模型抽取的由来、应用和国内外发展现状，然后介绍了本课题的背景，最后简要归纳了本文的主要内容。

第二章主要分析、介绍基于子图同构的门级模型抽取方法，这是后续实现工作的基础。先对图论中的子图同构理论进行了详细论述，后结合电路的特性，分析了电路图的子图同构判定问题的复杂性，并对目前使用最为广泛的两种子图同构算法进行了分析比较，确定了本课题所使用的核心算法。最后对算法进行了实现和改进。

第三章是根据实际科研任务，对基于 LINUX 平台上的电路晶体管级门级模型抽取器的实现方案进行分析和总体设计。

第四章是在前一章系统分析的基础上，对系统框图的各个模块进行了详细的设计实现。首先，分别对输入模块、匹配模块、门级模型输出模块进行了设计实现。最后，完成了整个系统在 LINUX 平台上的编译，生成了可执行程序。

第五章对系统进行测试，给出了实验结果和结论。

第六章总结了本文的主要工作，对基于模式匹配的电路晶体管级门级模型抽取的后续研究工作进行了展望。

第二章　基于子图同构的子电路识别方法分析

基于子图同构的子电路识别算法是整个门级模型抽取过程的关键。它的思想基础来源于任何一个电路都可以转换为与之等价的图的形式，如图 2-1 所示。然后采用子图同构技术，从主电路图中识别出与事先定义的子电路结构相同的单元。由于每一个子电路都与一个门级描述相对应，从而得到由这些子电路的门级描述组成的主电路门级模型。

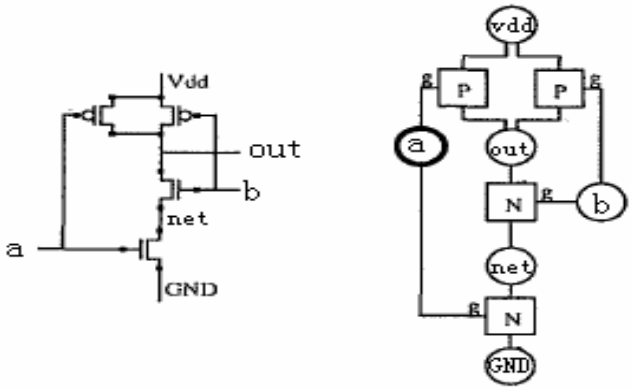


图 2-1 二输入与门的晶体管级电路与其对应的图

2.1 子图同构

2.1.1 子图同构的定义

子图同构是图论中一个重要概念。为了清楚子图同构的概念，下面先给出几个关于图的基本定义[26]。

定义 1 图$G=\langle V,E\rangle$是有序二元组，其中$V(G)$是有限非空集，V 中元素称为G的顶点或节点，V 称为G 的顶点集，$E(G)$ 是$V(G)$中诸顶点之间边或弧的集合，E称为G 的边集。

定义 2 设$G=\langle V,E\rangle$，$G'=\langle V',E'\rangle$ 是两个图，若$V'\subseteq V$，且$E'\subseteq E$，则称G'是G 的子图，记作$G'\subseteq G$。

定义 3 设两个无向图$G_1=(V_1,E_1)$和$G_2=(V_2,E_2)$，若存在双射函数$\varphi:V_1\to V_2$，使得对于任意的$e=(V_i,V_j)\in E_1$，当且仅当$e'=\left(\varphi(V_i),\varphi\left(V_j\right)\right)\in E_2$，则

称G_1与G_2同构，记作$G_1 \cong G_2$。例如，图 2-2 中 A 和 B 是同构的两个图。

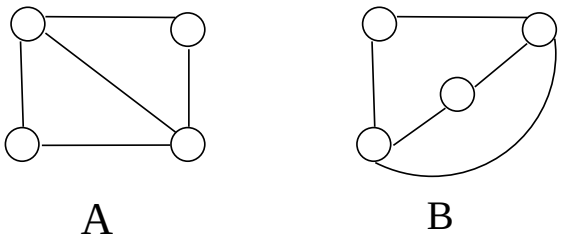


图 2-2 图同构的示例

根据上面三条定义，我们得到了关于子图同构的基本定义。

定义 4 $G_1=(V_1,E_1)$与$G_2=(V_2,E_2)$中的一个子图S_2同构，记作$G_1 \cong S_2 \subseteq G_2$，是指如果存在一个映射函数$\varphi:V_1 \to V_2$，使得$(V_i,V_j) \in E_1$，那么$\left(\varphi(V_i),\varphi\left(V_j\right)\right) \in E_2$。例如，图 2-3 是子图同构的示例。S 是 G 的一个子图，它包含了 G 中顶点集的一个子集，而且 S 中顶点的连接方式与 G 中相同，所以称 S 是 G 的一个同构的子图。

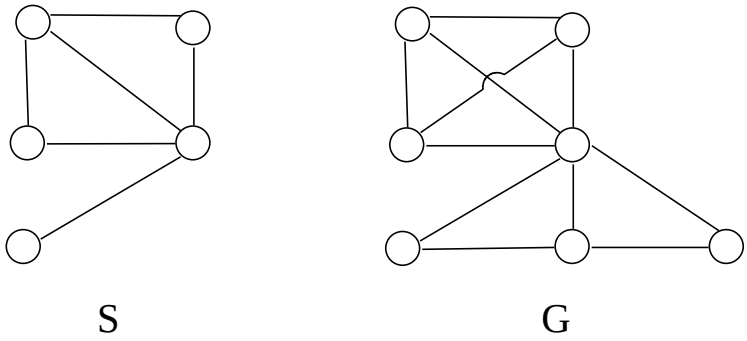


图 2-3 子图同构的示例

2.1.2 子图同构问题的复杂性

在理想情况下，问题都可以在多项式时间（也称为 P 时间）内解决。这意味着对于解决一个含有n项的问题，存在c， z， g几个常量，算法 T 所需时间受以下公式[27]的限制：

$$T \le c+ \ g^{z}r \qquad (2\text{-}1)$$

其中常量z是我们真正感兴趣的，它通常被叫做“增长因子”。它表示当n增加时算法预期增多的额外时间的数量。例如当z=1，n加倍时，算法所需的时间也加倍。

但是子图同构被证明是一个 NP 完全问题，即在最坏情况下，判定两个图子图

同构关系所需时间与图中所包含的节点数量成指数增长关系。换句话说，最坏情况下解决子图同构问题的时间复杂度为$o\left(e^n\right)$，其中n为规模较大的图中的节点个数。

虽然子图同构问题具有先天的复杂性，但是由于电路网表本身具有的一些特性[27]，使得仍然存在一些算法能将电路图中的子图同构问题简化，使得这一问题能在线性时间（$o(n)$）内解决。这些特性主要包括以下几点：

1、实际情况中，子电路所包含的结构会以相同的结构出现在包含与之匹配的电路结构中。这种局部同构的不变性可大大简化查找的复杂度。

2、常见的电路器件，如晶体管、电阻、电容等都有固定的端口，且端口的数量一般不会超过 4 个，这大大降低了电路图的复杂度。

3、在电路网表中，器件及器件的端口通常会有各自的标记，如“nfet”、“pfet”、“registor”、“gate”、“source”、“drain”等，从而降低了标记过程的难度。

4、在大多数网表中，一般网络的扇出数平均很小，而且一般不会随电路网表规模的增大而增加。所以 VLSI 图的复杂性小于一般的图。

正是由于上述 4 点特殊性，使得子图同构算法在处理电路图的子图同构问题比其它类型的图性能相对较好。

2.2 解决电路子图同构问题的主要算法

鉴于子图同构问题的复杂性和电路图的特殊性，很多研究者都对电路图的子图同构问题提出了各种各样的解决方法[28]-[37]，希望这一问题能在一个多项式时间里有效地解决。其中有代表性的算法有 SubGemini[28] 和 DECIDE[32]算法两种。

2.2.1 SubGemini 算法

SubGemini 的提出者 Miles Ohlrich 等人是第一批研究采用基于子图同构的方法来解决子电路识别问题的研究者之一，而且该算法已经成功应用于同名的商业软件 SubGemini 中。由于该算法全面的实验数据和快速的运行速度，它也成为了最常使用和研究的算法之一。整个算法的执行过程分为两个阶段：

第一阶段，SubGemini 在主电路图中确认出所有可能与子电路同构的位置。这是通过一个划分算法来完成的。首先需要在子电路图中找到一个关键节点（key vertex）K，然后在主电路图中确定所有可能与 K 匹配的节点，这些节点共同组成一个集合，称为候选集合（candidate set）。这样做的目的是为了减少需要检查的节

点数目。

第二阶段，在主电路图中，按照事先确定的候选集合，逐个判定包含该节点的示例是否与子电路同构。

实验结果表明，对于大型的 CMOS 电路，该算法判定子图同构所需时间与子电路中器件的数目成线性关系。但是由于算法中的重新标记算法依赖于这样一个假设，即同一个子电路中的外部线网与其它外部线网之间不会发生短路。因此，该算法不能识别短路电路中的子电路。这使得该算法在处理电路的类型上存在一定的局限性。

2.2.2 DECIDE 算法

2001 年台湾国立清华大学的 Wei-Hsin Chang 等人提出了一种采用递归方法完成识别过程的算法，称为 DECIDE 算法。该算法通过函数的迭代计算出电路图中每一个节点的权值，然后经过权值比较，确定候选节点集合，简化接下来的识别过程。

与 SubGemini 算法相比，该算法在确定候选节点集合阶段的操作要简单的多。而且在判定同构的过程中，DECIDE 算法由于采用了递归策略，所以具有速度快的优势。通过实验结果分析，该算法在 16.6 秒内可以处理带有 10 万个左右晶体管的电路。

2.3 算法选择

在上一小节中，我们分析了解决电路图中子图同构问题的两种主要算法 SubGemini 和 DECIDE。SubGemini 算法由于它全面的实验数据和快速的运行速度，已经成为了最常使用和研究的算法之一，而且在产业界也有相对成熟的应用。除了成功开发了商用工具 SubGemini 外，在 Avertec 公司开发的功能抽取器 YAGLE 中，SubGemini 算法与基于规则的识别方法相结合，用于处理带时序模块和模拟模块的电路。但是由于 SubGemini 算法在第一阶段采用的划分的预处理方式会消耗大量的时间，而且划分算法的选取直接影响整个识别算法的效率，因此在本课题中对子电路的识别将以比较新颖的 DECIDE 算法为基础。这不仅是因为 DECIDE 算法中确定候选节点集合的方法比 SubGemini 简单，还因为 DECIDE 算法目前尚未应用在任何理论的或商业的门级模型抽取器中。

2.4 基于 DECIDE 算法的子电路识别方法分析

根据文献[32]对 DECIDE 算法的描述，在其基本思想的基础上，本课题对其进行了一些启发式的改进，并用 C 语言实现了该算法。实现算法的最终基本流程如图 2-4 所示。根据流程图，算法的执行分为三个主要阶段：电路图的建立、权值计算和子电路识别。

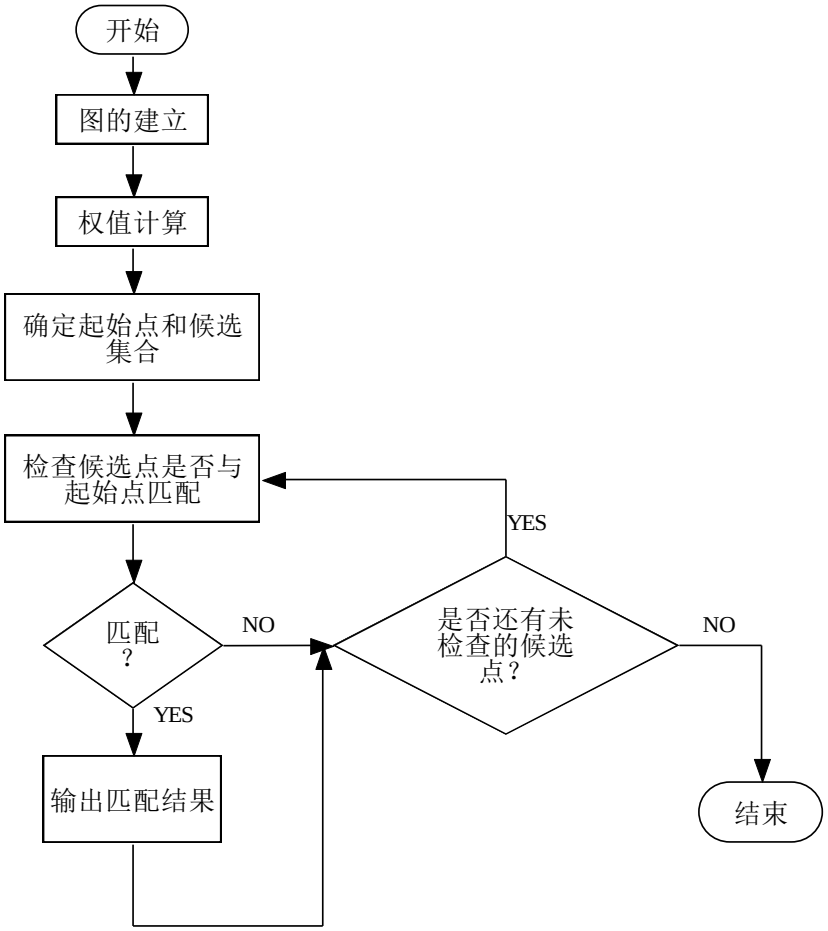


图 2-4 DECIDE 算法基本流程

2.4.1 电路图的建立

由于本课题采用的子电路识别算法是基于子图同构的，所以首先需要将电路网表转换成与之等价的图的形式。

任何电路都可以表示成图的形式$G=(V,E)$，其中V是节点的集合，E是边的集合。根据性质，图中的节点V可分为两大类：器件节点（device node）和端节点（terminal node）。器件节点用于表示电路中的各种器件，如 PMOS 和 NMOS 等。

端节点又分为内部节点和外部节点。通常用方框表示器件节点，圆圈表示端节点。每个节点都有与之相邻的节点，称为该节点的邻节点。如图 2-1 所示，P 和 N 是器件节点，a、b 和 out 是外部节点，net 是内部节点。节点 out 的邻节点为两个 P 节点和一个 N 节点。

2.4.2 权值计算

对图中节点进行权值计算是简化识别过程的前提。权值计算是根据图中节点之间的连接关系，通过权值函数的迭代来完成的。节点的权值实际上是该点在图中连接信息的量化表示。关于权值的计算公式，本文在文献[28] [30] [31]基础上总结得到的。端节点计算如式（2-2）所示：

$$W_Z^n = \begin{cases} C_{PMOS} \times C_P + C_{NMOS} \times C_N, n = 0 \\ C_P \times \sum_{i=0}^{C_{PMOS}} W_{P_i}{}^n + C_N \times \sum_{i=0}^{C_{NMOS}} W_{N_i}{}^n, \text{其他} \end{cases} \tag{2-2}$$

其中 n 是指权值计算的迭代次数，C_{PMOS} 和 C_{NMOS} 分别代表电路图中与该节点相连的 PMOS 和 NMOS 器件的个数，C_P 和 C_N 是分别计算 PMOS 和 NMOS 权值的系数，W_P 和 W_N 分别为该端节点连接的 PMOS 和 NMOS 器件节点的权值。器件节点权值为该节点所有邻节点权值之和，W_P 的计算公式如式（2-3）所示。

$$W_P{}^n = \sum_{i=1}^{3} W_{Z_i}{}^{n-1}, n \geq 1 \tag{2-3}$$

由于 PMOS 器件一般有源极、漏极和栅极三个信号连接端口，因此它的权值为所连的三个端节点权值之和。NMOS 器件节点的权值 W_N 计算公式与 W_P 相同。

为了方便匹配的进行，我们将除主要输出节点外的其它所有外部端节点的权值设定为 0。权值计算是一个迭代的过程，它的具体算法如下：

STEP1：初始化所有端节点的权值，$C_{PMOS} \times C_P + C_{NMOS} \times C_N$；

STEP2：根据已计算的端节点权值，得到所有器件节点的权值；

STEP3：根据已经计算的器件节点权值，按公式再次计算端节点的权值；

STEP4：重复STEP2和STEP3，直到达到预先设定的迭代次数。

从算法中可知，对节点每进行一次迭代的权值计算，相当于将该节点的更外一层的信息反传过来给它。但是迭代次数过大，权值计算阶段耗费的时间就过多；迭代次数过小，节点的信息收集不完全，达不到权值计算的目的。由于电路包含的基本门结构中晶体管串并联的个数一般不会超过4对，因此一般将迭代次数设为

3～4即可将输出节点附近的信息收集完全。

图2-5表示反相器中各个节点权值的计算过程。假设迭代次数n=2，C_P=3,C_N=2。先对端节点即out节点的权值进行初始化。由于该节点只连接了一个P节点和一个N节点，因此它的权值为1*3+1*2=5。开始第一次迭代，根据上一步得到的端节点的权值，计算P节点和N节点的权值，观察这两个节点的连接情况，发现只有连接的out节点有权值，因此这两个器件节点的权值都为5，out节点处的权值变为5*3+5*2=25。至此，第一次迭代完成。然后按照之前的步骤,继续第二次迭代计算。最后得到的out节点权值为125，P和N节点权值为25。

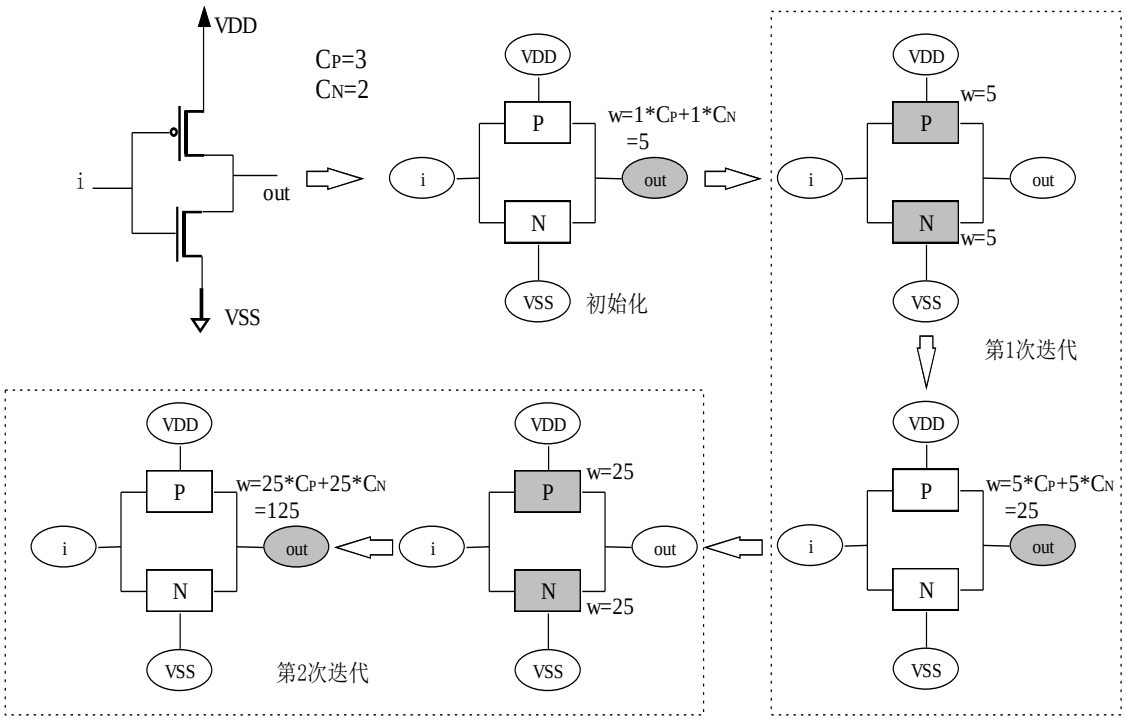


图2-5 非门结构的权值计算过程

2.4.3 子电路识别过程

识别子电路成功的关键在于找到互相匹配的节点，当然首先需要确定整个识别过程的起始点。我们将子电路的主要输出节点作为匹配的起始点。这是因为每一个子电路都有其主要输出节点，这一特点具有普遍性。这种启发式的策略对任何类型的子电路都适用，而且利于匹配递归终止条件的最终确立。

由于权值是电路图各个节点连接信息的量化，而且主电路规模通常比子电路大，所以主电路中与子电路起始点相匹配的节点权值不可能小于起始点本身的权值，而且主电路中的器件节点也不可能与起始节点（端节点）匹配，所以根据确

定的起始点的权值，将主电路中所有大于或等于该权值的端节点组成一个匹配候选集。通过这样启发式的选取策略，过滤了大部分不可能成功匹配的节点，避免了大量无效的匹配过程，从而可大大提高匹配的效率。

以图 2-6 为例，在子电路（反相器）中，选取主要输出节点（节点 2）作为匹配的起始点，那么在主电路中，只有权值大于或等于节点 2 权值的点才能作为匹配的候选点参与匹配。通过观察，主电路中节点 1 的权值为 272，节点 3 的权值为 125，节点 4 的权值为 640，均不小于起始节点的权值 125。因此，候选集中最终只有三个节点:节点 1、3 和 4。

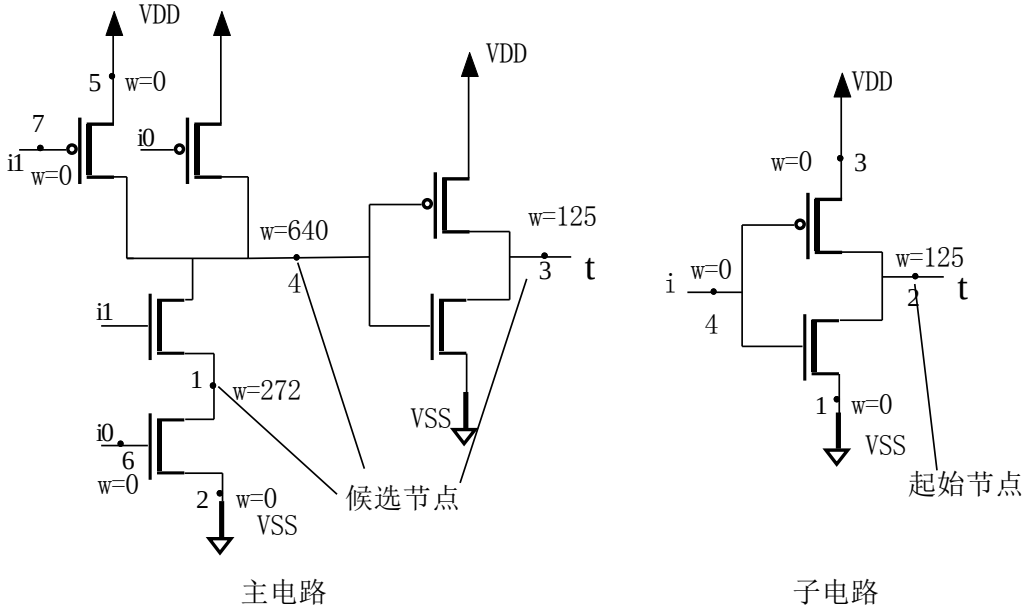


图2-6 主电路与子电路

在匹配的过程中，由于一个复杂结构中可能同时包含多个相同的子结构，所以候选集中的每一个节点都要与起始节点做比较。确认一个候选节点与起始节点是否匹配，就是要确认与它们相邻的对应节点是否匹配。这是一个递归的算法。从当前节点遍历到下一个节点(邻节点)必须满足以下两个条件:（1）当前节点的类型相同;（2）当前节点的C_{PMOS}和C_{NMOS}相同。

但是递归不可能无尽地进行下去，所以算法又确定了三条终止条件:（1）遇到VDD/GND;（2）遇到主要输入节点;（3）该节点所有的邻节点都已经匹配。当任何一条终止条件得到满足时，递归终止，返回匹配的结果。

以从图 2-6 所示的主电路中识别出子电路为例。假设先选择候选节点集中的节点 3 与起始点进行匹配，由于这两个节点类型相同，且相邻节点的个数也相同，

于是再进一步比较它们的邻节点 5 和 3。因为它们连接的都是电源，所以节点 5 和 3 是匹配的；同理，节点 3 的另一邻节点 2 和对应的起始点的邻节点 1 也是一对匹配节点。所以，主电路中的节点 3 与起始点匹配，即在主电路中识别到了一个与子电路同构的结构。以相同的方式，将候选节点集中的节点 1 和 4 也依次与起始点进行匹配，结果都不匹配。识别结束，最终在主电路中找到了一个与子电路同构的结构。

2.4.4 对原算法的改进

根据 DECIDE 算法的主要思想和策略，在 2.4 节中介绍和分析了本课题的核心——基于 DECIDE 算法的子电路识别方法，在实现过程中对原算法进行了如下几点的改进：

改进一：在权值计算过程中，将除主要输出节点外的所有外部节点的权值设定为 0。

改进二：在子电路识别过程中，将匹配的起始点明确地定为子电路的主要输出节点。

这两点启发式的改进在一定程度上简化了权值计算和子电路识别的过程。下面将详细介绍。

2.4.4.1 改进一

在权值计算过程中，将除主要输出节点外的其它所有外部节点的权值设为 0，主要是基于以下几个方面的考虑：

1、降低权值计算的复杂度和运算量。因为每个电路都有“VDD”和“VSS”这类固定的外部节点，而这些节点对电路图的同构判定不会产生什么影响。对于输入节点这一类节点而言，由于它们通常连接在晶体管的栅极上，但是晶体管电路主要是依靠晶体管的源极和漏极相连组成的，因此栅极上的节点并不会影响电路图内部节点的连接情况。所以将主要输出节点外的其它外部节点的权值设为 0 并不会影响权值计算本身的意义。而且，将递归运算中的一些无意义的值固定地设为 0 也降低了权值计算的复杂度和运算量。

2、简化判定终止条件。由于同构的判定过程是递归的，递归过程的结束需要终止条件。当节点的遍历进行到“VDD”或“VSS”时，就把它视为判定过程的终止条件之一。于是确认是否遍历到“VDD”或“VSS”，则可以通过检查节点的权值是否为 0 来完成。

2.4.4.2 改进二

在 DECIDE 原算法中，没有明确指出如何选择同构判定的起始点。本文在这方面做了一些探索。首先根据实际的电路和节点权值计算情况，分别制定了两种不同的起始点选取策略供选择。

策略一：选择权值最大的点作为判定开始的起始点。依据此策略实现的算法虽然侯选节点集合中节点的数目比较少，但在实际运行中会出现这样的问题，即并不是所有电路中权值最大的点都是该电路的主要输出节点。这使得需要判定输出节点的匹配性。例如在图 2-7 所示的电路中，通过权值计算，发现权值最大的点是节点 6，而电路的主要输出点是节点 2。如果从节点 6 出发，就需要判断节点 2 的匹配性。而在权值计算中，我们将除输出节点外的所有外部节点的权值都设定为零，就是为了在判定过程中当遍历到这些节点时，判定终止。但如果需要判定输出节点这一类外部节点的话，递归的判定过程的终止条件就不好设定。

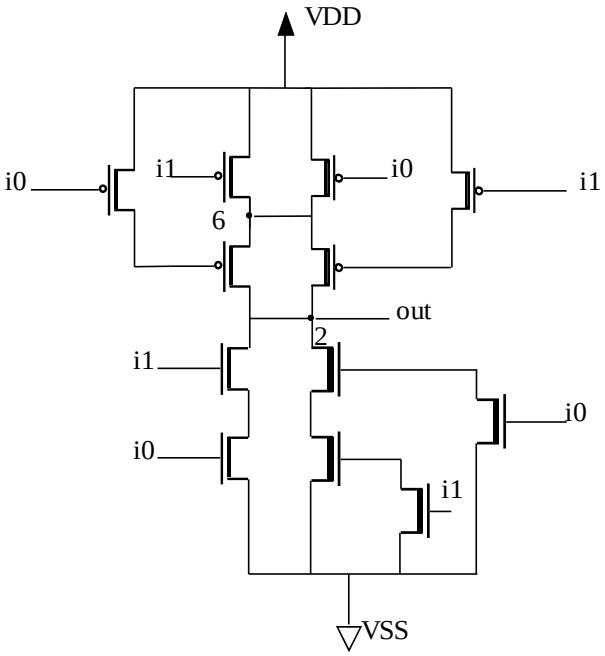


图2-7 权值最大的点不是主要输出点的例子

策略二：为简化判定难度，将电路图的主要输出点作为匹配的起始点。这样就不用考虑判定过程中遇到输出节点的问题。这样做的缺点就是侯选节点集合中节点的数目会比较多，尤其是当电路的主要输出节点不是权值最大的点时。但是对于大部分结构简单的基本电路，它们的主要输出点和权值最大的点是重合的。

综合上述两种策略优缺点，本文最终采用策略二的方法，将电路图的主要输

出点作为匹配的起始点。

总之，本课题的核心部分子电路识别算法以 DECIDE 算法的思想为基础，对其进行了两点启发式的改进，一方面降低了权值计算的复杂度，减少了计算所需的时间和内存，另一方面在一定程度上简化了识别的过程。关于这一部分的实验结果，将在第五章与整个系统的实验结果一起给出。

第三章　基于 LINUX 的系统方案设计

3.1 系统平台的选择

本系统选择在 LINUX 系统上开发和实现，不仅是为了满足成都集成电路数字形式验证工程技术研究中心开发集成电路芯片验证平台“巨微”等价性验证系统的需求，也是由于 LINUX 下 C 程序开发有其独特的优势。

3.1.1 LINUX 下 C 语言编程的特点

Linux[38]是一个奇迹，是由 UNIX 发展而来的，最初由一个芬兰大学生开发维护，现在已经成为最为流行的免费操作系统。Linux 的独特之处在于它的建立不受任何商品化软件的版权制约，全世界都能免费、自由地使用。世界各地有几十万自愿者为这个充满魅力的系统贡献着自己的才能，从初学者到计算机专业人士，还有经验丰富的黑客们，一起不断地改进和维护着这个系统。许多大学与研究机构、公司及家用 PC 都在使用 Linux。Linux 年轻而富有朝气，它从诞生到现在不过 15 年时间，但已经在市场上确立了自己的地位和广泛的影响。

而 C 语言[39]则是国际上广泛使用的计算机高级语言。C 语言最初用于描述和实现 UNIX 系统，后来逐渐被广大程序员所接受，成为最受欢迎的编程语言。在其后的发展过程中，C 语言不断吸收计算机方面新的成果，使这个古老的语言又焕发出新的魅力。作为 Linux 系统的开发语言，C 语言在 Linux 编程开发中扮演着重要的角色。

由于 Linux 作为一个操作系统，它的重要功能就是要支持用户编程；C 语言作为当前使用最广泛的编程语言，具有多平台、移植性好的特点，因此它们很快形成了完美的结合，为用户提供了一个强大的编程环境。

3.1.2 LINUX 下 C 语言主要编程工具

LINUX 下 C 编程[40]通常包括三个步骤：源代码输入，程序编译、链接、运行以及调试。Linux 系统中为 C 编程提供了许多编程工具。

1、编辑工具：在 Linux 下编程，使用的是类似于 EDIT 的工具---经典的 vi

来编辑源程序。当然，还有更高档一些的，如 joe、emacs 等。总之，编辑程序与编译工作是分开的。

2、编译工具：随 Slackware Linux 发行的 GNU C 编译器(GCC)是一个全功能的 ANSI C 兼容编译器。在使用 GCC 时，通常后跟一些选项和文件名来使用 GCC 编译器。gcc 命令的基本用法如下:

gcc [options] [filenames]

命令行选项指定的操作将在命令行上每个给出的文件上执行。GCC 有超过 100 个的编译选项可用。这些选项中的许多你可能永远都不会用到，但一些主要的选项将会频繁用到。

当使用 GCC 编译 C 代码时，GCC 通常会试着用最少的时间完成编译并且使编译后的代码易于调试。易于调试意味着编译后的代码与源代码有同样的执行次序，编译后的代码没有经过优化。有很多选项可用于告诉 GCC 在耗费更多编译时间和牺牲易调试性的基础上产生更小更快的可执行文件。这些选项中最典型的是 -O 和 -O2 选项。

-O 选项告诉 GCC 对源代码进行基本优化。这些优化在大多数情况下都会使程序执行的更快。

-O2 选项告诉 GCC 产生尽可能小和尽可能快的代码。-O2 选项将使编译的速度比使用 -O 时慢。但通常产生的代码执行速度会更快。

3、调试工具：Linux 包含了一个叫 gdb 的 GNU 调试程序。gdb 是一个用来调试 C 和 C++程序的强力调试器。它使你能在程序运行时观察程序的内部结构和内存的使用情况。以下是 gdb 所提供的一些功能:

（1）能监视程序中变量的值；

（2）能设置断点以使程序在指定的代码行上停止执行；

（3）能逐行执行代码。

gdb 支持很多的命令使你能实现不同的功能。这些命令从简单的文件装入到允许你检查所调用的堆栈内容的复杂命令，表 3-1 列出了你在用 gdb 调试时会用到的一些命令。

表 3-1　基本 gdb 命令描述

命令	功能
file	装入想要调试的可执行文件
kill	终止正在调试的程序
list	列出产生执行文件的源代码的一部分
next	执行一行源代码但不进入函数内部
step	执行一行源代码而且进入函数内部
run	执行当前被调试的程序
quit	终止 gdb
watch	使得可以监视一个变量的值而不管它何时被改变
break	在代码里设置断点，这将使程序执行到这里时被挂起
make	使得能不退出 gdb 就可以重新产生可执行文件
shell	使得能不离开 gdb 就执行 UNIX shell 命令

3.2 系统方案总体设计

3.2.1 系统要求及分析

根据要求，本系统实现的主要功能有如下几点：

1、可以从一个 SPICE[41]或 al 格式（Alliance[42]内部网表）的主电路中识别出它所包含的所有子电路，这个电路可能是展开（flattened）的，也可能是有层次（hierarchy）的；

2、最终输出一个用硬件描述语言[43] [44]描述的主电路门级结构的文件；

3、能处理较多类型的电路结构，而且整个处理过程耗时相对较短。

在系统实现中，本文将采用第二章里介绍的基于子图同构的子电路识别方法，由于该方法不采用对主电路进行切割的方式，可以节省许多预处理时间，而且通过权值函数，在判定同构之前就过滤掉那些不可能匹配的节点，避免了许多无效的判定过程。这些方法的使用都在一定程度上减少了整个系统的运行时间。

3.2.2 技术难点分析及解决方案

通过对系统功能的分析可以发现，在系统实现中主要有两个技术难点：特殊

类型电路的识别和电路图节点的存储。下面，本文将具体分析每个难点，并给出相应的解决方案。

3.1.2.1 特殊类型电路的识别

由于本系统需要尽量处理较多类型的电路，而系统中判定电路图子图同构的递归程序在一些特殊的情况下可能没办法正常运行。

1、带环状结构的电路的处理

在同构判定中，确认某一节点是否匹配时，必须先确定与其相邻的节点是否匹配。然而，如果当电路中有环形结构的时候，由于处在递归遍历的过程中，所以判定程序的入口又重新回到自己，从而导致循环无穷地进行下去，使得判定程序永远停不下来。

为了避免这种情况的发生，需要对遍历过的点进行标记label=1，稍后当这个点再次作为匹配的入口节点时，则无需再比较。但是这样又产生了新的问题。当子电路中的点被遍历过，而相对应的主电路的点又未曾被遍历时，程序会返回错误的判定结果。因此，除了对是否遍历过进行标记之外，对遍历过的次数tag也要记录。只有这两项指标都相符合的节点才不需要进行再次比较。

2、多匹配情况的处理

以图3-1为例，子电路是由串联的两个与非门组成的，而主电路是三个与非门串联的结构。当采用建立候选集合的方法解决子图同构问题时，会得到主电路中有两个子结构是与子电路同构的结论。因为通过权值计算，主电路中权值大于或等于节点out权值的节点有三个：out1、out2和out3。经过判定，节点out1和out2都被判定为与节点out匹配的节点。

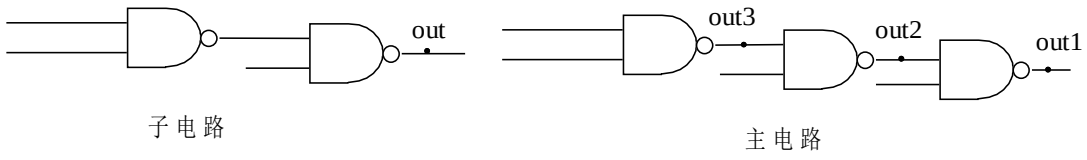


图3-1 多匹配的例子1

虽然这个结论对于识别过程并没有问题，但是在接下来的门级模型输出时就会发生逻辑错误。为了避免这种情况的发生和门级模型输出的方便，在本系统中所定义的子电路库中的电路暂时均为基本门，如与、或、非、与非和或非等。

但是这种方法又会导致另一种多匹配现象的产生，如图3-2所示，在输出t的节

点3处，既可以识别出与与门同构的子结构，又可以识别出与非门同构的子结构。为了避免门级模型输出时的混乱，就需要对已经匹配成功过的节点进行标记，使它不再参与接下来与其它类型子电路的比较。这种方法还可以进一步减少接下来参与匹配的节点个数。

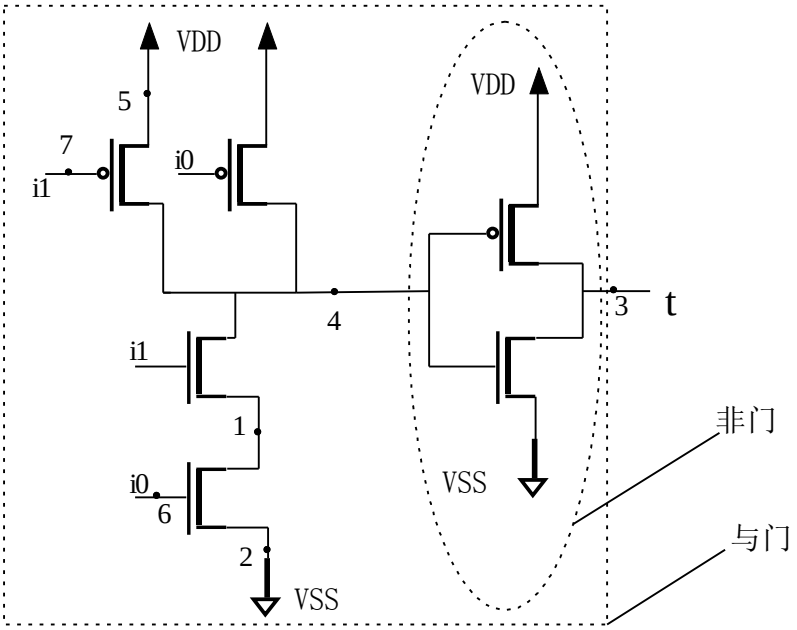
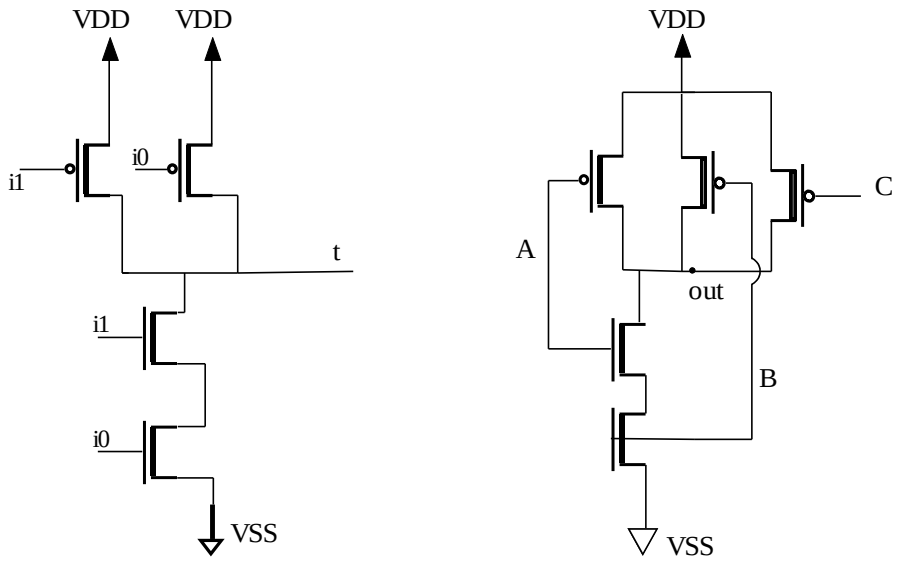


图3-2 多匹配例子2

第三种多匹配的例子如图3-3。图(b)中的电路不是标准的CMOS设计方式。假设我们用一个标准的NAND2的结构(a)与它相比较，仍旧可以得到该电路是与标准的NAND2同构的判定，其中A和B是输出节点。这是因为在判定的过程中，节点out会作为候选集合中的节点之一，而且当程序在标准的NAND2电路图(a)中遍历时，遍历完两个PMOS和两个NMOS后递归程序就会终止，而(b)的电路也有相同的结构，因此能得到正确的同构判定结果。



(a)标准NAND2　　(b)非标准NAND2

图3-3 多样匹配的例子3

3.1.2.2 图的存储

程序中需要用变量（如各种简单类型变量、数组变量等）保存被处理的数据和各种状态信息，变量在使用之前必须安排好存储：放在哪里、占据多少存储单元等等，这个工作被称作存储分配。但是许多运行中的存储需求在写程序时无法确定。本系统在实现中就存在这样的问题，因为系统处理的电路可能是非常小的电路单元，也可能是大规模的集成电路，而通过定义变量的方式不能很好地解决这类问题。为此就需要一种机制，使我们能利用它写出一类程序，其中可以根据运行时的实际存储需求分配适当大小的存储区，以便存放到在运行中才能确定大小的数据组。

C 语言为此提供了动态存储管理系统[45]。说是“动态”，因为其分配工作完全是在动态运行中确定的，与程序变量的性质完全不同。动态存储管理[14]的基本问题是系统如何应用用户提出的“请求”分配内存？又如何收回那些用户不再使用而“释放”的内存以备新的“请求”产生时重新进行分配？程序里可以根据需要，向动态存储管理系统申请任意大小的存储块。由于动态分配的存储块无法命名（命名是编程序时的手段，不是程序运行中可以使用的机制），而在一般的语言里都通过指针实现这种访问，用指针指向动态分配得到的存储块（把存储块的地址存入指针），所以通过对指针的间接操作，就可以去使用存储块了。引用动态分配的存储块是指针的最主要用途之一。与动态分配对应的是动态释放，即如果以前动态分配得到的存储块不再需要了，就应该考虑把它们交回去。

动态分配和释放的工作都由动态存储管理系统完成，这是支持程序运行的基础系统（称为程序运行系统）的一部分。这个系统管理一片存储区，如果需要存储块，就可以调用动态分配操作申请一块存储；如果以前申请的某块存储不需要了，可以调用释放操作将它交还管理系统。为了方便起见，将统称已分配给用户使用的地址连续的内存为“占用块”，称未曾分配的地址连续的内存为“可利用块”或“空闲块”。不管什么样的动态存储管理系统，刚开始时，整个内存区是一个“空闲块”。随着用户进入系统，先后提出存储请求，系统则依次进行分配。因此，在系统运行的初期，整个内存区基本上分隔为两大部分：低地址区包含若干占用块；高地址区（即分配后的剩余部分）是一个“空闲块”。例如图 3-4（a）所示为依次给 8 个用户进行分配后的系统的内存状态。经过一段时间以后，有的用户运行结束，它所占用的内存区变成空闲块，这就使整个内存区呈现出占用块和空闲块交错的状态，如图 3-4（b）所示。

(a)系统运动初期

(b)系统运行若干时间后

图 3-4 动态内存分配过程中的内存状态

综上所述，在本系统中采用动态存储管理的方法，不仅可以根据运行时的实际电路的规模分配适当大小的存储区，还能及时回收不用的内存，减少整个系统总的内存需求量。

3.3 系统实现框图

下面给出系统实现的功能框图，如图 3-5 所示。

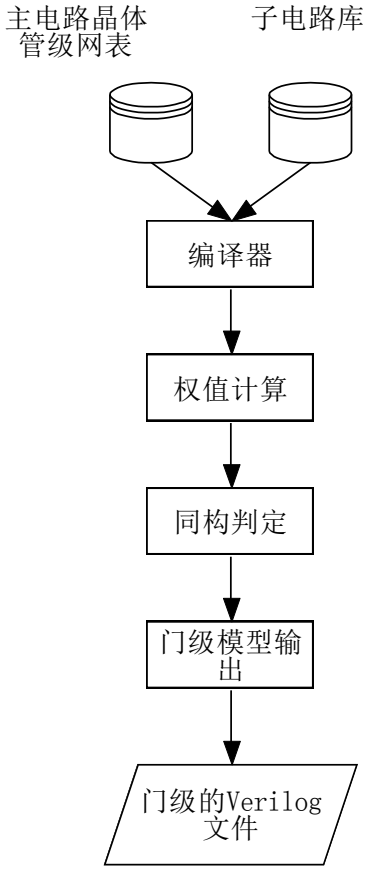


图 3-5 基于子图同构的门级模型抽取系统实现框图

如图 3-5 所示，系统实现中，主要有以下几个部分：

1、输入处理模块：编译器；

2、抽取模块：权值计算、同构判定；

3、输出模块：门级模型输出器。

综上所述，本文基于子图同构的设计思想，根据递归的实现策略，结合电路结构自身的特点，解决了一些特殊类型电路的识别问题。各个模块的具体实现在第四章进行详细的叙述。

第四章　模块设计及实现

4.1 功能模块分析

系统的实现框图如上一章图 3-5 所示，图中给出了具体的模块划分，本章将主要对各个功能模块的设计实现进行详细的分析和描述。

系统的模块主要分为三个部分：输入处理模块、抽取模块和输出产生模块。其中，输入模块是对输入的电路网表文件进行信息提取和存储，转换为系统的内部数据格式备用。抽取模块作为系统的核心部分，分为权值计算和同构判定两个子模块，共同完成门级模型的抽取。输出产生模块则配合抽取模块完成门级模型的输出工作。

4.2 编译器模块的设计实现

4.2.1 编译器模块的功能分析

该模块接受用户的设计描述，进行语法、语义检查。在检查通过后，将用户的描述数据转换为系统内部的数据格式，存入设计数据库中备用。

4.2.2 设计描述文件分析

描述电路晶体管层次的网表有很多不同格式，如比较通用的 spice 格式、Cadence 公司开发的 EDA 工具使用的 CDL 格式等。本系统主要可以接受 spice 和 Alliance（Xinlinx 开发的 EDA 软件）中使用的 al 这两种格式的网表文件。下面将分别介绍这两种格式的文件。

spice 是 Simulation Program with Integrated Circuit Emphasis 的缩写，是一种功能强大的通用模拟电路仿真器，已经具有几十年的历史了，该程序是美国加利福尼亚大学伯克利分校电工和计算科学系开发的，主要用于集成电路的电路分析程序中，spice 的网表格式变成了通常模拟电路和晶体管级电路描述的标准，其第一版本于 1972 年完成，是用 Fortran 语言写成的，1975 年推出正式实用化版本，1988 年被定为美国国家工业标准，主要用于 IC，模拟电路，数模混合电路，电源电路

等电子系统的设计和仿真。这种格式的网表使用广泛，为大家所熟悉，所以这里关于它的具体描述格式就不再累述了。

al 格式的网表是 Alliance 系列软件中生成的用于描述晶体管级电路的网表格式。本系统之所以采用该格式网表作为输入是因为独立完成编译器编写的工作量和难度非常大，特别是数据结构的确定关系到整个系统的成败。一个有效而且正确的可存储电路网表所有相关信息的数据结构的确定需要花费大量的时间，因此为了减低工作难度，本课题在实现时重用了 Alliance 系统中定义的部分数据结构，并在此基础上根据本课题的一些特殊需求做了修改。al 网表的描述格式如图 4-1：

```
V ALLIANCE : 4
H n1_y,L, 8/ 2/95
C i,IN,EXTERNAL,4
C f,OUT,EXTERNAL,2
C vdd,IN,EXTERNAL,3
C vss,IN,EXTERNAL,1
T N,1,5.3,1,4,2,1.7,1.7,14.2,14.2,9.6,13.5
T P,1,10.6,3,4,2,1.7,1.7,24.7,24.7,9.6,28.4
S 1,EXTERNAL,0.0111337,vss
S 3,EXTERNAL,0.0114575,vdd
S 4,EXTERNAL,0.0109221,i
S 2,EXTERNAL,0.0111909,f
EOF
```

图 4-1 al 网表示例

上图中 al 网表描述的是一个反相器电路。其中 C 表示外部端口“connector”，包括输入和输出端口。T 表示晶体管“transistor”，这里有两种类型的晶体管 N 型和 P 型。S 表示电路中的信号“signal”，包括内部信号和外部信号，该电路只有 4 个外部信号，分别标记为 1、2、3 和 4，与输入端口 vss、vdd 和 i 以及输出端口 f 相连接。

4.2.3 系统内部数据结构

根据上节描述，本系统使用的数据结构是根据自身特点，在 Alliance 系统数据结构上做了部分修改得到的。最终系统所采用的数据结构定义如下：

```
typedef struct lofig                    /* logical figure */
{
 struct lofig *NEXT ;                   /* next figure */
 struct chain *NODECHAIN ;              /* list of models */
 struct locon *LOCON ;                  /* connector list head */
 struct losig *LOSIG ;                  /* signal list head */
 struct ptype *BKSIG ;                  /* signal block list head */
 struct loins *LOINS ;                  /* instance list head */
 struct lotrs * LOTRS ;                 /* transistor list head */
 struct locap *LOCAP ;                  /* capacitance list head */
 struct lores *LORES ;                  /* resistance list head */
 struct loself *LOSELF ;                /* inductor list head */
 char          *NAME ;                  /* figure name (unique) */
 char           MODE ;                  /* 'A' or 'P' */
 struct ptype *USER ;                   /* Application specific */
}
lofig_list
```

图4-2 lofig结构定义

该结构是一个结构体，可用于存储不同数据类型的成员，由于电路网表中通常包括端口、信号、晶体管、电容、电阻等诸多信息，因此采用该结构可以较完整地存储整个电路的相关信息。而且在该结构体中，每一个成员又是一个结构体。例如用于存储晶体管相关信息的成员是指向lotrs类型的链表，链表的头指针为LOTRANS，其中lotrs结构体又由下面几个不同类型的项目构成：

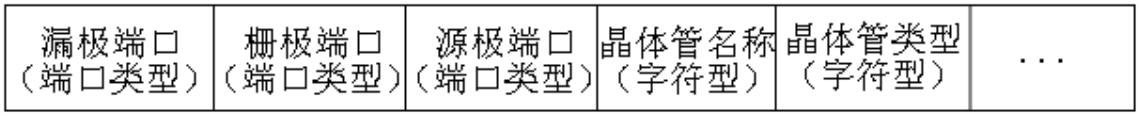


图4-3 lotrs结构体成员构成

值得注意的是，在定义电路信号存储结构时，在原结构的基础上，添加了用于标记该信号是否被遍历过的整型成员label和记录遍历次数的长整型成员tag。修改后的信号存储结构如下图所示：

struct losig *NEXT
struct chain *NAMECHAIN
long INDEX
char TYPE
struct lorcnet *PRCN
struct ptype *USER
int flag
long tag

图4-4 losig结构示意图

正是通过lotrs和losig这两个关键数据结构对电路重要信息的存储，保留了电路的器件和信号之间的连接关系，从而构造出电路相对应的图的结构。

4.2.4 编译器模块实现

编译器模块由两部分组成：分析器和检查器，如图 4-5 所示。

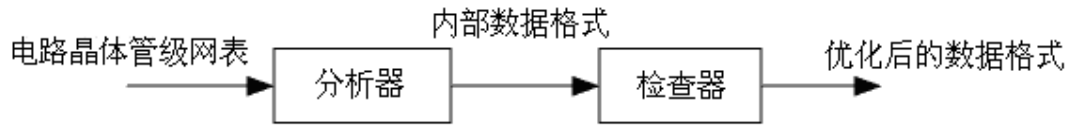
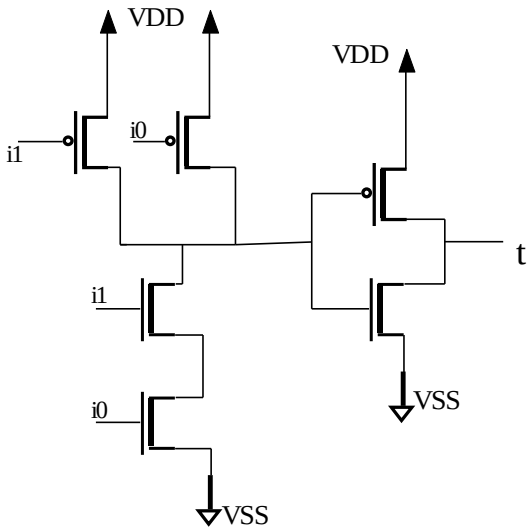


图 4-5 编译器模块实现框图

4.2.4.1 分析器的实现

分析器是对输入的电路晶体管级网表文件进行分析，包括语法和语义分析，并将网表描述的信息转化为系统内部的数据格式，存入数据库中备用。这是编译器模块的主要功能实现部分。检查器则是对已经形成的数据结构进行检查，检查其内部连接关系，对已经存储的数据进行筛选和优化。

下图 4-6 所示的是一个电路的晶体管结构及其对应的生成的内部数据格式。由于该电路中只有晶体管类型的器件，因此分别指向存储电容和电阻信息的结构 LOCAP 和 LORES 的指针的地址是 0x0。因为该电路的网表是展开的，即不包含任何的示例（instance），因此指向结构 LOINS 的指针的地址也为 0x0。



NEXT=0x0
MODELCHAIN=0x0
LOCON=0x97844b8
LOSIG=0x977c460
BKSIG=0x977937c
LOINS=0x0
LOTRS=0x97849a8
LOCAP=0x0
LORES=0x0
LOSELF=0x0
NAME=0x9770170'"
MODE=65'A'
USER=0x0

图 4-6 电路的晶体管结构及其对应的内部数据格式

4.2.4.2 检查器的实现

在某些情况下，经过分析器得到的内部数据格式并不是最终存在于数据库的格式，还需要经过检查的步骤。检查器针对晶体管级电路器件的连接情况，主要从以下几个方面进行检查及修改：

1、确认是否存在并联的晶体管，如果存在，则用一个等价的结构替换；

2、确认是否存在阻抗，即 NMOS 的栅极连在 VDD 上或 PMOS 的栅极连在 VSS 上；

3、确认是否存在二极管，即晶体管栅极和漏极信号相同，且栅极不连在 VDD 和 VSS 上；

4、检查是否存在电容，即晶体管的漏极与源极所连信号相同；

5、确认是否存在从未导通的晶体管，即 PMOS 栅极连在 VDD 上或 NMOS 栅极连在 VSS 上。

通过这些检查，对已经生成的内部数据格式做一些小的修改，形成最终的存储格式，存放在数据库中备用，同时在命令框里显示相应的警告或错误发生的信息。系统中定义的警告主要有以下几种：

[WAR]Drain of transistor is not connected 说明晶体管的漏极没有信号连接；

[WAR]Source of transistor is not connected 说明晶体管的源极没有信号连接；

[WAR]Transistor used as a diode 说明晶体管的漏极（或源极）连在了栅极上；
[WAR]Transistor used as a resistance 说明 P 型（或 N 型）晶体管的栅极接地（或电源）；
[WAR]Transistor is never passing 说明 P 型（或 N 型）晶体管的栅极接电源（或地）；
[WAR]Transistor used as a capacitance 说明晶体管的漏极和源极连在了一起；
[WAR]Transistor is (are) no used in the circuit 说明这些晶体管在电路中没有对任何的晶体管栅极起上拉或下拉作用；
[WAR]Conflict may occur on signal 说明该信号使得上拉电路和下拉电路同时导通。
定义的错误有以下几种：
[ERR] Transistor gate signal is not driven 说明晶体管的栅极信号没有被驱动；
[ERR]Gate of transistor is not connected 说明晶体管的栅极没有连接信号；
[ERR]No VDD connector in the circuit 说明电路中没有外部信号 VDD；
[ERR]No VSS connector in the circuit 说明电路中没有外部信号 VSS；
[ERR]Connector is power supply and ground 说明电路中有一个外部信号的名字同时为 VDD 和 VSS。

4.3 权值计算模块的设计实现

4.3.1 权值计算模块的功能分析

权值计算模块主要根据电路的连接情况，按权值计算公式，对各个信号节点的权值进行计算。因为权值计算过程实际上是一个迭代的过程，在 C 语言中通过递归调用权值计算函数即可方便地实现。由于权值计算的具体算法在 2.4.2 节中已有详细的说明，因此关于权值计算函数 weight_compute( )的构造在本节中就不再累述。在本模块的设计实现中，除权值计算函数的构造外，权值的存储是另一个关键问题。

4.3.2 权值的存储

在本模块中，通过递归运算得到的各节点的权值存储在一维数组里。通常申明一个数组时必须指定数组长度，如果用户不知道需要多大的数组，就可能趋向于指定一个超出实际需要的数组长度。另一方面，如果数组长度过小，就必须编辑程序，改变数组长度，并重新编译程序。为了解决这一问题，也是为了使模块

适用于不同规模的电路，因此本模块采用的是动态一维数组。

所谓动态一维数组是指在申明定义数组时用指针来表示，而没有给出数组确定的大小。在 C 语言中，动态一维数组是通过 malloc 动态分配空间来实现的。值得注意的是，由于数组结构具有根据下标访问的特点，因此在储存时，我们将数组元素的下标和信号节点的索引一一对应，使得节点权值的读取更为方便。例如下面的两条 C 语句：

```
unsigned long* pt_weight_net;
pt_weight_net=(unsigned long)*malloc((length+1)*sizeof(unsigned long));
```

这是系统中用来申明动态一维数组 pt_weight_net(length+1)所用的语句，pt_weight_net 是一个指向 unsigned long 型的指针，分配的内存大小为(length+1)*sizeof(unsigned long)，其中 length 代表的是节点的个数。注意程序结束后要用 free( )将其释放，否则内存会泄漏。以图 4-6 中电路为例，经过权值计算后，最终得到的结果存储情况如下所示：

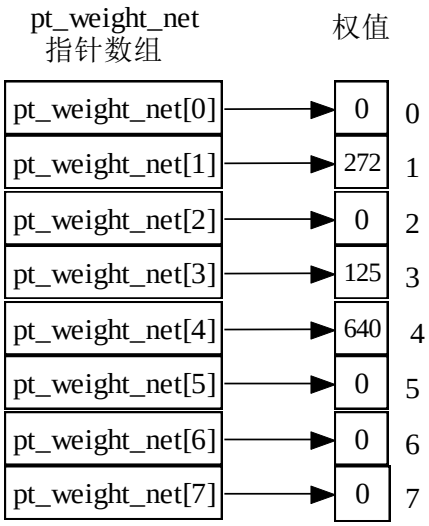


图 4-7 权值存储情况图

从图中可以发现，由于数组的下标是从 0 开始的，但是为了节点的索引与数组的下标一一对应，所以虽然电路图中只有 7 个节点，即 length=7，但需要分配的空间大小却是(7+1)*sizeof(unsigned long)。

4.3.3 权值计算模块实现

权值计算模块分成两个部分：功能模块和一维数组，其中功能模块是通过有限次地调用权值计算函数 weight_compute( )实现的。实现框图如图 4-8 所示。

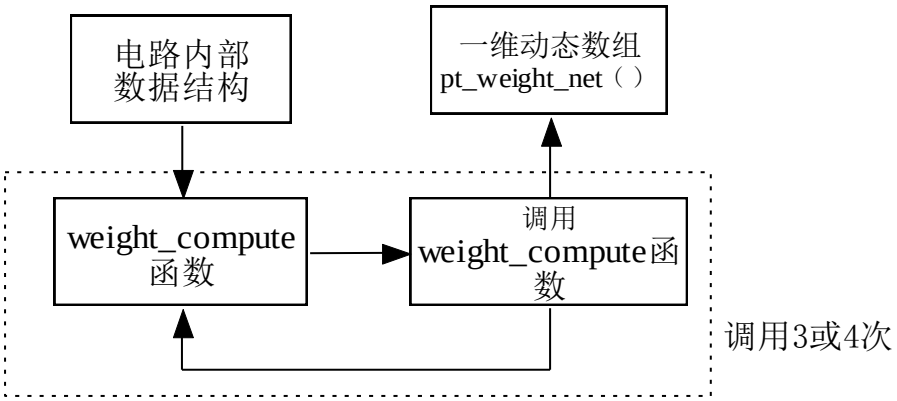


图 4-8 权值计算模块实现框图

在图 4-8 中，根据从编译器模块中得到的电路信息，通过递归调用权值计算函数完成权值计算。递归调用的次数一般为 3 次或 4 次，这一点在前面也已经讨论过。最后将计算结果存在一个一维的动态数组中备用。不管电路中有多少数量的节点需要计算权值，动态分配都会根据节点的数量来分配数组的内存空间。而且系统读入的主电路和子电路都需要按照相同的方法计算它们的节点权值，分别存在不同的数组里备用。

4.4 同构判定模块的设计实现

4.4.1 同构判定模块功能分析

同构判定模块功能的实现分为两个阶段：第一阶段确定判定的起始点和匹配候选集；第二阶段采用递归的方式进行同构判定，即判定起始点是否与匹配候选集中的节点匹配。第一阶段得到的匹配候选集的大小直接影响了第二阶段同构判定的效率。

4.4.2 起始点和匹配候选集的确定

对于起始点的选择原则在 2.4.4.2 节中早有分析，将子电路中的主要输出节点作为起始点。这一原则的确定是本系统对原 DECIDE 算法的一个重要改进。在具体实现上只是一个遍历查找的简单操作。

在内部格式中存在一个专门用于存储电路外部端口的结构 locon，它的组成如下图所示：

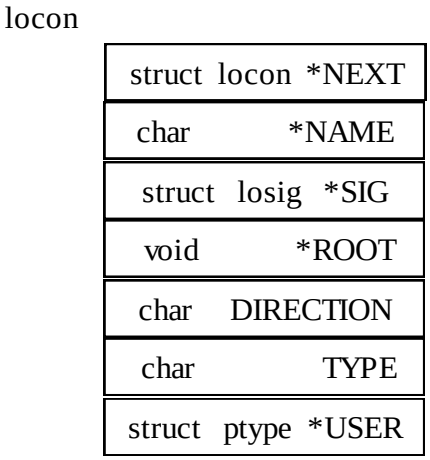


图 4-9 locon 结构示意图

在 locon 结构中有一个成员 DIRECTION，它是用于存储电路外部端口的方向。例如外部端口为输入端口，那么它的 DIRECTION 就存为‘I’，如果该端口为输出端口，那么它的 DIRECTION 就存为‘O’。因此通过遍历子电路的 locon 结构，查找到 DIRECTION 为‘O’的端口，将它所连接的信号节点（即结构中的成员 SIG）作为起始点，并将它的索引记录下来。

在确定起始点之后，接下来就是确定候选集中的节点。确定原则是将主电路中所有大于或等于起始点权值的端节点组成一个匹配候选集。由于在权值计算时，每个端节点的权值和索引都一一对应的存放在一个一维数组中，因此只要将数组中的每一个权值与起始点权值比较大小，将符合条件的节点做上标记即可。

4.4.3 同构判定

同构判定在这一阶段已经转化为节点的匹配判定，换句话说，就是判定候选集中的节点是否与子电路的起始节点相匹配，以确定主电路中是否存在与子电路同构的结构，以及存在几个结构。判定的流程图如图 4-10 所示：

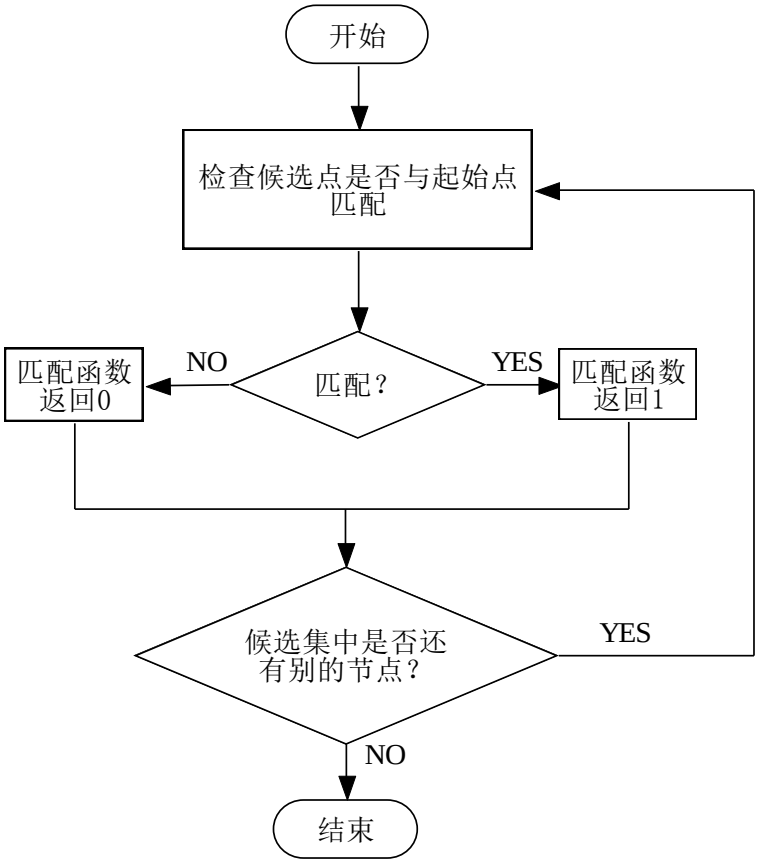


图 4-10 匹配判定流程图

从流程图中可以看到，判定是否匹配的关键是匹配函数。对于候选集中的每一个节点，在判定它是否与起始点匹配时都要调用匹配函数。下面将详细介绍该函数的实现。

在本系统中，由于整个匹配判定的过程是一个递归的过程，即判断一个节点是否与起始点匹配时，必须从起始点出发，判断起始点的每一个邻节点都与它的邻节点是否匹配，因此在系统的实现时，将匹配函数设计成一个递归函数。

递归函数是指一个含直接或间接调用本函数语句的函数。当函数调用自己时，在栈中会为新的局部变量和参数分配内存，函数的代码用这些变量和参数重新运行。但是递归调用并不是把函数代码重新复制一遍，仅仅参数是新的。所以当每次递归调用返回时，老的局部变量和参数就从栈中消除，从函数内此次函数调用点重新启动运行。因此可递归的函数被理解为自身的“推入”和“拉出”。

匹配算法如下所示：

```
identify(node_main, node_sub){
if (node type equal)
 {
   if (node type is not input signal)
      if (number of p_neighbor equal and number of n_neighbor equal)
          if ( node_sub is drain/source) /*  for signal which is the drain/source
                                                of the transistor*/
             if( node_sub_gate_neigh is internal)
                if (identify(node_main_ds_neighbor, node_sub_ds_neighbor)&&
                     identify(node_main_gate_neighbor, node_sub_gate_neighbor))
                   return 1;
                else return 0;
             else
                if(identify(node_main_ds_neighbor, node_sub_ds_neighbor))
                   return 1;
                else return 0;
          else         /* for signal which is the gate of the transistor*/
               if (identify(node_main_ds_neighbor, node_sub_ds_neighbor))
                    return 1;
               else return 0;
      else return 0;
else                    /*for vdd/vss*/
     if (node_sub and node_main are both vdd or vss)
          return 1;
     else return 0;
}
else return 0;
}
```

图 4-11 identify 函数算法

如图4-11这样的递归函数，实际上是一个循环函数，为了循环不会无止境地进行下去，需要终止条件。在本系统中设定终止条件为：

(1)遇到VDD/GND；

(2)遇到主要输入节点；

(3)该节点所有的邻节点都已经匹配。

当上述任何一条终止条件得到满足时，递归终止，返回匹配的结果。若identify函数返回1，则说明进行比较的两个节点匹配，否则表示匹配失败。但是不管匹配成功与否，都需要将候选集中的每一个节点与起始点都做一次匹配检查，以确认主电路中包含的与子电路同构的单元的数量。

匹配的成功与否，即identify函数返回1还是0，取决于一些规则。例如当两个选定的节点，如果它们的类型不同，则直接返回0。如图4-12所示，以一个not门为

例来说明identify的过程。

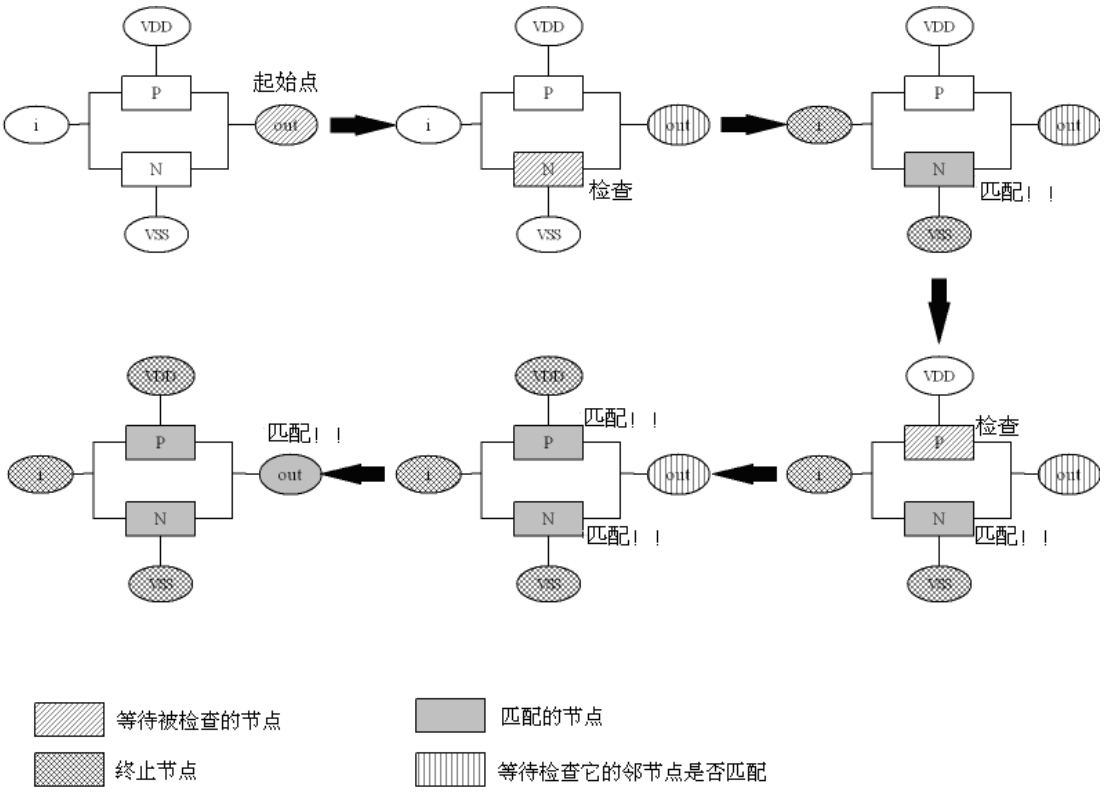


图4-12 not门的identify函数遍历过程

4. 4. 4 同构判定模块实现

根据同构判定模块的功能划分，该模块的设计实现主要分为两部分：一部分是起始点的判定和匹配候选集中节点的标记，一部分是匹配判定。

由于是通过权值的比较来确定匹配候选集，因此我们要把前面用来存储权值的一维动态数组结构改成包含该数组的结构体。这是因为在 C 语言中，结构体成员可以是任何类型的。当成员为数组时，程序可以像引用任何数目一样引用数组成员。系统中使用的数组结构定义如下：

```
typedef struct netweight
{
  unsigned long *pt ;
  int label ;
  } weight ;
```

图 4-13 权值存储结构

图 4-14 中是参与同构判定的主电路与子电路，通过权值计算和比较后，将子电路中权值最大的节点 2 作为匹配的起始点，标记为 1。主电路中的节点 1，3 和 4 的权值都不小于起始点权值，因此也标记为 1，共同组成匹配候选集，它们在内存中的存储情况如图 4-15 所示。

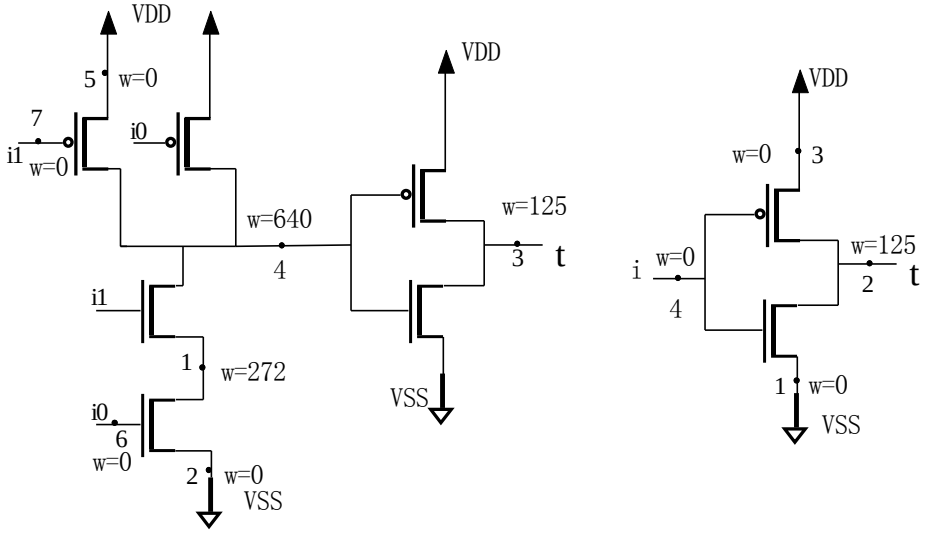


图 4-14 主电路与子电路

	*pt	label
weight[0]	0	0
weight[1]	272	1
weight[2]	0	0
weight[3]	125	1
weight[4]	640	1
weight[5]	0	0
weight[6]	0	0
weight[7]	0	0

主电路

	*pt	label
weight[0]	0	0
weight[1]	0	0
weight[2]	125	1
weight[3]	0	0
weight[4]	0	0

子电路

图 4-15 结构数组存储示意图

在确定好起始节点和匹配候选集后，通过循环调用 identify 函数，检查候选集中的每一个点是否与起始节点匹配。以图 4-14 中的电路为例，通过匹配判定，只有 identify(3,2)返回 1，说明主电路图中只有节点 3 和起始节点 2 匹配。

4.5 门级模型输出模块的设计实现

4.5.1 门级模型输出模块功能分析

门级模型输出模块的主要功能是产生一个能反映晶体管级电路门级连接关系的文件。该文件能作为后面等价性验证工具的输入，检验其与参考电路的等价性关系。门级模型输出模块实现的关键是输出格式的确定以及门级模型的输出。

4.5.2 门级输出格式确定

在本系统中，门级模型将通过 Verilog 硬件描述语言来描述。对于采取该语言进行描述的原因和它的基本语法下面将具体介绍。

4.5.2.1 Verilog HDL 简介

早期的集成电路设计实际上就是掩膜设计；电路的规模是非常小的，电路的复杂度也很低，工作方式主要依靠手工作业和个体劳动。40 年后的今天，超大规模集成电路的电路规模都在百万门量级；由于集成电路大规模、高密度、高速度的需求，使电子设计越来越复杂，为了完成 10 万门以上的设计，需要制定一套新的方法，就是采用硬件描述语言设计数字电路。HDL（Hardware Description Language）于 1992 年由 Iverson 提出，随后许多高等学校、科研单位、大型计算机厂商都相继推出了各自的 HDL，但最终成为 IEEE 技术标准的仅有两个，即 VHDL[43]和 Verilog HDL[44]。Verilog HDL 语言提供非常简洁，可读性很强的语法，使用 Verilog 语言已经成功地设计了许多大规模的硬件。

Verilog HDL 是在 1983 年，由 GDA（Gate Way Design Automation）公司的 Phil Moorby 首创的。Phil Mooby 后来成为 Verilog-XL 的主要设计者和 Cadence 公司（Cadence Design System）的第一个合伙人。在 1984-1985 年 Moorby 设计出第一个关于 Verilog-XL 的仿真器，1986 年他对 Verilog HDL 的发展又做出另一个巨大贡献，提出了用于快速门级仿真的 XL 算法。

随着 Verilog-XL 算法的成功，Verilog HDL 语言得到迅速发展。1989 年，Cadence 公司收购了 GDA 公司，Verilog HDL 语言成为 Cadence 公司的私有财产。1990 年，Cadence 公司公开了 Verilog HDL 语言，成立了 OVI（Open Verilog International）组织来负责 Verilog HDL 的发展。IEEE 于 1995 年制定了 Verilog HDL 的 IEEE 标准，即 Verilog HDL 1364-1995。

1987 年，IEEE 接受 VHDL（VHSIC Hardware Description Language）为标准 HDL，即 IEEE1076-87 标准，1993 年进一步修订，定为 ANSI/IEEE1076-93 标准。现在很多 EDA 供应商都把 VHDL 作为其 EDA 软件输入/输出的标准。例如，Cadence、Synopsys、Viewlogic、Mentor Graphic 等厂商都提供了对 VHDL 的支持。

由于 Verilog HDL 早在 1983 年就已推出，至今已有十三年的历史，因而 Verilog HDL 拥有广泛的设计群体，成熟的资源比 VHDL 丰富。而 Verilog HDL 与 VHDL 相比最大的优点是：它是一种非常容易掌握的硬件描述语言，而掌握 VHDL 设计技术就比较困难。

目前版本的 Verilog HDL 和 VHDL 在行为级抽象建模的覆盖范围方面也有所不同。一般认为 Verilog HDL 在系统抽象方面比 VHDL 强一些。Verilog HDL 较为适合算法级（Alogrithm）、寄存器传输级（RTL）、逻辑级（Logic）、门级（Gate）、设计。而 VHDL 更为适合特大型的系统级（System）设计。

因此，在本系统中采用 Verilog HDL 对电路的门级进行建模。

4.5.2.2 Verilog HDL 基本语法

作为高级语言的一种，Verilog 语言以模块集合的形式来描述数字系统，其中每一个模块都有接口部分，用来描述与其它模块之间的连接。一般说来，一个文件就是一个模块，但又不绝对如此。这些模块是并行运行的，但通常用一个高层模块来定义一个封闭的系统，包括测试数据和硬件描述。这一高层模块将调用其它模块的实例。

模块代表硬件上的逻辑实体，其范围可以从简单的门到整个大的系统，比如，一个计数器、一个存储子系统、一个微处理器等。模块的结构如下：

module 〈模块名〉（〈端口列表〉）；

〈定义〉

〈模块条目〉

endmodule

其中，〈模块名〉是模块唯一性的标识符；〈端口列表〉是输入、输出和双向端口的列表，这些端口用来与其它模块进行连接；〈定义〉一段程序用来指定数据对象为寄存器型、存储器型、线型以及过程块，诸如函数块和任务块；而〈模块条目〉可以是 initial 结构、always 结构、连续赋值或模块实例。

Verilog 语言有三种描述方法：

1、结构型描述：通过实例进行描述的方法。将 Verilog 预定义的基元实例嵌入到语言中，监控实例的输入，一旦其中任何一个发生变化，便重新运算并输出。

2、数据流型描述：这是一种描述组合功能的方法，用 assign 连续赋值语句来实现。连续赋值语句完成如下的组合功能：等式右边的所有变量受持续监控，每当这些变量中有任何一个发生变化，整个表达式被重新赋值并送给等式左端。这种描述方法只能用来实现组合功能。

3、行为型描述：这是一种使用高级语言的方法，它和用软件编程语言描述没有什么不同。具有很强的通用性和有效性。它是通过行为实例来实现的，关键词是 always，其含义是，一旦赋值给定，仿真器便等待变量的下一次变化，有无限循环之意。

4.5.2.3 门级模型输出格式分析

根据对 Verilog 语言的分析，结合本系统的特点，门级模型输出文件将采用第一种描述方法——结构型描述。而结构型描述又分为以下几种：

1、内置门原语（在门级）；

2、开关级原语（在晶体管级）；

3、用户定义的原语（在门级）；

4、模块实例（创建层次结构）。

不管采用何种方式，都是通过线网来连接。

在本系统中采用的是内置门原语和用户定义的原语这两种结构描述方式。下面以二选一的选择器为例，采用内置门原语的结构描述方式描述它的结构：

```
module MUX2_1 (out, a, b, sel);   //端口定义
    output out;
    input a, b, sel;              //输入输出列表
    not (sel_, sel);
    and (a1, a, sel_);
    and (b1, b, sel);
    or (out, a1, b1);     //结构描述
endmodule
```

对应的电路硬件图如图 4-16 所示：

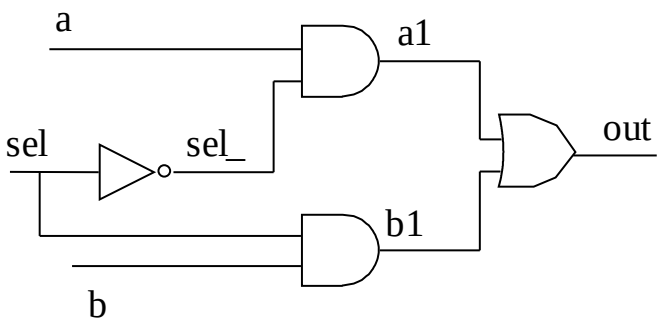


图 4-16 MUX2_1 电路模块示意图

在这一实例中，a, b 和 sel 是设备的输入端口，out 是输出端口，所有信号都从这些端口流入和输出。and，or，not 是 Verilog 中内置门原语。关键词 module 和 endmodule 之间包含完整的二选一多路选择器的设计实现。当在其它模块中用到这一多路选择器的模块时只需使用其模块名和所定义的端口名，不需要知道其内部的具体实现。

4.5.4 门级模型输出模块实现

由于系统的门级模型采用 Verilog 的结构描述的方法，因此输出文件具有如下的结构：

module module_name (port1, port2, ...) ;

// Declarations;

Input,output,inout，

reg, wire, parameter，

function, task, . . .

//S t a t e m e n t s;

Gate instantiation

endmodule

其中在Verilog中定义的内置门有and、nand、or、nor、not、xor、xnor和buf，因此在子电路库中必须包含这些内置门的晶体管级网表。除此之外，还有诸如简单的选择器，加法器等这样的基本电路。

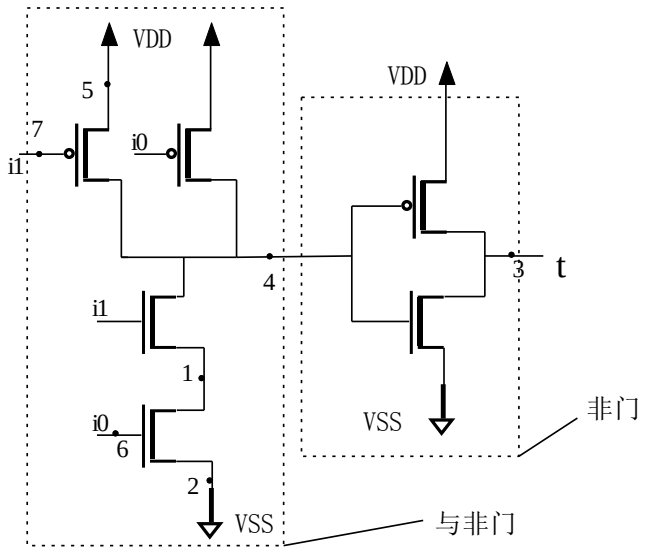


图 4-17 门级模型例子

以图 4-17 所示电路为例，经过同构判定，发现电路中包含两个子电路，分别为 not 门和二输入的 nand 门。因此，它的门级模型描述为：

```
module a2p_y(t,i0,i1) ;
ouput t;
input i0,i1;
wire i2;
nand(i2,i0,i1);
not(t,i2);
endmodule
```

实际上，这个 Verilog 描述文件的产生还涉及到 C 语言中关于文件的一些操作。文件在进行读写操作之前要先打开，使用完毕要关闭。所谓打开文件，就是建立文件的各种有关信息，并使文件指针指向该文件，以便进行其它操作。关闭文件则断开指针与文件之间的联系，也就禁止再对该文件进行操作。 在 C 语言中，文件操作都是由库函数来完成的。

在 C 里使用 fopen 函数打开文件，格式为:

fopen(文件名, 文件使用方式)；

此函数返回一个指向 FILE 类型的指针，其中文件使用方式有如下几种：

表 4-1 文件使用方式表

字符	含义
“r”	打开文字文件只读
“w”	创建文字文件只写
“a”	增补，如果文件不存在则创建一个
“r+”	打开一个文字文件读/写
“w+”	创建一个文字文件读/写
“a+”	打开或创建一个文件增补
“b”	二进制文件（可以和上面每一项合用）

当文件的读写操作完成之后, 使用 fclose 函数关闭文件。格式如下:

fclose（文件指针）;

该函数返回一个整型数。当文件关闭成功时，返回 0，否则返回一个非零值。可以根据函数的返回值判断文件是否关闭成功。

因此，门级模型输出模块的实现框图如下：

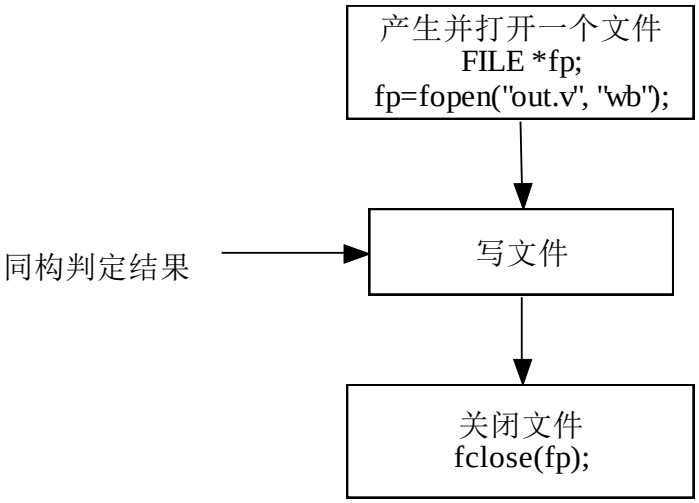


图 4-18 门级模型输出模块实现框图

如图 4-18 所示，根据前一阶段同构判定的结果，程序将主电路中与子电路一一对应匹配的节点记录下来，作为 Verilog 门级描述中各个门实例的端口。主电路模块的输入、输出端口所连信号以及内部信号情况则可通过查询主电路端口数据结构 locon 和信号数据结构 losig 得到。

4.6 基于 LINUX 的系统实现

由于 LINUX 下 C 编程通常包括三个步骤：源代码输入，程序编译、链接、运行以及调试。在按照前几节的设计实现思路对编写好系统的代码后，目前的关键问题是对写好的程序进行编译、链接、运行和调试。

4.6.1 系统的自动化编译和链接

为了能实现自动化编译，就需要根据事先编好的头文件和源文件编写出一个 makefile 文件。makefile 文件一旦写好，只需要一个 make 命令，整个工程完全自动编译，极大的提高了软件开发的效率。make 是一个命令工具，是一个解释 makefile 中指令的命令工具，一般来说，大多数的 IDE 都有这个命令，比如：Delphi 的 make，Visual C++的 nmake，Linux 下 GNU 的 make。

实际上，makefile 是 Unix 和 Linux 下的 C/C++项目的工程管理文件，它关系到整个工程的编译规则。一个工程中的源文件有很多，其按类型、功能、模块分别放在若干个目录下，makefile 定义了一系列的规则来指定，哪些文件需要先编译，哪些文件需要后编译，哪些文件需要重新编译，甚至于更复杂的功能操作，因为 makefile 就像一个 shell 脚本一样。

编写一个 makefile 来告诉 make 命令如何编译和链接这几个文件，一般的规则是：

1、如果这个工程没有编译过，那么所有 C 文件都要编译并被链接；

2、 如果这个工程的某几个 C 文件被修改，那么只编译被修改的 C 文件，并链接目标程序；

3、如果这个工程的头文件被改变了，那么需要编译引用了这几个头文件的 C 文件，并链接目标程序。

只要 makefile 写得够好，make 命令会自动智能地根据当前的文件修改的情况来确定哪些文件需要重编译，从而自己编译所需要的文件和链接目标程序。

4.6.2 系统的运行和调试

GNU DDD 是命令行调试程序，如 GDB、DBX、WDB、Ladebug、JDB、XDB、Perl Debugger 或 Python Debugger 的可视化图形前端。它特有的图形数据显示功能（Graphical Data Display）可以把数据结构按照图形的方式显示出来。

图 4-19 显示的是 DDD 的主窗口。它主要由选单栏、工具条、数据窗口、源文件窗口、机器码窗口、控制台和命令工具窗口等几部分组成。其中，数据窗口用于观察复杂的数据结构，在删除数据之后，显示仍然有效；源文件窗口显示源代码、中断点和当前执行到达的位置，选择该窗口中的“Display”项，可以显示任意运算式的值；机器码窗口显示当前所选函数的机器代码，但仅对于 GDB 来说是可用的；在 Debugger 控制台里，用户可以于 DDD 内置调试器的命令行介面进行交互，等同于执行命令工具栏中的命令。

图 4-19　DDD 主窗口

开始调试之前，必须用程序中的调试信息编译要调试的程序。这样，ddd 才能够调试所有使用的变量、代码行和函数。如果要进行编译，需要在 gcc 下使用额外的 -g 选项生成调试信息。由于在编译时我们编写了一个 makefile 文件，因此只需在该文件中的 gcc 添加上-g 选项即可。

如果一切正常，程序将执行到结束。程式中是有错误的，需要进行调试，输入下面的命令启动 DDD 调试器，调试这个可执行程式：

#ddd main

一段时间之后，DDD 的主窗口就会出现。找到怀疑出错的地方，在相应的代码上设置中断点对程序进行调试。

4.7 系统工作流程

对系统进行编译和调试后，整个系统运行的流程如下所示：

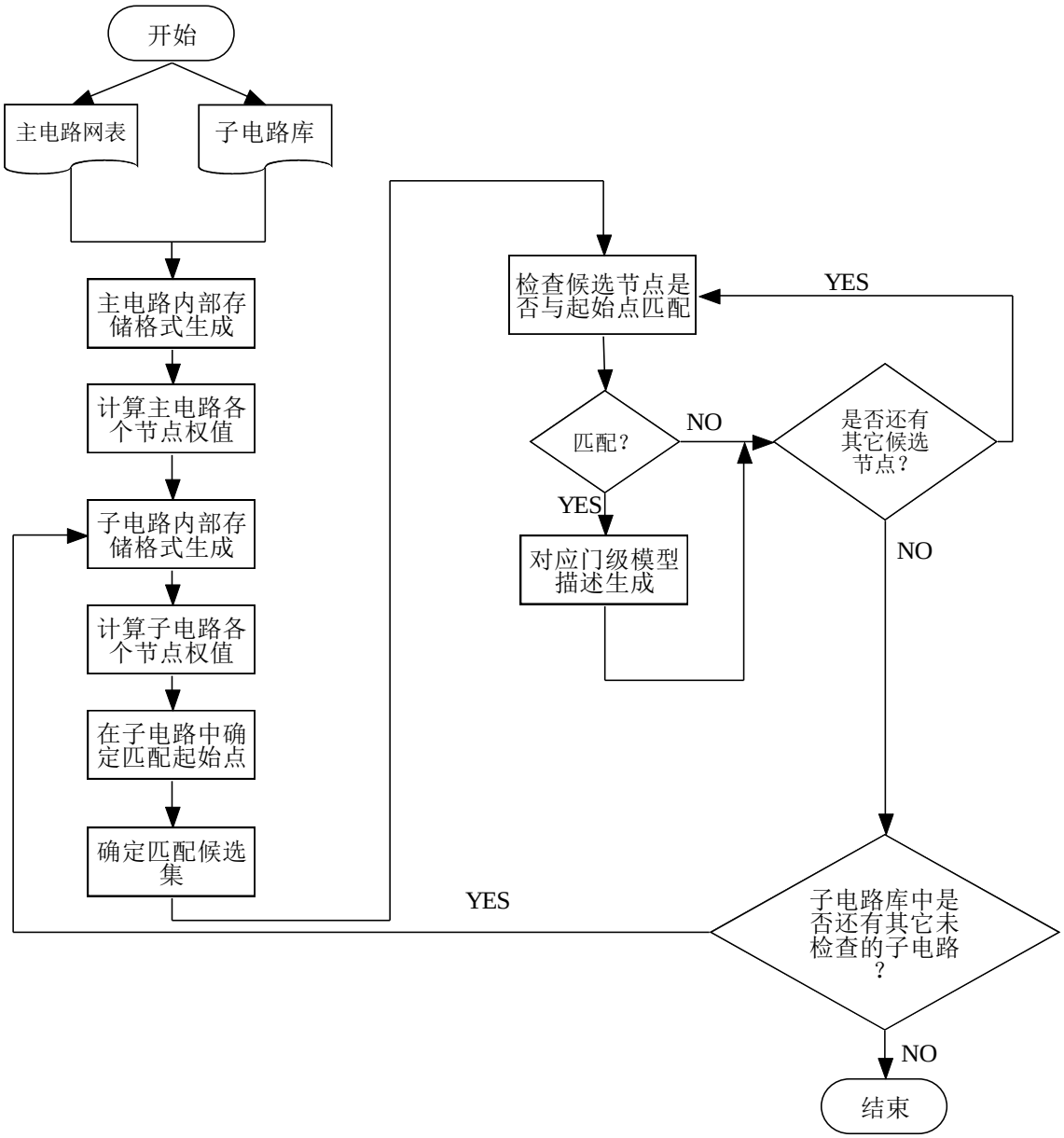


图 4-20　系统总流程图

配合图 4-20 系统总流程图，系统的流程用文字说明如下：

（1）读入主电路晶体管级网表文件，计算出包含的节点和晶体管数量以及节点权值；

（2）读入子电路库中子电路的晶体管级网表文件，计算出包含的节点和晶体管数量及节点权值，并将子电路中权值最大的节点作为匹配的起始点；

（3）将主电路中所有权值比起始点大的节点作为匹配侯选集，只有匹配侯选集中的节点才参与匹配；

（4）依次检验匹配侯选集合中的节点是否与起始节点匹配，若匹配，输出对应门级模型的Verilog描述；

（5） 检查子电路库中是否有其它未匹配的子电路，若有，重复（2）-（4）；

（6） 若无，结束。

第五章 系统测试及实验结果

5.1 系统的编译和运行

系统的设计由 C 语言完成，在 x86 平台(Intel Celeron 1.80GHz CPU，224MB RAM，Red Hat Enterprise Linux 3)上编译运行，具体操作步骤如下所示：

（1）cd ./pattern matching 进入 pattern matching 目录下；

（2）make 编译，得到可执行文件；

（3）./main 运行可执行文件。

5.2 系统资源使用情况分析

由于我们所采用的算法是递归式的，从所选定的起始点开始，会一直不断调用自己，直到遇到所设定的终止条件（VDD、GND或是主要输入节点）为止。假设子电路中起始点到VDD、GND或是主要输入节点为止的平均层数为L，且在不失一般性的前提下，令每个节点有3个邻节点，则我们所提出的identify函数的总共所需时间为：$T(L)=3\times T(L-1)=3\times 3\times T(L-2)=3\times 3\times\cdots 3\times T(0)=3^{L}\times T(0)$（$T(0)$：相对于VDD/GND/主要输入节点所需的时间）。整个匹配算法的 Big-O 表示式为：$O(3^{L})$，但实际上由于匹配的过程中，一旦有一个邻节点匹配不成功，那么即使剩下的邻节点全部都匹配成功，该节点也不会匹配成功，因此只要有一个邻节点匹配失败，程序即可终止，不必继续剩下的匹配工作。以这种分支定界（branch and bound） 的方式可以大幅缩短程序执行的时间，一个候选节点往往只需一两个步骤就可以结束。因此在速度上比以往一般图的遍历的匹配方式效果更好。 一般来说，在最坏情况下，如果遍历的深度为 N_p，由主电路得到的候选节点的个数为 N_c，那么复杂度为 $o(N_c\cdot 3^{N_p})$。由于分支定界的方式会缩短程序执行的时间，因此实际的运行时间会小于 $o(N_c\cdot 3^{N_p})$，但是在 $o(N_c\cdot 3^{\log(N_p)})\sim o(N_c\cdot N_p)$ 之间。

从另一方面对内存的占用情况进行分析，由于采用的是通用的数据结构，因此每个节点所占用的空间是相同的。内存空间与节点的数量成正比，因此内存占用复杂度为 $o(N_p+N_t)$，其中 N_t 代表的是主电路节点的个数。

5.3 实验结果

系统在 x86 平台(Intel Celeron 1.80GHz CPU，224MB RAM，Red Hat Enterprise Linux 3)上运行。

表5-1说明的是子电路规模与权值计算的迭代次数和候选节点个数之间的关系。根据实验结果可以发现对于与、或、非等基本单元电路，由于它们包含的MOS管数目一般不超过4对，所以在3～4次迭代计算后，候选集中节点个数不会发生太大变化。而对于较复杂的，包含的晶体管数目超过4对的子电路，则需要进行多于3～4次的迭代，才能将电路的信息收集完整，从而确定出正确的候选点，避免过滤掉本应匹配的节点。

表5-1 迭代次数与候选点个数的关系

主电路vs子电路	子电路MOS管个数	不同迭代次数对应的候选节点个数				
		2次	3次	4次	5次	6次
a2p_y vs invert	2	3	3	3	3	3
mx3_y vs nand2	4	4	4	5	5	5
ms2rx vs nand2	4	4	6	6	6	6
a2 vs b2	6	35	39	39	40	42
a3 vs b3	32	37	38	38	50	72

根据表5-1的数据，将迭代次数设为3，C_P=3，C_N=2，测试从规模不同的电路中识别同一子电路所需的时间及候选集合中节点的个数，结果如表5-2所示。其中主电路和子电路的规模大小通过电路图的节点个数来反映。

表5-2 测试结果

主电路（T）	子电路（S）	T的大小	S的大小	候选节点个数	识别出的S的个数	识别所用时间（s）
ms2rx_y	inverter	52	6	14	3	0.044
adder4	inverter	218	6	80	22	0.139
core	inverter	371	6	117	18	0.189
addaccu	inverter	810	6	153	22	0.359
calcul	inverter	4715	6	1585	326	6.205
tuner	inverter	7915	6	2654	462	13.879
amd2901	inverter	9287	6	3090	560	19.918

从实验结果可以看出，通过权值比较对主电路的节点进行筛选后，大大减少了需要进行匹配的节点个数，与原电路节点的个数相比平均减少了67.2%。而且对

同一子电路而言，识别所需时间基本上与候选节点的个数呈线性关系，如图5-1所示。与文献[31]实验结果相比，识别的准确性也得到了提高，能准确识别出主电路中包含的所有子电路。

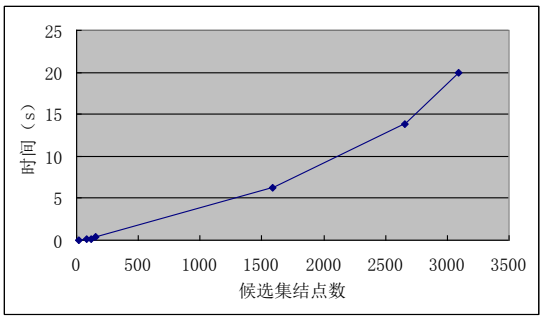


图5-1 候选集节点数与识别时间的关系

最后测试整个门级模型抽取器的抽取效果。其中电路输入为一系列规模不同的电路晶体管级网表，除电路 A2P、C17、C499 和 C6288 为组合电路外，其它电路均带有时序单元；子电路库则重用了 Alliance 系统的部分库文件，实验结果如表 5-3 所示。

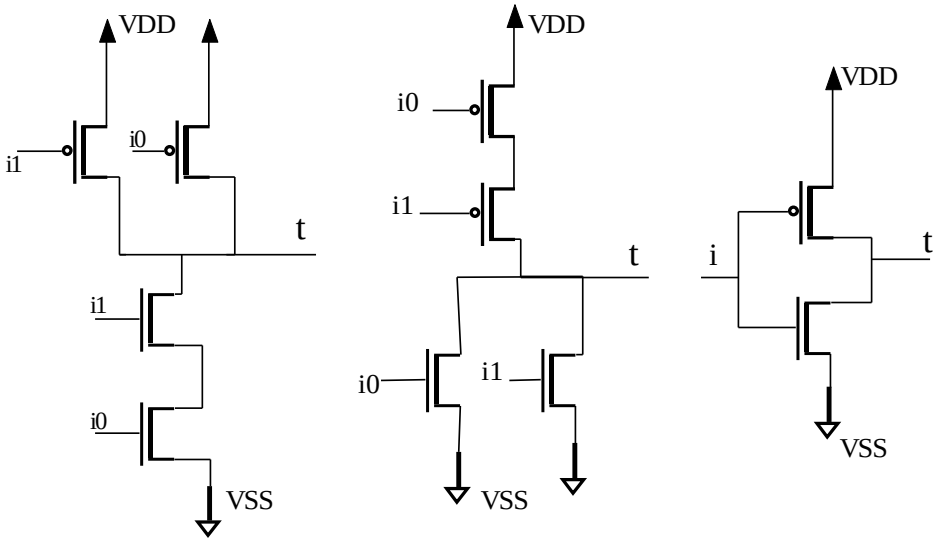
表 5-3 实验结果

电 路		可 识 别 门数量	抽取率	时间(s)
名称	晶体管个数			
A2P	6	2	100%	5.493
C17	24	6	100%	6.530
C499	1796	202	100%	43.658
C6288	10142	2416	100%	200.154
core	242	19	47.1%	4.697
addaccu	642	38	35.5%	9.230
amd2901	6166	602	27.6%	166.058

实验结果表明，由于系统的子电路库中暂时只定义了一些基本门电路的晶体管结构，没有定义时序单元如触发器、缓存器的晶体管结构，因此该系统只对组合电路的抽取比较准确，能得到它们完整的门级模型，门级模型的正确性可通过 QUARTUS II 仿真得到。但是对带时序单元的电路，只能识别出电路中的组合部分，达到部分抽取的效果。

此外，对组合电路的实验结果进行进一步分析。因为在子电路库中基本结构

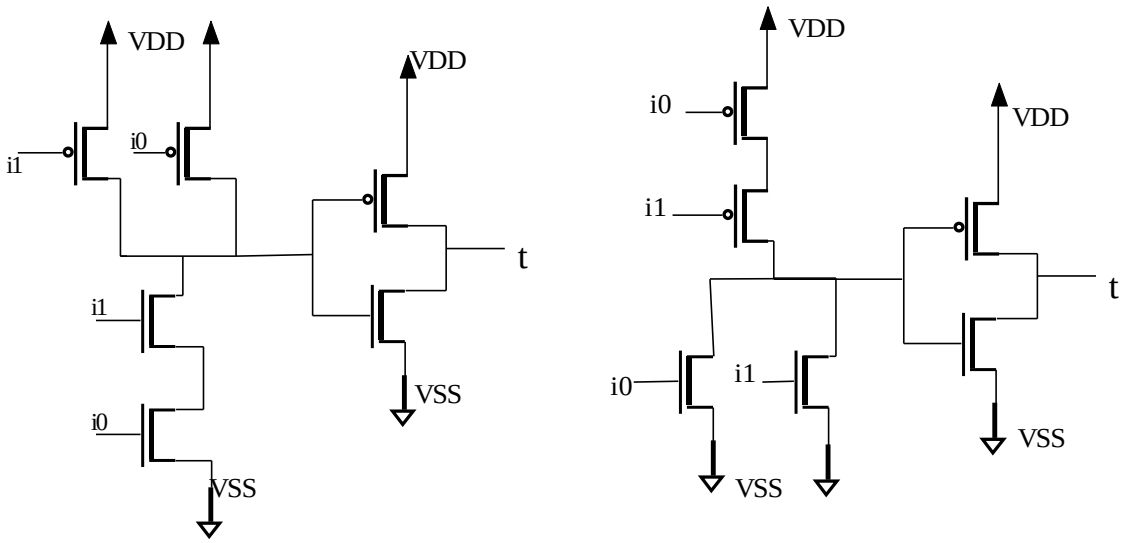
包括与、或、非、与非、或非、异或等，其中非、与非和或非是构造其它类型电路的基本逻辑门，如图 5-2 所示。



(a)NAND2　　(b)NOR2　　(c)NOT

图 5-2 CMOS 基本门

由图 5-2 中的这些基本门可以组成其他各种基本电路，如图 5-3 所示。图 5-3 中(a)是由一个二输入与非门和一个非门串联组成的二输入与门，(b)是由一个二输入或非门和一个非门组成的二输入或门。



(a)AND2　　(b)OR2

图 5-3 由基本门构成的 AND2 和 OR2

因此，如果在处理的主电路中包含了如与门和或门等这些由基本门组合而成

的复合电路，那么在识别中会将它随机地识别成一个复合结构或多个基本门相连的结构。例如，我们可能把一个二输入的或门（如图 5-3（b））抽取成一个或非门和一个非门串联的结构，也可能直接把它抽取为一个单独的或门。所以，这种随机的处理方式其实很不合理，得到的门级模型也不是优化的形式。所以，在今后系统的改进中要在识别过程中加入局部最优策略，使得每个节点的匹配都是最理想的匹配。

第六章　全文总结

随着 VLSI 的发展，电路日益增长的复杂性给计算机辅助设计 CAD 工具带来巨大挑战的同时，电路的层次化描述也愈发重要。自动地从电路的晶体管级网表得到一个较高层次的描述如门级描述在 VLSI 设计中起着非常重要的作用。抽取得到的门级模型即可用于对电路进行功能仿真，也直接套用 Stuck-at-fault 模型，尤其在在形式化验证的等价性检验中，门级模型的自动抽取也扮演着非常重要的角色，这使得对晶体管电路门级模型的抽取系统的设计显得非常重要。

本课题就是基于“巨微”等价性验证系统中抽取器的开发要求，针对基于子图同构的晶体管级电路门级模型抽取系统的设计实现展开工作。主要的研究成果总结如下：

1、对晶体管级电路门级模型的抽取方法进行了研究总结，对实现该方法的 SubGemini 算法和 DECIDE 算法进行了分析比较，由于 SubGemini 算法在第一阶段采用的划分预处理方式会消耗大量的时间，而且划分算法的选取直接影响整个识别算法的效率，所以最终确定 DECIDE 算法为本系统的主要算法，并对该算法做了一些启发式的改进和实验。

2、针对系统设计中出现的技术难点进行了分析并提出了解决方法，使得系统能够避免子图同构的递归判定函数导致的错误，解决了对带环状结构电路的识别以及多匹配问题。

3、对提出的新方案进行了详细分析，设计实现了系统中的每个功能模块。首先，根据方案的系统设计，给出了系统的输入处理模块的设计实现，使用动态存储管理机制对输入网表中的信息进行存储，并进行电气连接上的检查。然后，给出了整个系统的核心模块——抽取模块的设计实现，采用了一种递归式的处理方法。最后，给出了系统输出模块的设计实现，分析和确定了门级模型输出的文件格式。最终，对程序进行了编译、链接、运行和调试。

4、对实现的晶体管电路门级模型抽取系统的软件进行了测试，并对实验结果进行了分析。实验表明，通过权值比较对主电路的节点进行筛选后，可大大减少需要进行匹配的节点个数，而且对同一子电路而言，识别所需时间基本上与候选节点的个数呈线性关系。

本文对晶体管级电路门级模型的抽取系统的设计与实现工作进行了一定深度

的研究，取得了阶段性的成果。但是，本文所实现的系统还只是初级实验阶段，离实际的商业应用还有较大距离，必须在本文所取得的成果的基础上，继续进行更深一步的研究开发。后续工作应主要集中在如下几个方面：实现对更大规模的集成电路的处理，提高处理的效率；实现对时序单元的功能抽取；完善子电路库中电路的类型和规模；实现对更多常见网表类型的处理，而不仅局限于对 spice 网表和 al 网表的处理等。继续研究和改进本文中所实现的晶体管级电路门级模型的抽取系统，使其更加完善，需要我们做出不懈的努力。

致谢

首先，作者要感谢指导老师李绍荣副教授，他无论在作者研究生学习期间还是参与课题期间都给予了极大的帮助和支持。李老师精深的学术造诣、严谨的治学风范、一丝不苟的敬业精神都给我留下了极其深刻的印象，给我留下了为人立业的榜样。在此，谨李老师致以深深的敬意和诚挚的感谢！

同时，还要感谢吴尽昭教授、成都集成电路数字形式验证工程技术研究中心的博士孙秀丽和博士生高新岩在我学习和课题完成的无私帮助，与他们的相处交流是我生活与学习上的宝贵财富。

然后，要感谢成都集成电路数字形式验证工程技术研究中心的闫伟师兄、何安平同学、周宁同学、岳园同学，和他们合作的非常顺利，他们在项目中体现的认真负责的精神让人难忘。

最后，感谢我的父母，他们在我学习、工作和生活中给予我的支持、理解、关怀和照顾让我终生不忘。

正是得到以上这许多老师、同学、亲人和朋友的帮助、支持和理解，本文才得以顺利完成。

参考文献

[1] John P. Uyemura. 超大规模集成电路与系统导论（周德润译）. 北京：电子工业出版社，2005

[2] 潘中良. 数字电路的仿真与验证. 北京：国防工业出版社，2006

[3] S. Jolly, A. Parashkevov, T. McDougall. Automated equivalence checking of switch level circuits. Proc. IEEE/ACM Design Automation Conference, 2002: 299-304

[4] 韩俊刚，杜慧敏.数字硬件的形式化验证. 北京：北京大学出版社，2001

[5] Christoph Kern, Mark R.Greenstreet. Formal verification in hardware design: a survey. ACM Transactions on Design Automation of Electronic Systems, 1994: 123-193

[6] K.L.McMillan. Symbolic Modeling Checking.Kluwer Academic Publisher, Boston, 1994

[7] D.A.Cyrluk, M.K.Srivas.Theory Proving: Not an Esoteric Diversion but the Unifying Framework for Industrial Verification, ICCD'95:538-544

[8] R. E. Bryant. A switch level model and simulator for the MOS digital systems, IEEE Trans Comput, vol. C-33:160-177

[9] R. E. Bryant, D. Beatty, K. Cho, et al. COSMOS: A Compiled Simulator for MOS Circuits. 24th DAC, 1987

[10] D.T.Blaaiw, D.G.Saab, P. Banerjee, et al. Functional abstraction of logic gates for switch level simulation. European Conference on Design Automation, 1991:329-333

[11] V.D.Agrawal, S.T.Chakradhar. Combinational ATPG theorems for identifying untestable fault in sequential circuits. IEEE Trans.on Computer-Aided Design, vol.14:1155-1160

[12] I. Parulkar, M. A. Breuer, C. A. Njinda. Extraction of a high-level structural representation from circuit description with applications to DFT/BIST. Proc.IEEE/ACM Design Automation Conference, 1994:345-350

[13] R.E.Bryant. Extraction of gate level models from transistor circuits by four valued symbolic analysis.International Conference in Computer-Aided Design, 1991:350-353

[14] M.S.Abadir, J. Ferguson.An improved layout verification algorithm (LAVA).Proc.European Design Automation Conference. 1990:391-395

[15] Sandip. Kundu.GateMaker: A transistor to gate level model extractor for simulation, automatic test pattern generation and verification.Proc.of International Test Conference, 1998:372-381

[16] Michael Boehner. LOGEX – an automatic logic extractor from transistor to gate level MOS technology.Proc.of the 25th DAC, 1988:517-522

[17] M.Laurentin, A.Greiner, R.Marbot. DESB, a functional abstractor for CMOS VLSI. IEEE European Design Automation Conference, 1992

[18] G.Pelz, U.Roettcher. Pattern matching and refinement hybrid approach to circuit comparison.1994 IEEE Transactions on Computer-Aided Design, vol. 13:264-275

[19] L.Yang, C-J.R.Shi. FROSTY: a program for fast extraction of high-level structural representation from circuit description for industrial CMOS circuits.Integration, the VLSI Journal. 2005, vol. 39

[20] Nian Zhang, Donald C.Wunsch II, Frank Harary. The subcircuit extraction problem.IEEE POTENTIALS.AUGUST/SEPTEMBER 2003:22-25

[21] T. Kostelijk, B. De Loore. Automatic verification of library-based IC designs. IEEE Journal of Solid-State Circuits, March 1991

[22] Vincent Remie. Bachelors Project: Graph isomorphism problem. Eindhoven University of Technology Department of Industrial Applied Mathematics, 2003

[23] A. Lester, P. Bazargan-Sabet, A. Greiner.YAGLE, a second generation functional abstractor for CMOS VLSI circuits. Proc. of the Tenth International Conference on Microelectronics, 1998: 5-268

[24] Conformal™ LEC Logic Equivalence Checker and Transformal™ LTX Logic Transistor Extraction User Manual. Version 3.0 .August 2001

[25] www.cadence.com/products/functional_ver/conformal/index.aspx

[26] Jack Whitham. A Graph Matching Search Algorithm for an Electronic Circuit Repository，2003-2004

[27] Thomas Stephen Chanak. Nelist Processing for cunstom VLSI via pattern matching.1995

[28] Miles Ohlrich, Carl Ebeking, Eka Ginting, et al. SubGemini: identifying subcircuits using a fast subgraph isomorphism algorithm.30th ACM/IEEE Design Automation Conference,1993: 31-37

[29] I. Parulkar, M. A. Breuer, C. A. Njinda.Extraction of a high-level structural representation from circuit description with applications to DFT/BIST.Proc.IEEE/ACM Design Automation Conference,1994:345-350

[30] K.-T. Huang, D. Overhauser. A novel graph algorithm for circuit recognition. Proc. IEEE International Symposium on Circuits and Systems. 1995:1695-1698

[31] N. Vijaykrishnan, N. Ranganathan. SUBGEN: a genetic approach for subcircuit Extraction. 9th International Conference on VLSI Design, 1996:343-345

[32] Wei-Hsin Chang, Shuenn-Der Tzeng, Chen-Yi Lee. A novel subcircuit extraction algorithm by recursive identification scheme. Circuits and Systems, 2001

[33] Nikolay. Rubanov. SubIslands: the probabilistic match assignment algorithm for subcircuit recognition. 2003 IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 22:26-38

[34] Zong Ling, David Y. Yun. An efficient subcircuit extraction algorithm by resource management. 2nd International Conference on ASIC, 1996:9-14

[35] Scott W. Hadley, Brian L. Mark, Anthony Vannelli. An efficient eigenvertor approach of finding netlist partitions. 1992 IEEE Transaction on Computer-Aided Design, vol. 11

[36] Georg Pelz, Uli Roettcher. Circuit comparison by hierarchical pattem matching. Intemational Conference on Computer-Aided Design, 1991:290-293

[37] Kai-Ti Huang, David Overhauser. A novel graph algorithm for circuit recognition. Circuits and Systems, 1994, 3: 1695-1698

[38] 李玉波，朱自强，郭军. LINUX C 编程.北京：清华大学出版社，2005

[39] Brain W.Kernighan, Dennis M.Ritchie. C 程序语言设计(徐宝文，李志译). 北京：机械工业出版社，2004

[40] Kris Jamsa, Lars Klander. C/C++程序员实用大全——C/C++最佳编程指南. 北京：中国水利水电出版社，2001

[41] 王志功，沈永朝.集成电路设计基础. 北京：电子工业出版社，2004

[42] A. Greiner, F. Pécheux. Alliance: a complete set of CAD tools for teaching digital VLSI design. 3rd Eurochip Workshop on VLSI Design Training, 1992:230-237

[43] 侯伯亨，顾新. VHDL 硬件描述语言与数字逻辑电路设计. 西安：西安电子科技大学出版社，1997

[44] 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，2004

[45] 严蔚敏，吴伟民. 数据结构. 北京：清华大学出版社，2002

攻硕期间取得的研究成果

[1] 徐玉婷，李绍荣，吴尽昭，高新岩. 一种子电路识别的启发式算法. 全国第 14 届计算机辅助设计与图形学学术会议，2006：121-126

[2] 李绍荣，徐玉婷. 基于 BDD 的组合电路等价性验证方法研究. 计算机科学，2007，34（3）：293-295

基于子图同构的晶体管级电路门级模型抽取的研究

作者：徐玉婷
学位授予单位：电子科技大学