申请上海交通大学工程硕士学位论文

模拟电路符号化仿真器（GRASS）图形化用户界面开发与电路设计案例[1]

学　　校： 上海交通大学

院　　系： 微电子学院

班　　级： Z0721091

学　　号： 1072109041

工程硕士生： 李骥

工程领域： 软件工程

导　　师： 施国勇（教授）

上海交通大学微电子学院

2009 年 11 月

[1] 此研究由上海市浦江人才基金(项目编号 07pj14053)和国家自然科学基金(项目编号 60876089)资助。

A Dissertation Submitted to Shanghai Jiao Tong University for

Master Degree of Engineering

DEVELOPMENT OF A GRAPHICAL USER INTERFACE FOR AN ANALOG SYMBOLIC SIMULATOR (GRASS) WITH DESIGN APPLICATIONS

Author: Li, Ji

Specialty: Software Engineering

Advisor: Prof. Shi, Guoyong

School of Microelectronics

Shanghai Jiao Tong University

Shanghai, P.R. China

November, 2009

模拟电路符号化仿真器（GRASS）图形化用户界面开发与电路设计案例

摘 要

符号化模拟电路仿真器可以快速导出模拟电路频域指标关于每个电路参数的解析表达式。本文介绍了特别为这样一种符号化仿真器开发的图形界面。它以图形的方式实时直观地呈现出各种电路设计指标关于多个电路参数的灵敏度，能让设计者直观地看到电路设计指标关于电路参数依赖关系的整体图像，从而快速准确地选择合适的参数组合以达到最佳设计指标，突破了传统数值电路仿真工具在多参数协同优化应用中的局限性。一个带图形界面的符号化仿真工具可应用在以提高电路可靠性和生产良率以及各种频域性能指标为目的的模拟集成电路设计中。

关键词：图形化用户界面，GRASS，GTK+，OpenGL，OpenMP

Development of a Graphical User Interface for an Analog Symbolic Simulator (GRASS) with Design Applications

ABSTRACT

A symbolic analog circuit simulator efficiently generates an analytic frequency response function in terms of all parameters in a linear circuit. This thesis presents the development and application of a graphical user interface (GUI) using OpenGL for a recently developed symbolic simulator, GRASS. It is demonstrated that by displaying the performance dependence on multiple parameters in a 3D surface, the designer can interactively select the parameter values satisfying the performance qualification meanwhile gain an intuitive confidence on the sensitivity property coming with the parameter selection. The GUI supported symbolic simulation tool provides the analog designers a new approach to robust design for achieving high yield.

Keywords: GUI，GRASS，GTK+，OpenGL，OpenMP

目 录

第 1 章 引言

1.1 符号化仿真器的研究背景

自从功能强大的计算机得到普及以来，电子工程师们就开始开发各种应用软件试图让计算机完全自动化地执行系统分析任务。今天所有的电子工程师和相关领域的学生们都在使用功能强大的电路仿真器（如 HSPICE，ELDO 和 SPECTRE 等）。目前工业界主要使用基于加州大学伯克利分校开发的 SPICE[1]作为电路仿真的标准。尽管如此，现有的电路仿真器并未覆盖到集成电路设计中的所有需求。从本质上看现有的电路仿真器只能对已知大小的电路行为进行验证，而通过研究性能曲线和设计参数之间的关系，设计者必须能够预测规模未知的电路的行为。这些曲线与设计参数之间的关系通常包括转输函数、零极点、根轨迹、参数的波特图、失谐系数等等。目前这些关系的导出主要还是靠手工完成。能自动完成这种推导过程的工具被称为符号化仿真器。

第一代符号化仿真器出现在 20 世纪 60 年代，当时只提出了一些最基本的分析技术。这些工具未能在设计师群体中得以广泛应用，一部分原因是昂贵的计算成本和脱离了电路设计师的真实需求。到 20 世纪 70 年代，由于人们普遍缺乏定对制模拟集成电路的兴趣致使符号化仿真器的关注度远低于数值仿真器。不过这一时期仍然有符号仿真器的研究成果产生，其中的代表是 SNAP[2]和 NAPP[3]。随着定制模拟和混合信号 ASIC 的发展前景越发光明，这种情况在 80 年代期间得到改变。人们逐渐意识到符号化仿真器的强大之处。符号化模拟电路仿真器在最佳拓扑结构选择、设计空间拓展、行为模型产生以及故障检测等方面较数值型仿真器有很大的优势[4]。它可以让模拟电路设计者从漫长的电路功能分析中解脱出来，在建模和设计流程中需要大量重复计算的地方显示其强大的功能。这些电路设计中的需求都是推动随后符号化仿真器发现的动力。这段时期内出现的新一代符号化仿真器的主要特点是符号近似、表达式标识处理以及对弱非线性电路的扩展分析，主要代表有 ISAAC[5][6]，ASAP[7][8]，SYNAP[9][10]，SAPEC[11]，SSPICE[12]，SCYMBAL[13]，SCAPP[14]和 GASCAP[15]。

1.2 符号化仿真器的最新研究情况及展望

进入 21 世纪以来，随着计算机性能的大幅提升以及一批高效算法的提出，同时模拟集成电路的复杂度迅速增加，符号化仿真器又重新为人们所重视。以前一些受限于计算机性能和计算成本的符号化分析算法在拥有最新 CPU 和低成本大容量内存的 PC 也能顺利实现。

目前的符号化仿真器从基本理论上主要分为两大类：代数分析方法，拓扑分析方法。代数方法主要通过建立和求解电路方程得到最终的分析结果。例如用克莱姆法则通过求解矩阵行列式来计算电路方程。拓扑方法把电路看作一张有向图或一个网络，通过特定的操作可以得到用某种数据结构存放的符号化分析结果，例如枚举电路中的基本回路或是生成树。

基于以上两种理论，现有的符号化仿真器的具体算法大体上可分为五种：矩阵行列式法[16]、符号流图法[17]、生成树枚举法[18]、参数提取法[19]和数值插入法[20]。矩阵行列式法通过直接求解（如高斯消元法或拉普拉斯展开法等）符号化的线性方程组获解；符号流图法的本质是梅森公式，它通过明确定义的法则来寻找表征电路方程中各项所对应的路径和回路获解；生成树枚举法通过枚举电路图中的生成树可以求得计算结果中的每一个乘积项，其中电路图的每个分去都有相应的权重；参数提取法仍然基于电路方程组所对应的行列式，通过递归分解行列式，例如每次把行列式分解成包含所需提取参数的部分和不包含所需参数的部分来得到最终的结果；数值插入法基于电路某些工作点的数值分析结果进行分析。显然矩阵行列式法和参数提取法为代数型分析法，符号流图法和生成树枚举法是基于拓扑分析的方法。当前被广泛采用的主要是矩阵行列式法和符号流图法。上海交通大学微电子学院 EDA 实验室 2006 年开发的符号化仿真器[21]成功解决了生成树枚举算法在处理各种类型受控源时遇到的难题，采用生成树枚举法达到了与国外同时期最先进的矩阵行列式同等级的电路符号化处理能力。

最近由于多核 CPU 成为主流，可以充分发挥多核共享内存并行计算优势的算法研究如雨后春笋般涌现出来，将多核并行计算应用于特定领域实现高效的计算成为很多算法研究的趋势。利用多核 CPU 上的并行计算来提高符号化仿真器性能也是大势所趋。

1.3 符号化模拟电路仿真器与数值仿真器的对比

符号化分析的优势大致有以下几点：

(1) 揭示电路行为的本质特征。符号化的分析法可以自动地提供电路特性符号化表达式，这些表达式不会随着电路参数的变化而改变，通过带入具体的数值或者忽略数值较小的部分来简化表达式都可以很快且准确的得到具体电路的计算结果。

(2) 在电路表达式的推算过程中不存在舍入误差和收敛性问题，可以提高设计的可靠性和设计精度。

(3) 电路分析模型的产生和电路尺寸的自动最优化。符号化仿真器可以产生描述电路交流特性的分析模型，从而为电路尺寸调节提供依据。

(4) 电路结构的交互拓展。电路拓扑结构调整后，符号化仿真器可以快速的产生新的电路特性表达式，交互式的界面可以便捷地绘制不同的电路结构，为电路结构的自动化调整提供基础。

(5) 电路参数的迭代调整。电路结构不变时，符号化的分析结果不会发生改变。模拟电路设计常常进行多次电路参数迭代来获得理想的性能，符号化仿真器只需将不同的参数值代入电路表达式求值，而无须重新执行分析过程。

(6) 自动生成运算放大器、滤波器等模拟电路模块的行为模型。无需使用具体器件来搭建这些模块，为快速进行电路系统的行为分析提供便利。

(7) 辅助模拟可测性分析和故障检测。

与数值仿真器的比较：

数值分析法的结果是描述电路特性的数据或者图表。由于数值分析理论的成熟发展，数值分析法的仿真速度比较快。数值分析法主要可以用来验证电路的功能特性，检验电路是否符合设计的要求。

数值分析法的缺点在于数值分析法无法非常直观地揭示哪个电路元件对电路性能起决定作用，使用数值分析法需要通过多次实验以及设计经验来找出关键元件；其二，正因为数值分析法提供的结果不是形式化的结果，所以无法自动化地提供给设计者相关的设计修改建议和电路所存在的设计问题；第三，就是每当电路中某个元件的数值发生变化，就必须重新运行数值分析法重新进行仿真。

符号化分析法的分析结果是以符号形式表示的电路特性表达式。一般而言，由于符号化电路分析方法分析的对象是线性电路，所以符号化分析主要面向电路的交流

（AC） 特性。只要电路的结构不发生变化，符号化表达式就保持不变。换句话说，即使电路元件参数值发生变化，符号化分析的结果也并不会发生变化。基于这个特性，符号化分析方法能给模拟电路设计提供很多的便捷，因为模拟电路设计通常需要调节电路元件的参数而不是电路的结构，这样对于符号化的表达式，只需代入不同的数值进行计算就可，而不需要重新进行符号化分析。

目前的符号化仿真器需要解决的难题有：现有的分析方法的运算效率仍然比较低；符号化分析的结果——符号化电路表达式，由一系列符号化的生成项组成，而对于符号化的分析方法来说，这些生成项的数目（算法的空间复杂度）在最坏情况下会随着电路尺寸（电路元件个数）的增加而呈指数级增长，这对于这些项的生成和存储都是巨大的挑战；由于求出生成项的算法在最坏情况下具有指数级增长的特点，适用于符号化分析的电路的尺寸受到了一定限制。

在传统模拟电路设计中，模拟电路设计者必须对针对不同的电路参数进行大量的数值仿真以确保达到所有设计指标。对于模拟集成电路的设计，由于芯片生产工艺不可避免地受到随机扰动因素的影响，流片得到的器件参数和其标称值之间总存在着随机的偏差，一定比例的芯片无法达到设计指标的要求而被认定为次品。为了改善生产良率，由于缺乏适当的工具展示良率与电路设计的直观关系，电路设计者常常不得不以保守的架构和参数选择、增大的芯片面积和功耗等为代价来保证芯片的可靠性和可允许的生产良率，但导致电路性能下降。而事实上，如果有更好的工具支持，完全有可能通过选择更佳的电路参数组合和低敏感性设计来实现高可靠性和高良率的设计产品，同时降低设计和生产成本。

SPICE 等数值仿真器提供了两类容差分析方法来解决以上问题。一类是以灵敏度分析为基础的方法，例如最坏情况分析（Worst Case）。另一类是统计抽样方法，例如蒙特卡洛分析（Monte Carlo）。最坏情况分析虽然需要的仿真次数不多，但不考虑电路参数的相关性，所以是一种保守的设计方法，对于以可靠性和良率为目的的芯片设计通常不是最经济最有效的方案。蒙特卡洛分析本质上是对电路性能的统计分布规律的估计。由于要进行成千上万次仿真才能得到比较可靠的数据，所以蒙特卡洛分析的计算成本很高，计算时间很长。特别是频域蒙特卡洛分析，对于每一个抽样电路在每一个频率点都要进行一次完整的电路仿真，并存储历史数据，计算的时间和存储空间消耗都非常高。另外，实际电路参数变化的统计分布规律本身很难准确估计，对于不同的电路设计方案和生产条件都必须事先选择符合实际的统计模型，这也在客观上增加了蒙特卡洛分析的难度。蒙特卡洛的另一局限性是无法容易的得到敏感度信息，除了能在一定程度验证设计的有效性外，对于电路的性能优化所能提供的帮助有限。

事实上对于有经验的模拟电路设计者而言，根据设计需求确定电路的主体结构并不是主要困难，他们的大多数精力往往花费在对极少关键参数的协同优化上。但现有的主流数值仿真器，都没有提供多参数协同优化的功能，设计者大多以试错(trial-and-error)的方式从事设计优化，难以直接对设计的最终优化程度做出判断。

1.4 符号化模拟电路仿真器 GRASS 简介

近期由上海交通大学微电子学院 EDA 实验室开发的基于图约化算法的符号化模拟电路仿真器（GRASS）采用高效的数据结构和算法[21]，能精确地求出较大规模模拟电路传输函数的符号化解析表达式。从解析形式的传输函数可以方便地求得各种参数化频域指标，如相位裕度、带宽、直流增益等与任何一组电路参数间的解析对应关系。对于某一特定电路，这种设计指标关于电路参数的解析关系只要建立一次。随之，任何电路参数的变化而导致的电路性能的变化都可以对内存中的解析表达式直接求直得到，速度极快，避免了反复进行数值仿真的麻烦。在这基础上，通过友好的图形化用户界面支持，设计者可以通过交互方式选择重要设计参数，直接观察由于所选设计参数变化而带来的性能指标以三维图形展示的变化，从根本上改变了传统数值仿真器以点点仿真所能得到的局部设计，让设计者从电路参数的整体变化来把握电路的全局可靠性，从而提高了设计可靠性和优化程度。

1.5 本章小结

本章介绍了符号化模拟电路仿真器的若干年来的研究背景。符号化分析方法已经有较长的历史，相比与数值型电路仿真器更适宜于对模拟电路深层次的分析。现在已经有相对较为成熟的符号化模拟电路仿真器（如 GRASS），但没有能配套使用的交互式图形化用户界面提供给用户，这使得符号化模拟电路仿真器在实际电路设计中的应用受到一定的制约。

本文详细介绍了针对一个符号化模拟电路仿真器 GRASS 开发的一套跨平台交互式图形化用户界面。这套图形化用户界面力图将符号化模拟电路仿真器的易用性和强大功能用友好的方式提供给用户，对符号化模拟电路仿真器在实际电路设计中的应用进行了初步的探索。本文第二章介绍了 GRASS 的基本原理，第三章详细介绍交互式图形化用户界面（GUI）的设计过程和实现方法，第四章介绍了该用户界面结合符号化模拟电路仿真器在实际电路设计案例中的应用，第五章对全文进行总结。

第 2 章　符号化模拟电路仿真器的基本原理

本章将从理论上介绍符号化模拟电路仿真器的基本原理。

2.1 符号化电路分析思想简介

电路级的符号化分析法是一种形式化的分析方法。符号化分析研究主要针对线性电路的频域分析。对于一个集中总的线性时不变电路，对于已知输入相应输出的传输函数可以表示成两个多项式相除的形式，其自变量为 x，如式(2.1)所示：

$$H(x)=\frac{N(x;p_1,\cdots,p_m)}{D(x;p_1,\cdots,p_m)}=\frac{\sum_i x^i a_i(p_1,\cdots,p_m)}{\sum_i x^i b_i(p_1,\cdots,p_m)}, \tag{2.1}$$

其中 x 是复数频率变量，对应于离散时域电路的 z 变量或连续时域电路的 s 变量；$p_1,\cdots,p_m$ 是电路参数，即元件的各种参数。$a_i(p_1,\cdots,p_m)$ 和 $b_i(p_1,\cdots,p_m)$ 都是由电路参数组成的多项式。

图 2-1 是一个模拟滤波器的电路图，

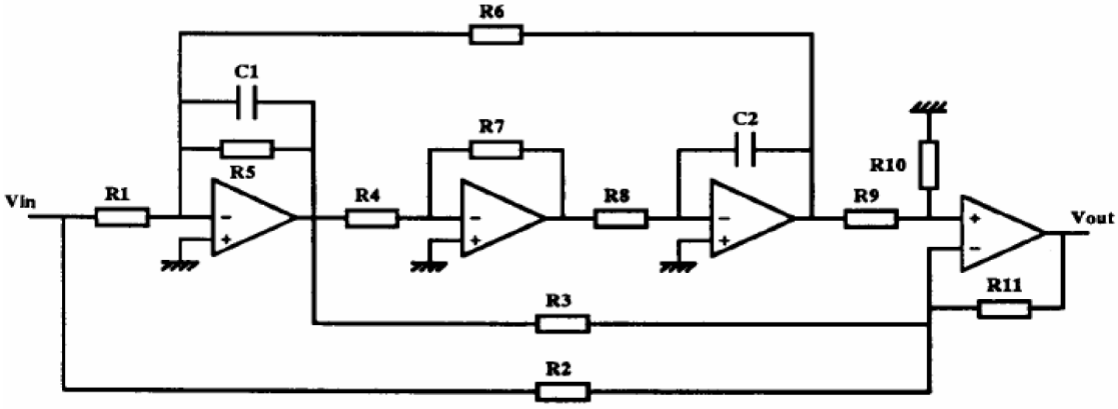


图 2-1　一个模拟滤波器电路图

Fig. 2-1　An analog filter schematic

符号化电路仿真得到图 2-1 所示电路的传输函数符号表达式(2.2)：

$$
\begin{aligned}
H(s) = & \\
& [-G_4G_8\left(G_1G_2G_9 + G_1G_3G_9 + G_1G_9G_{11} + G_2G_6G_9 + G_2G_6G_{10}\right) \\
& +sG_2G_7\left(G_1G_3G_9 + G_1G_3G_{10} - G_2G_5G_9 - G_2G_5G_{10}\right) - s^2G_2G_7C_1C_2\left(G_9 + G_{10}\right)] \\
& \Big/\Big[G_{11}\left(G_9 + G_{10}\right)\left(G_4G_6G_8 + sG_5G_7G_2 + s^2G_1G_2G_7\right)\Big].
\end{aligned} \tag{2.2}
$$

2.2 二分判定图简介

二分判定图由最初由 Aker 在[22]中提出，目的是为了将布尔函数转换成判定图，Lee 在[17]中首先提出了一个直线化的转换程序。到了 1983 年，Byrant 将二分判定图概念有效地使用到门级电路的研究领域，提出了简化有序的二分判定图（Reduced Ordered Binary Decision Diagram，ROBDD，往往被简称为 BDD）[23]的数据结构并发明了基于这个数据结构的布尔函数操作的算法[24][25]。图 2-2 是与布尔函数 $F = x_0x_1 + x_2$ 对应的一个二分判定图：

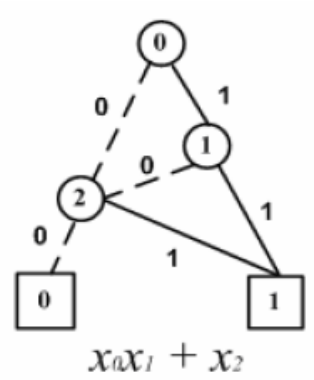


图 2-2　一个布尔函数的二分判定图结构

Fig. 2-2　A BDD structure of a Boolean function

目前二分判定图在数字集成电路计算机辅助设计工具中被广泛地使用，应用的领域包括电路行为验证、测试向量生成、故障仿真和逻辑综合[26]。二分判定图本来应用于数字逻辑表达式的简化以及二元项的存储，它可以非常有效和精简地表示一些与子集相关的问题。GRASS 在寻找电路拓扑结构的生成树[27]时就是利用二分判定图作为算法实现的基础，并利用其高度的空间压缩性充分控制算法的空间复杂度。

2.3 符号化模拟电路仿真器 GRASS 的基本原理

GRASS[21]是近期由上海交通大学微电子学院 EDA 实验室开发的基于图约化算法的符号化模拟电路仿真器。它采用了紧凑的数据结构 BDD 和高效的有向图匹配算法，能精确地求出较大规模模拟电路传输函数的符号化解析表达式。

GRASS 和 DDD[27]是目前世界上极少数能精确分析较大规模模拟集成电路（约 30 个晶体管）频域解的符号化仿真器。与 DDD 利用电路改进节点法方程进行纯代数求解不同，GRASS 按照一套固定的规则对电路的拓扑结构以图的形式进行逐步约化，在约化过程完成时就得到了整个电路的传输函数[28]。由于 GRASS 采用了基于二分判定图（Binary Decision Diagram, BDD）的紧凑共享型数据结构来存储整个电路的传输函数，并且约化过程中的每一步只处理一个电路参数（元件），所以能有效地帮助设计者深入地剖析电路行为与一个或一组电路参数间的关系，快速定位到影响电路行为的关键参数，并针对其进行优化。

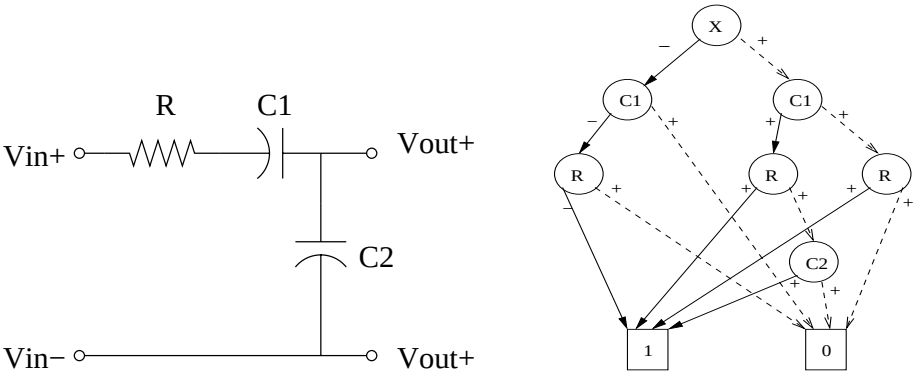


图 2-3 (a) 一个简单的 RC 电路网络　(b) 电路网络(a)的符号判定图

Fig. 2-3 (a) A simple RC network　(b) The SDD of (a)

以图 2-3a 所示电路网络为例，GRASS 读取 SPICE 格式的网表文件后进行符号化分析，将得到图 2-3b 所示的符号判定图（Symbol Decision Diagram, SDD）。SDD 等价于图 2-3a 所示电路网络的传输函数，本质上是以 $ax+b$ 形式嵌套的多项式(2.3)：

$$-\left(-\left(-R^{-1}+0\right)sC_1+0\right)X(s)+\left(\left(R^{-1}+\left(C_2+0\right)\right)sC_1+\left(R^{-1}+0\right)\right) \qquad (2.3)$$

图 2 中的实箭头表示乘法，虚箭头表示加法，正负号表示箭头指向的子多项式的符号。由基本电路定律可以证明[28]对应于图 2-3a 电路网络的多项式(2.3)恒为 0，且满足 $H(s) = 1 / X(s)$, 其中 $H(s)$是电路网络的传输函数，所以

$$H(s)=\frac{1}{X(s)}=\frac{C_1sR^{-1}}{(C_1+C_2)R^{-1}s+C_1C_2s^2} \qquad (2.4)$$

于是只经过一次符号化分析，GRASS 就能得到形如图 2-3b 所示的电路网络符号化传输函数的等价表示，再反复利用该结构就能对相位裕度、开环增益、带宽、频域灵敏度[29]等电路性能指标进行快速准确的符号化分析。传统的数值仿真器要计算某个参数在特定范围内的频域指标，则必须反复进行 AC 分析，而且得到的数据没有连续性，直观性也较差。对于多个参数的同步分析，即使对于最简单也是最常见的两个参数的情形，由于一般情况下要得到可靠结论所需的采样数目较大，传统的数值仿真器也难以在短时间内给出有效且直观的分析结果。而 GRASS 能重复利用一次仿真得到的传输函数解析表达式，快速地计算出每一组参数值对应的分析结果供设计者参考。这是 GRASS 优于传统数值仿真器之处。

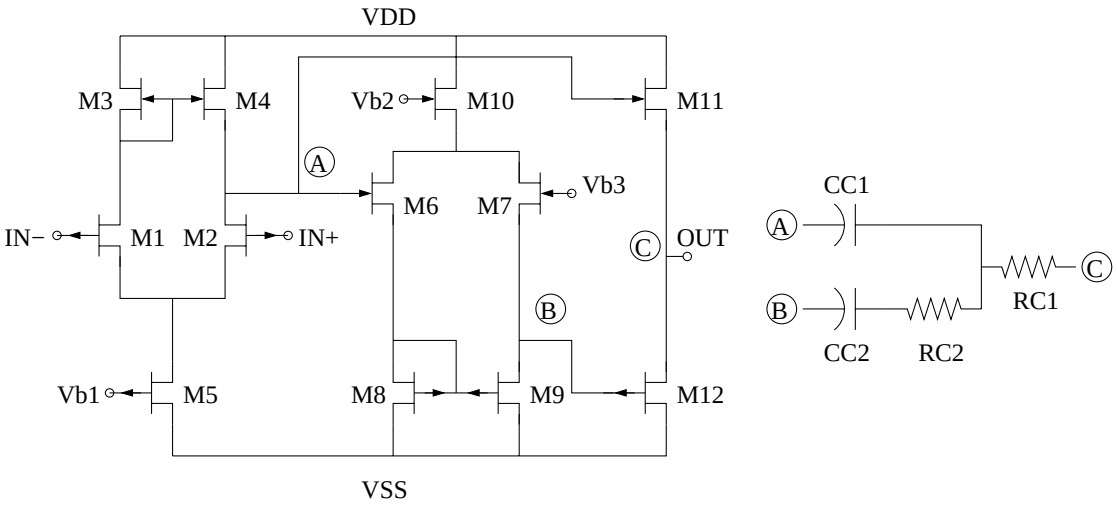


图 2-4　一个含 RC 反馈网络的 3 级运算放大器电路[30]

Fig. 2-4　A 3-stage operational amplifier with an RC feedback network[30]

对于类似于图 2-4 的含晶体管模拟集成电路，GRASS 的符号化仿真过程如下：

1) 利用数值方法计算电路的静态工作点；
2) 用图 2-5 所示的 SPICE level 3 线性小信号电路模型替换电路中的全部晶体管；

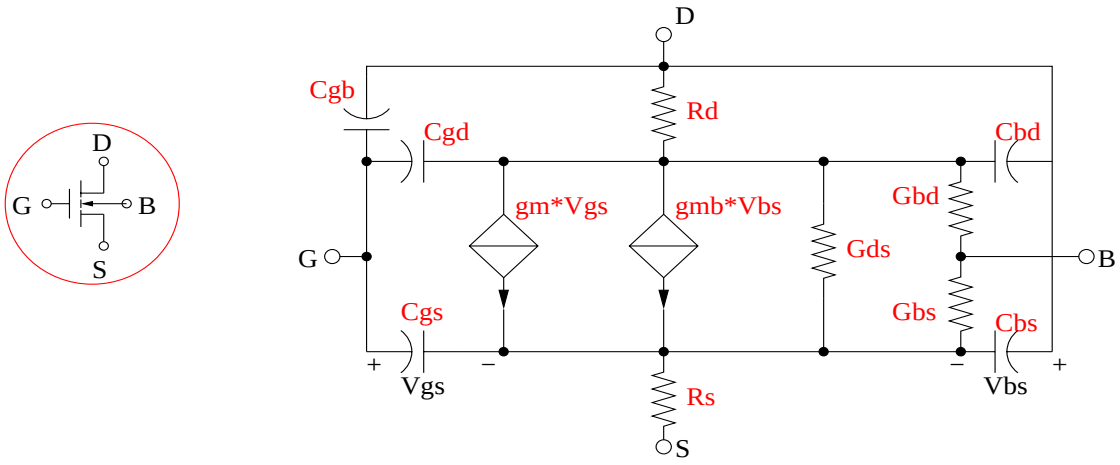


图 2-5　一个 NMOS 及其 SPICE level 3 线性小信号模型

Fig. 2-5　An NMOS and its SPICE level 3 linear small signal model

3) 根据第 1 步确定的静态工作点计算出每个小信号模型的内部元件值；

4) 对整块线性化等效电路进行符号化分析，得到传输函数的解析表达式 $H(s; c_1, c_2, \ldots, c_n)$，其中 c_i，$i = 1, 2, \ldots, n$ 表示所有的电路参数；

5) 利用 $H(s; c_1, c_2, \ldots, c_n)$计算所需的各种频域电路指标；

例如，相位裕度与电路参数 c_1, c_2 的关系可以用式(2.5)所示的二元函数来计算

$$PM(c_1, c_2) = \arg\left(H\left(j\omega_0; c_1, c_2, c_3^0, \cdots, c_n^0\right)\right) + 180° \qquad (2.5)$$

其中 ω_0 为系统增益$|H(s; c_1, c_2, \ldots, c_n)|$为 1 时输入信号的最低频率，$c_3^0, \cdots, c_n^0$ 为其余不随 c_1, c_2 改变的电路参数。

2.4 本章小结

本章介绍了符号化仿真器的理论基础和核心数据结构（BDD），并着重介绍了上海交通大学微电子学院 EDA 实验室开发的基于图约化算法的符号化模拟电路仿真器 GRASS 的基本原理和工作流程，展示了 GRASS 在模拟电路频域分析中的强大功能。由于 GRASS 符号化分析结果的复杂性，拥有一个友好的用户界面自然成为 GRASS 应用中首先需要解决的问题。

第 3 章 交互式图形化用户界面的设计与实现

本章将详细介绍我们的交互式图形化用户界面的实现细节。这套图形化用户界面完全采用开源的图形库，可稳定运行于多种操作系统。由于在设计之初就考虑到该用户界面的可扩展性，它拥有一套简单的软件接口，不仅可以与 GRASS 结合使用，也可以方便地应用于其他需要多参数协同分析的场合。

3.1 针对单参数符号化分析的 2 维图形化用户界面

本节主要介绍 2 维图形化用户界面的相关背景和实现细节。

3.1.1 2 维图形界面开发库 GTK+介绍

GTK+（GIMP Tool Kit）[31]是一套跨平台的图形函数库，可以用于建立 X 窗口系统下以图形为基础的应用程序。它最初只作为 GIMP（GNU Image Manipulation Program）[32]的专用开发库，后来发展为 Unix-like 系统下开发图形界面的应用程序的主流开发工具之一，如今已移植至其他平台，诸如 Microsoft Windows、 DirectFB、以及 Mac OS X。它使用 C 语言写成，但是其设计者使用面向对象技术，使其具有良好的可移植性和易用性，无论在 Pidgin[33]这样的小型程序中还是在 GNOME[34]这样的大型程序中都有十分良好的表现。

由于 GTK+遵守 LGPL 协议（GNU Lesser Genera Public License）[35]使得任何人都能参与到 GTK+的多年来开源社区有大批志愿者为 GTK+增加了丰富的功能，并基于此开发出大量的应用程序。从 1998 年的第一个稳定版本发布自今，GTK+总共发布了 11 个主要的稳定版本。目前已得到大量应用的一系列与 GTK+相关的第三方库包括 libgnome、libgnomeui、libgnomeprint、libgnomeprintui、libglade、libgnomecanvas、libegg、libeel 和 gtkglext。其中 libglade[36]是与我们的 2 维图形化用户界面相关的最主要的第三方库。

3.1.2 2 维图形界面设计工具 Glade 介绍

Glade 是针对 GTK+图形函数库的快速图形界面开发的开源工具，囊括了几乎所

有的 GTK+图形模块，如文本框、对话框标签、数字输入、复选框和菜单等控件。使界面设计者只要简单地将控件摆放到 Glade 提供的画布上就能够实现图形界面设计过程“所见即所得”的效果，避免了单纯依赖程序员的想象力来开发图形界面的方式。程序员只需要将精力集中于回调函数的编写和软件的整体架构设计上。这会使图形化用户界面的开发速度更快、质量更易于控制。

Glade 将设计者规划好的图形界面设计布局和回调函数名称存储于 XML 格式的文件中，从而使这些设计可以方便地与外部接口结合。这类 XML 文件与应用程序一起发布，在必要的时候才由应用程序通过 libglade 库动态加载。因为使用了 libglade 库，Glade 生成的 XML 文件能够被 C、C++、Java、Perl、Python 以及 C#等等多种语言支持。由于 XML 文件具有相当规范的格式，增加对其他的语言的支持也是十分方便的。Glade 的这些特点使得图形化用户界面接口本身的开发可以独立于各种计算机编程语言进行。在统一的开发规范下，用 Glade 开发图形化界面的过程可以和其他代码的开发同步进行，这对于程序开发时的项目管理也很有帮助。

该 2 维图形化用户界面就是在 Glade 上设计完成的，以独立的 XML 文本文件格式与主程序一起发布，在运行时动态加载。图 3-1 是 Glade 的设计环境；图 3-2 展示了用 Glade 设计好界面外观后得到的 2 维图形化用户界面的 XML 文件样例（部分，包含主窗口和一个退出按钮），用于动态生成图 3-3 所示的图形化用户界面。

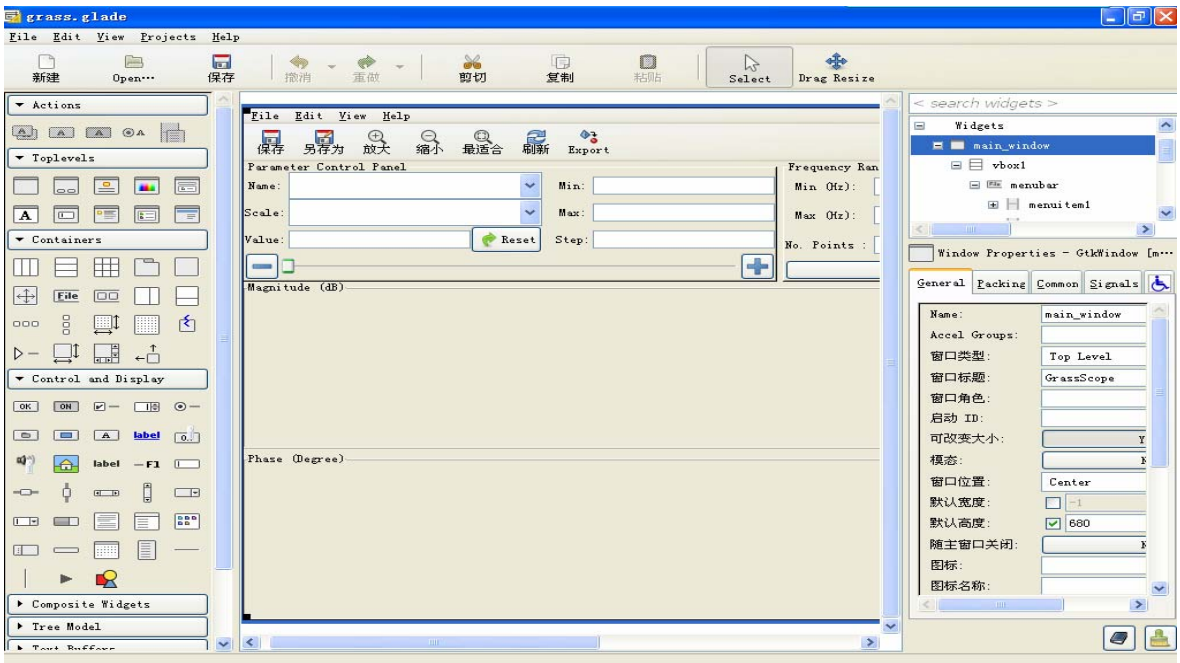


图 3-1 Glade 设计环境

Fig. 3-1 Glade design environment

```
<?xml version="1.0"?>
<glade-interface>
  <!-- interface-requires gtk+ 2.16 -->
  <!-- interface-naming-policy toplevel-contextual -->
  <widget class="GtkWindow" id="main_window">
    <property name="visible">True</property>
    <property name="title" translatable="yes">GrassScope</property>
    <property name="window_position">center</property>
    <property name="default_height">680</property>
    <child>
      .....
          <child>
            <widget class="GtkImageMenuItem" id="quit_imagemenuitem">
              <property name="label">gtk-quit</property>
              <property name="visible">True</property>
              <property name="use_underline">True</property>
              <property name="use_stock">True</property>
              <signal name="activate" handler="on_quit_imagemenuitem_activate"/>
            </widget>
          </child>
      .....
    </child>
  </widget>
</glade-interface>
```

图 3-2 从 Glade 导出的 XML 样例

Fig. 3-2　A sample XML file from Glade

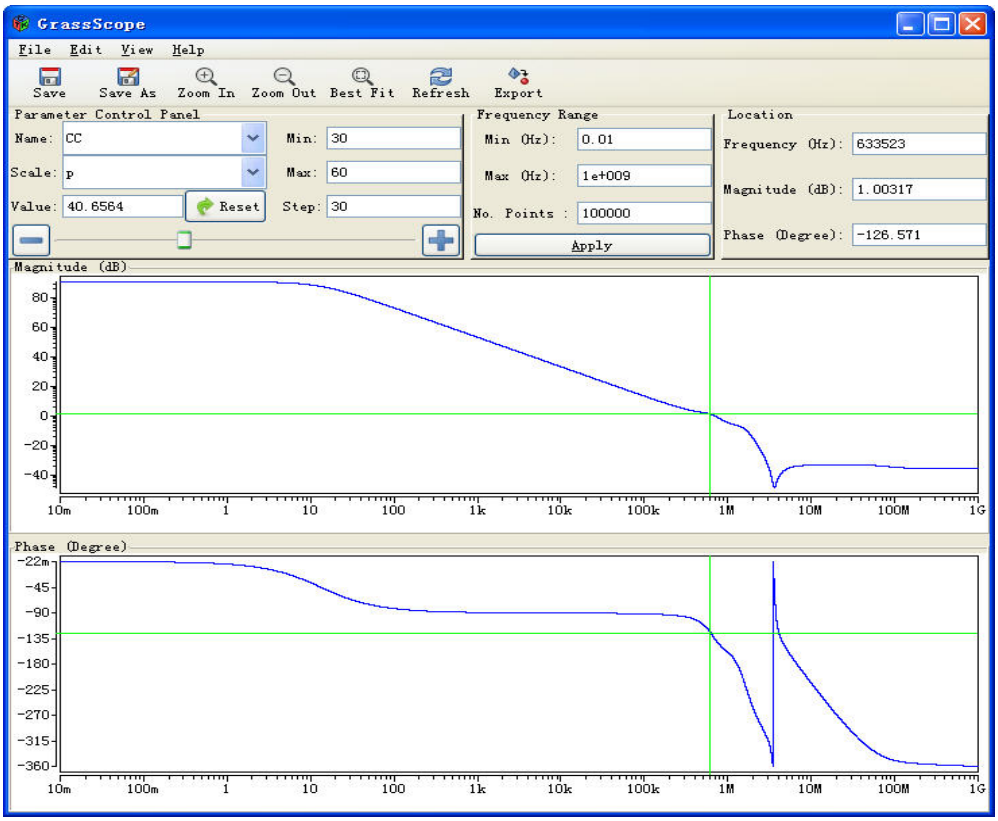


图 3-3 为 GRASS 设计的 2 维图形化用户界面

Fig. 3-3 2 D GUI for GRASS

3.1.3 2 维图形化用户界面的架构和基本功能描述

图 3-4 给出了 2 维图形化用户界面与 GRASS 结合之后的主体架构图：

(1) 调用 GRASS 对输入网表进行符号化分析，得到电路的符号化传输函数。

(2) 从符号化分析的结果中获取电路参数符号表，建立用户可控的参数表。

(3) 控制面板中为用户提供了若干种调节电路参数的控件，可实时获取用户的最新设置。

(4) 每当用户更新设置后，传输函数的幅度和相位图像会实时同步更新。

(5) 允许用户在当前的设置下用鼠标从图像上直接捕获数据，或者将当像图像上的数据点全部导出外部数据文件中用其它软件进行后续分析。

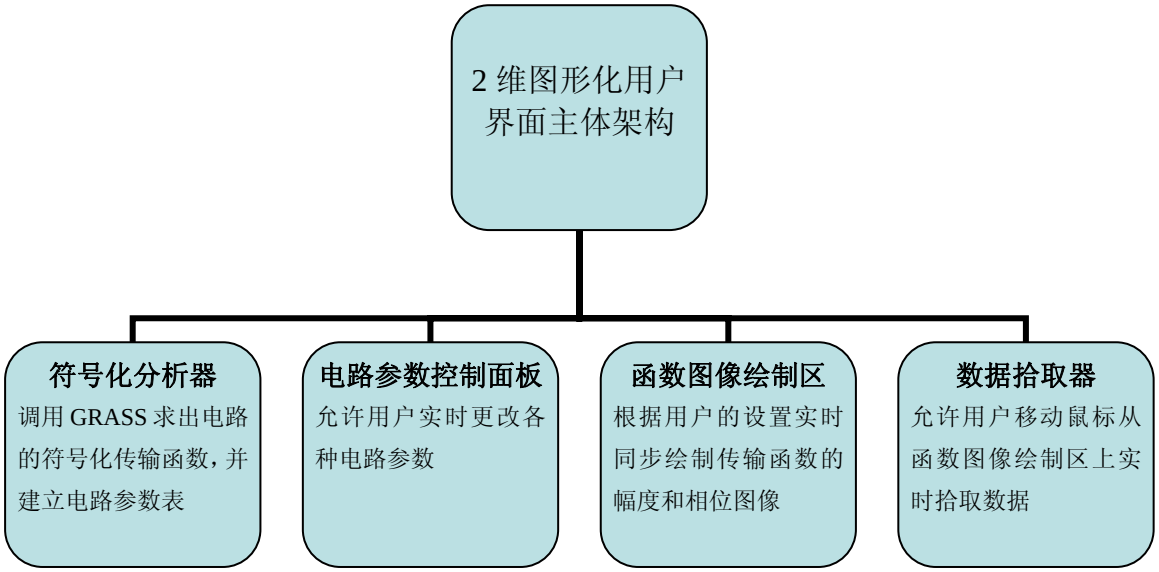


图 3-4 2 维图形化用户界面主体架构

Fig. 3-4 Main structure of the 2D GUI

3.1.4 2 维图形化用户界面的实现

本节将详细介绍 2 维图形化用户界面的实现细节。

3.1.4.1 动态载入 Glade 导出的用户界面信息

首先使用 Glade 完成用户界面上的各种控件的布局和属性设置工作（图 3-1）。完整的界面布局和控件属性设置信息都保存在 XML 文本文件中。C++代码中使用的控件名及其回调函数名都包含在该 XML 文件内。下面的 C++代码实现了在应用程序运行时将 XML 文件动态载入的功能：

```
#include <glade/glade.h>
const char GLADE_FILE[] = "grass.glade";
GladeXML *xml = glade_xml_new(GLADE_FILE, "main_window", 0);
```

其中 glade/glade.h 是由 libglade 库提供的头文件，包含类型 GladeXML 和函数 glade_xml_new()的声明，用于解析 Glade 导出的 XML 文件。这里的“grass.glade”就是事先用 Glade 导出的用户界面 XML 文件（图 3-2）。整段代码把用户界面主窗口“main_window”及其包含的所有子对象全部载入到内存中，用户将看到图 3-3 所示的主窗口出现在屏幕上。

3.1.4.2 为每个控件编写回调函数（Callback Function）

GTK+虽然采用面向过程的 C 语言编写，但是其使用了面向对象的代码风格。每一个控件就是 GTK+的一个内置对象。这些对象对于不同的信号会有不同的响应。GTK+的用户只要将某个函数指定为特定对象对于特定信号的响应，每当这个对象捕获到该信号就会运行这个由用户指定的函数。这样的函数称为回调函数（Callback Function）。回调函数又被称为对象的行为。一个 GTK+对象可以有多个回调函数与之关联，也可以没有任何回调整函数。

下面以界面中主窗口（main_window）为例来说明如何指定一个 GTK+对象对于特定信号的回调函数。

```
#include <glade/glade.h>
#include <gtk/gtk.h>
using namespace callbacks;
const char GLADE_FILE[] = "grass.glade";
const char MAIN_WINDOW[] = "main_window";
GladeXML *xml = glade_xml_new(GLADE_FILE, MAIN_WINDOW, 0);
GtkWidget *main_window = glade_xml_get_widget(xml, MAIN_WINDOW);
g_signal_connect(
  G_OBJECT(main_window), "destroy", G_CALLBACK(quit), 0);
```

由于用户界面的所有回调函数被封装于命名空间 callbacks 中，所以在使用它们前必须使用 C++的 using 指示：

```
using namespace callbacks;
```

glade_xml_get_widget()是 libglade 库提供的函数，需要包含 glade/glade.h 头文件以找到其正确的声明。它通过分析已经载入内存的用户界面（GladeXML 对象），找到由常量字符串 MAIN_WINDOW 指定的 GTK+对象，并将该对象的内存地址以 GtkWidget 指针的形式返回。

gtk/gtk.h 是 GTK+库最顶层的头文件。通过它可以找到 GTK+控件的基本类型 GtkWidget、把信号和回调函数关联起来的函数 g_signal_connect()，以及实现类型转换的宏 G_OBJECT 和 G_CALLBACK。

这段代码涉及到 GTK+使用的面向对象的技术多态性。如图 3-2 所示，控件“main_window”的实际类型是 GtkWindow；通过 glade_xml_get_widget()返回的指针 main_window 指向该控件，但指针类型是 GtkWidget；在使用 g_signal_connect()指定

回调函数时，GtkWidget 指针 main_window 又经过宏 G_OBJECT 的转换，以 GObject 指针的形式传入 g_signal_connect()内部。整个转换过程基于图 3-5 所示的 GTK+面向对象的继承关系图：

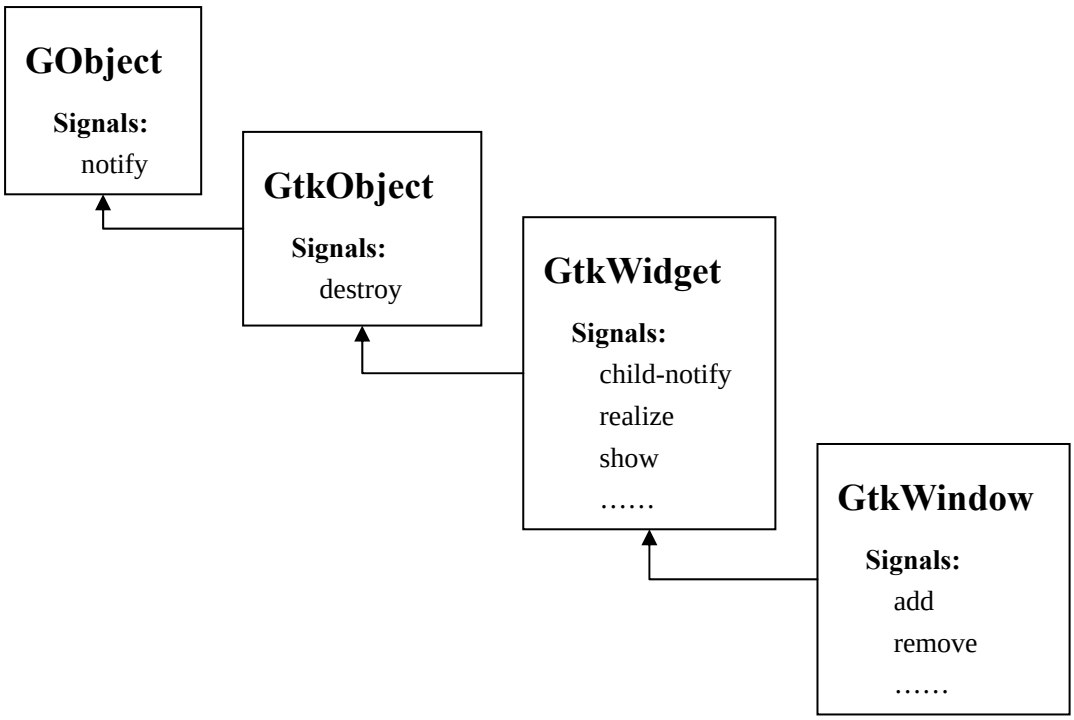


图 3-5 Gtk+的对象层次图

Fig. 3-5 GTK+ object hierarchy

GtkWindow 类型本身没有定义对于信号“destroy”的响应，但是其父类型 GtkObject 对此进行了定义，所以子类型 GtkWindow 也继承了该属性。因此将回调函数 quit()作为 GtkWindow 对象“destroy”信号的响应成为可能。

宏 G_CALLBACK 的定义如下：

#define G_CALLBACK(f) ((GCallback) (f))

它将一个函数指针转换为 GCallback 类型：

typedef void (*GCallback) (void);

这种函数指针类型被用于转换所有传入 g_signal_connect()的 GTK+回调函数签名。但这并不意味着回调函数即不能带任何参数也不能返回任何值。实际所需的回调函数要依具体情况而定[37]。不同 GTK+对象的不同信号所需的回调函数声明可能并不相同。例如，GtkObject 对象与信号“destroy”相关的回调函数原型必须是：

void (*) (GtkWidget *, gpointer)

其中 gpointe 是 GTK+中声明的类型，通常是 void *的别名。而 GtkDrawingArea 对象

与信号“expose-event”关联的回调函数原型则必须是：

gboolean (*) (GtkWidget *, GdkEventExpose *, gpointer)

其中 gboolean 是 GTK+中声明的类型，通常是 int 的别名。GdkEventExpose 是 GTK+中定义的结构体类型，仅当某个窗口变得可见并且需要重绘时才会由 GTK+库创建该对象，并将其地址作为第二参数传入回调函数。有关该类型的详细定义可参见[37]。

另外，如果使用 C++编译器编译包含调用回调函数的代码，则必须使用 extern "C"对所有 GTK+回调函数的声明进行包装。这是因为 GTK+使用了宏 G_CALLBACK 进行函数指针的类型，这就使得 C++编译器的符号改编（Name Mangling）[38]机制导致 g_signal_connect()在查找回调函数的定义时失败，成为编译过程中的障碍。包含 extern "C"声明的函数可以绕开 C++编译器的这种名称修饰机制，使汇编代码中的函数名与 C++代码中的函数名保持一致，从而避免了由于使用宏进行函数指针类型转换而产生的编译错误。

为了按上述方法统一处理用户界面的所有回调函数声明，回调函数的声明被统一置于同一个头文件中，并用 extern "C"包围所有的回调函数声明，所有在使用回调函数前只需要包含该头文件即可。下面是这个包含所有回调函数声明的头文件的一部分内容：

```
#ifndef _CALLBACKS_H_
#define _CALLBACKS_H_
#include<gtk/gtk.h>
#include<gmodule.h>
namespace callbacks
{
extern "C"
{
G_MODULE_EXPORT void quit(GtkWidget *widget, gpointer user_data);
G_MODULE_EXPORT gboolean on_magnitude_drawingarea_expose_event(
   GtkWidget *widget,
   GdkEventExpose *event,
   gpointer user_data);
// other Callback functions
} // extern "C"
```

```
} // namespace callbacks
#endif // _CALLBACKS_H_
```

由于 2 维图形化用户界面的代码被设计为可以在 Windows 上编译，所以上面的代码在每个回调函数的声明前用到了宏 G_MODULE_EXPORT。这个宏定义在头文件 gmodule.h 中：

```
#ifdef G_PLATFORM_WIN32
#    define G_MODULE_EXPORT __declspec(dllexport)
#else // !G_PLATFORM_WIN32
#    define G_MODULE_EXPORT
#endif // !G_PLATFORM_WIN32
```

由于操作系统的差异，在 Windows 系统上编译这份代码时，需要用这个宏把回调函数标记为可以动态导出。在 Linux 和其它类 Unix 系统上这个宏什么也不做。

3.1.4.3 使用组合框（GtkComboBox）实现可选符号选择表

为了让用户能够自由地选择电路中的某个参数，并对其数据进行实时调节，用户界面使用组合框（GtkComboBox）来实现这种功能。

组合框允许用户从一组给定的选项中选定其中一项。它能显示出当前选定的项目。当用户激活组合框后，将弹出一个下拉菜单以显示所有的可用选项，并允许用户选择新的项目作为当前选项。它的具体样式受当前用户的窗口系统环境影响，在不同的窗口系统下的显示风格略有不同。

组合框用一种链表结构（GtkListStore）来保存事先设定的所有可用的选项，并提供了丰富的函数接口对这种结构进行增添和渲染操作。用户界面中的函数 init_combobox()使用从符号化分析结果中提取的完整符号表来创建一个包含所有符号名称的组合框（combobox）：

```
void init_combobox(
   GtkWidget *combobox, const gchar *str_list[], int str_list_size)
{
   const gint N_COLUMNS = 1;
   const gint COLUMN_STRING = N_COLUMNS - 1;
   GtkTreeIter iter;
   // create a list for all available items
```

```
GtkListStore *list_store = gtk_list_store_new(
   N_COLUMNS, G_TYPE_STRING);
for (int i=0; i < str_list_size; ++i)
{
   // add a new row to the model
   // the store will keep a copy of the string internally
   gtk_list_store_append(list_store, &iter);
   // sets the value of one or more celss in the row refereced by iter
   // the variable argument list should contain integer column numbers,
   // each column number followed by the value to be set
   // NOTE: The list is terminated by a -1.
   gtk_list_store_set(
      list_store, &iter, COLUMN_STRING, str_list[i], -1);
}
// GtkCellRenderer is used primarily by GtkTreeView widget.
GtkCellRenderer *renderer = gtk_cell_renderer_text_new();
gtk_cell_layout_pack_start(
   GTK_CELL_LAYOUT(combobox), renderer, TRUE);
gtk_cell_layout_set_attributes(
   GTK_CELL_LAYOUT(combobox), renderer, "text", 0, 0);
// Add list_store to widget
gtk_combo_box_set_model(
   GTK_COMBO_BOX(combobox), GTK_TREE_MODEL(list_store));
}
```

由这个函数处理过的可选符号选择表如图 3-6（在 Windows 平台上的 MinGW 环境下的运行结果）所示：

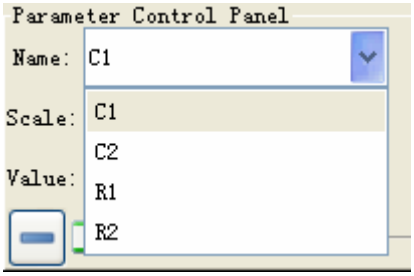


图 3-6　用 GtkComboBox 实现的 2 维用户界面中的参数符号选择表

Fig. 3-6　A parameter symbol choice table in 2D GUI implemented by GtkComboBox

用户从组合框中选定一个符号的名称后，用户界面就会在后台准备好与该名称对应的符号的相关资源，以响应用户的进一步设置。后台进行的这一系列动作需要用回调函数来实现，而用户从组合框中选择一个符号的操作会产生释放一个信号“changed”给 GtkComboBox 对象。采用 3.1.4.2 中提到的方法可以具有下面函数原型的函数绑定到 GtkComboBox 对象的“changed”信号上：

void on_combobox_changed(GtkComboBox *, gpointer);

在用户界面中该回调函数的主要任务是：找到当前名称对应的电路参数对应的数据指针，便于用户通过其他控件修改当前的电路参数值。

用户界面中还在另一处使用了组合框，实现了对当前所选电路参数数量级单位的实时变换。如图 3-7 所示，对于电容 C_1 的值，为了让右侧文本框中显示的数据更具可读性，10^{-9} 量级（nF）是一个比较合适的单位，所以可以在组合框中选择“n”来改变文本框中数据的显示格式。该操作不会影响后台计算的精度而仅仅影响用户界面的显示效果。

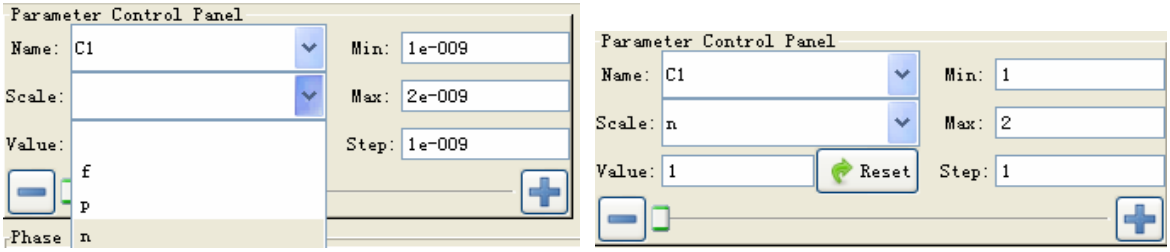


图 3-7　组合框中的不同选择对数据显示效果的影响

Fig. 3-7　Impact of selecting different items in a GtkComboBox object on the digital display style

3.1.4.4　使用水平拖拽控件（GtkHScale）实时调节参数值

要绘制出符号化表达式的图像就必须依据某中条件确定每个符号参数的具体数值。GRASS 将电路网表包含的原始数据作为符号参数的默认值，2 维用户界面也继

承了这套方案。为了让用户实时地看到某个参数的改变对于传输函数图像（包含幅度和相位图像）的影响，2 维用户界面使用水平拖拽控件（GtkHScale）实现这样的功能。

水平拖拽控件允许用户使用鼠标或键盘控制滑块的位置，根据滑块的位置从指定数值范围中选取一个数值。如图 3-8 所示，对应当前选定的参数名称而设定的数值显示在下侧 Value 文本框中，右侧的 Min 和 Max 文本框可实时设定水平拖拽控件的数值范围，Step 文本框可实时设定每按一次 PageUp 和 PageDown 键使滑块移动的距离。

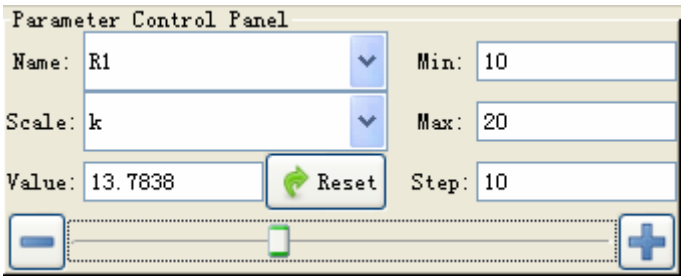


图 3-8　水平拖拽控件及其在文本框中显示的实时数据

Fig. 3-8　A GtkHScale object and its real-time value displaying in a text box

因滑块位置改变而引发的信号“value-changed”会触发相应的回调函数。该回调函数会依次进行执行下列操作：

(1) 把当前滑块位置对应的实时数据保存到已经在组合框中选定的参数名称对应的变量中，从而实时地改变符号化传输函数的这个参数值；

(2) 调用更新后的符号化传输函数在图形绘制区同步更新幅度和相位图像，实时地反映出当前参数改变对于传输函数幅度和相位的影响。

3.1.4.5　使用绘图区（GtkDrawingArea）绘制函数图像

图形化用户界面与符号化仿真器结合后最主要的工作就是要实时地绘制出传输函数的图像。2 维图表化用户界面使用 GtkDrawingArea 来完成这项工作。

GtkDrawingArea 本身实际上是一个空控件，但可以在回调函数中通过 GtkDrawingArea 的成员变量来绘制图形。在所有的信号中，“configure-event”、“expose-event” 和“motion-notify-event”信号最常用。

“configure-event”信号在绘图区大小发生变化时（如用户拖动鼠标改变窗口的大小）释放。其回调函数的原型为：

```
gboolean on_configure_event(
  GtkWidget *, GdkEventConfigure *, gpointer);
```

“expose-event” 信号负责在必要的时候（如当前绘图区从被其窗口遮挡的位置回到屏幕最前方时）触发回调函数重新绘制整个绘图区。其回调函数的原型为：

gboolean on_expose_event(

GtkWidget *, GdkEventExpose *, gpointer);

“motion-notify-event” 信号的释放意味着鼠标正在绘图区内移动，可用于实时捕捉当前的鼠标位置，以实时获取图像上的数据。其回调函数的原型为：

gboolean on_motion_notify_event(

GtkWidget *, GdkEventMotion *, gpointer);

图形化用户界面会按下面的步骤绘制出符号化传输函数的图像，图 3-9 显示了最终的图形化用户界面：

(1) 任何时候绘图区接收到“configure-event”和“expose-event”都会调用回调函数重绘整个传输函数的图像；

(2) 在指定频率范围内按照用户给定的频率采样点数，利用最新的符号化传输函数计算出每个频率采样点处的幅度和相位值，将所有的频率值、幅度值和相位值分别存放于不同的数组中；

(3) 根据用户的设置，绘制幅度图上的频率轴刻度（log10 尺度）和幅度轴刻度（dB），并将频率和幅度值数组中的数据点按绘图区纵横比例缩放调节后，用首尾相接的直线段绘制出来；

(4) 根据用户的设置，绘制相位图上的频率轴刻度（log10 尺度）和相位轴刻度（线性），并将频率和相位值数组中的数据点按绘图区纵横比例缩放调节后，用首尾相接的直线段绘制出来；

(5) 如果用户通过界面上的水平拖拽滑块、Reset 按钮或者手工输入数据至 Value 文本框的方式更改了传输函数的参数值，则在这些控件的回调函数中直接调用原本绑定到绘图区的回调函数重复前面 3 个步骤，以实时更新传输函数图像；

(6) 无论鼠标处于绘图区的任何位置，都能实时地在 Location 面板中的文本框中显示出当前鼠标所指频率及其对应的传输函数幅度值和相位值，并在绘图区的相应位置用绿色交叉线标注；

(7) 允许用户在界面的 Frequency Range 区域重新设置频率范围和频率采样点数，点击 Apply 按钮后将采用最新的设置重绘函数图像。

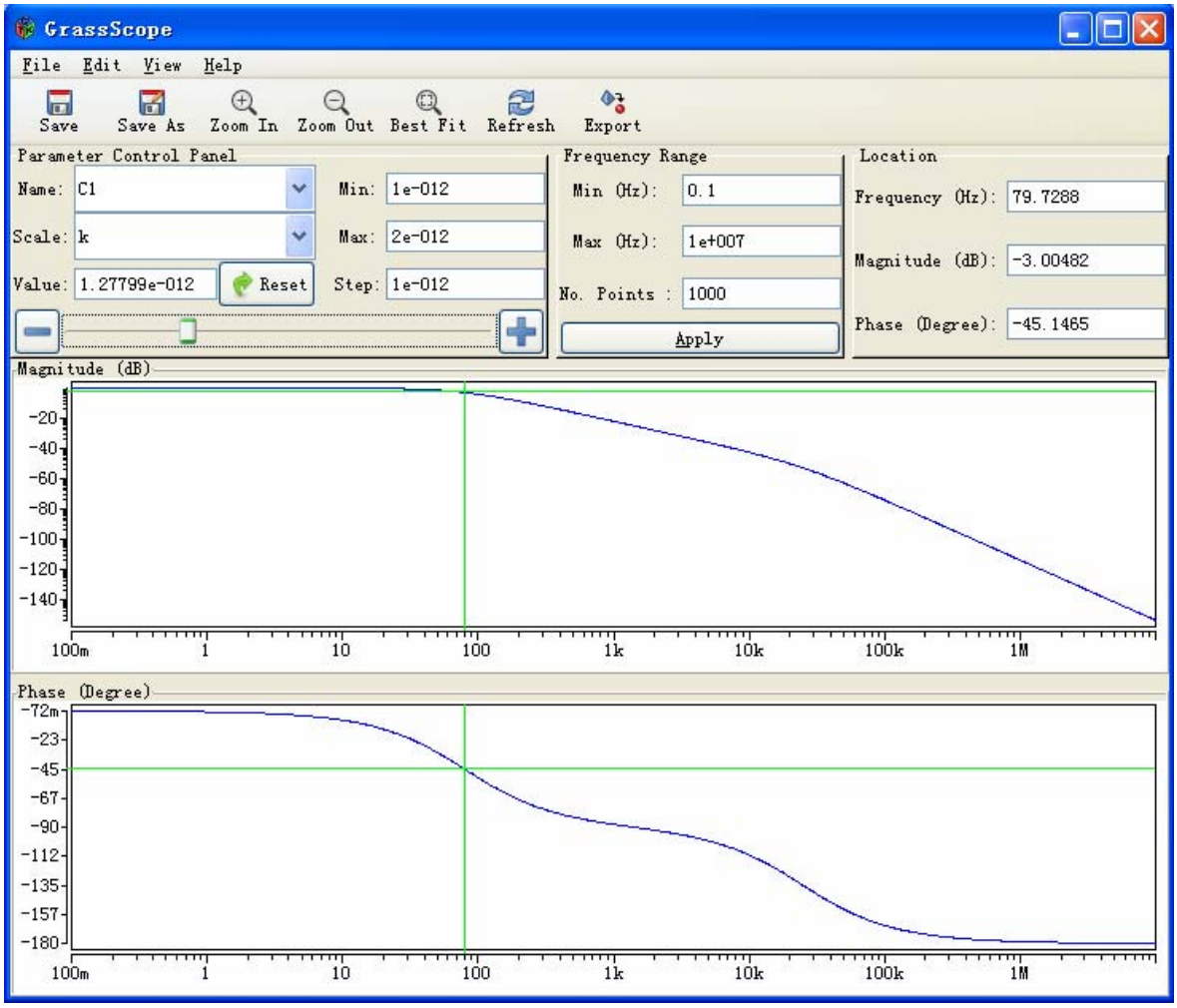


图 3-9 一个完整的图形化用户界面

Fig. 3-9 A full 2D GUI

3.1.4.6 用文件选择对话框（GtkFileChooserDialog）导出数据

为了让用户能导出传输函数图像数据，用户界面上特别设置了 Export 按钮，通过点击该按钮来创建一个 GtkFileChooserDialog 对象来实现这样的功能。

GtkFileChooserDialog 是一个独立的窗口，需要用 Glade 设计并与主窗口保存在同一级别下。每当用户点击 Export 按钮时，关联到该按钮“clicked”信号的回调函数就会使用 glade_xml_get_widget()查询 GtkFileChooserDialog 对象在内存中的地址，并设置好当前目录、默认输出文件名等相关信息，然后进入无限循环等待用户的响应，如图 3-10 所示。此时用户可以改变输出文件名并点击 OK 或直接点击 Cancel 按钮退

出该循环，用户界面恢复至普通状态。

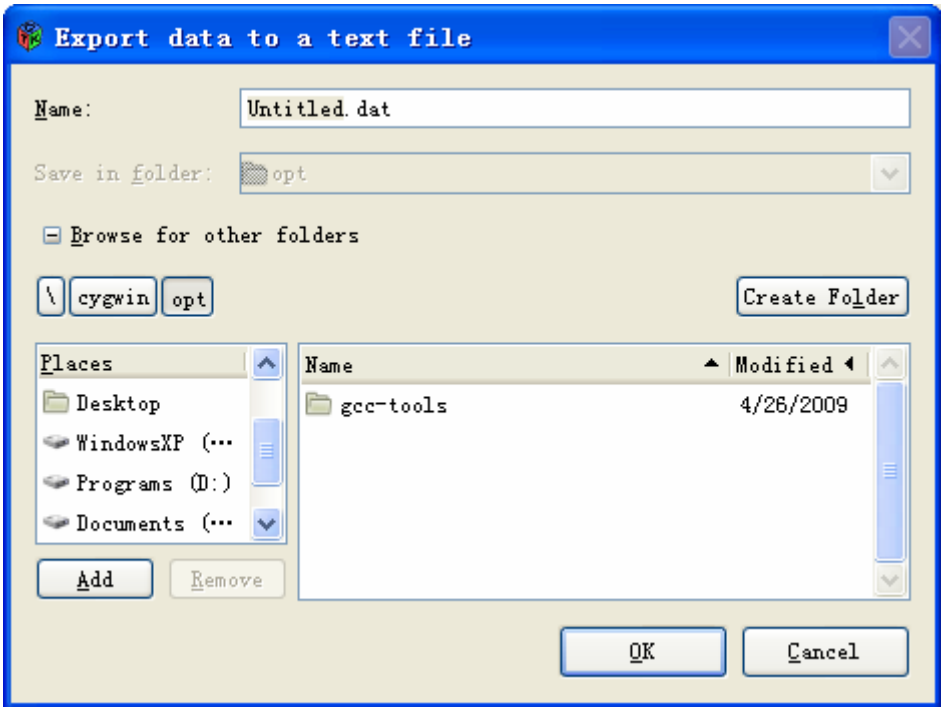


图 3-10 一个用于导出数据的对话框

Fig. 3-10 A dialog for exporting data

3.2 针对多参数符号化分析的 3 维图形化用户界面

对于单参数的优化问题，传统数值仿真器已经具有一定的解决方案，但直观性和用户互动性均较差。对于两个以上参数的同步优化问题，模拟电路设计者更希望有互动的图形界面，对参数选取能有一定的控制自由度。由于电路参数优化通常是一个综合权衡的过程，设计者需要综合考虑电阻值、电容值的选取对于电路板图（面积）、功耗是否会发生过大影响，而不单纯是某个数值选取的过程。出于这种考虑，图形化用户界面应当提供交互式展示性能指标关于参数的函数曲面（3 维图形），让用户自行选择合适的电路参数值，并得到关于性能敏感度的一个直观感觉。

本节将详细介绍 3 维图形化用户界面的相关背景和具体实现。

3.2.1 3 维图形界面开发库 OpenGL 和 GLUT 介绍

OpenGL（Open Graphics Library）是一个定义了一个跨编程语言、跨平台的编程接口的规格。它提供了为数不多的基本几何图元，如点、直线和多边形和一系列复杂

的渲染方法。用户利用这些基本工具就能绘制出具有很好视觉效果的复杂图形，而且很多基于图形的运算在图形处理单元（GPU）上完成，能够有效地减轻 CPU 的负荷[39]。

OpenGL 常用于 CAD、虚拟实境、科学可视化程序和电子游戏开发。其高效实现（充分利用了专用图形加速硬件，如 GPU）是功能强大、调用方便的底层图形库，存在于 Windows、很多 UNIX 平台和 Mac OS。这些实现一般由显示设备厂商提供，而且非常依赖于该厂商提供的硬件。它能帮助程序员实现在 PC、工作站、超级计算机等硬件设备上的高性能、极具冲击力的高视觉表现力图形处理软件的开发。

GLUT（OpenGL Utility Toolkit）是用于编写 OpenGL 程序的独立于窗口系统的工具集。它为 OpenGL 实现了一个简单的窗口应用程序接口，使学习和研究 OpenGL 编程变得更加容易。GLUT 提供了一套可移植的应用程序接口，这使得编写中小型的可独立运行的跨平台 OpenGL 程序变得简单易行。

3.2.2 3 维图形化用户界面的架构和基本功能描述

图 3-11 给出了 3 维图形化用户界面与 GRASS 结合之后的主体架构图：

(1) 读取用户设置文件，从中确定待分析的两组电路参数和若干电路指标约束条件；
(2) 调用 GRASS 对输入网表进行符号化分析，得到电路的符号化传输函数，将符号化传输函数绑定到全局对象上；
(3) 根据用户的设置，绘制每个电路指标关于这两组电路参数的空间曲面；
(4) 在空间曲面窗口中绘制各种空间曲面和一张半透明的、可由用户实时调节高度的平面，用户还能实时操控观察角度；
(5) 在等高线窗口中同步绘制空间曲面中平面与曲面相交得到的等高线；
(6) 用户可在等高线窗口中测量和比较不同曲面的等高线数据；
(7) 用户可以从状态窗口中获取更为详细的数据资料。

图 3-12 给出了一个包含 3 维图形化用户界面全部 3 个窗口的例子。

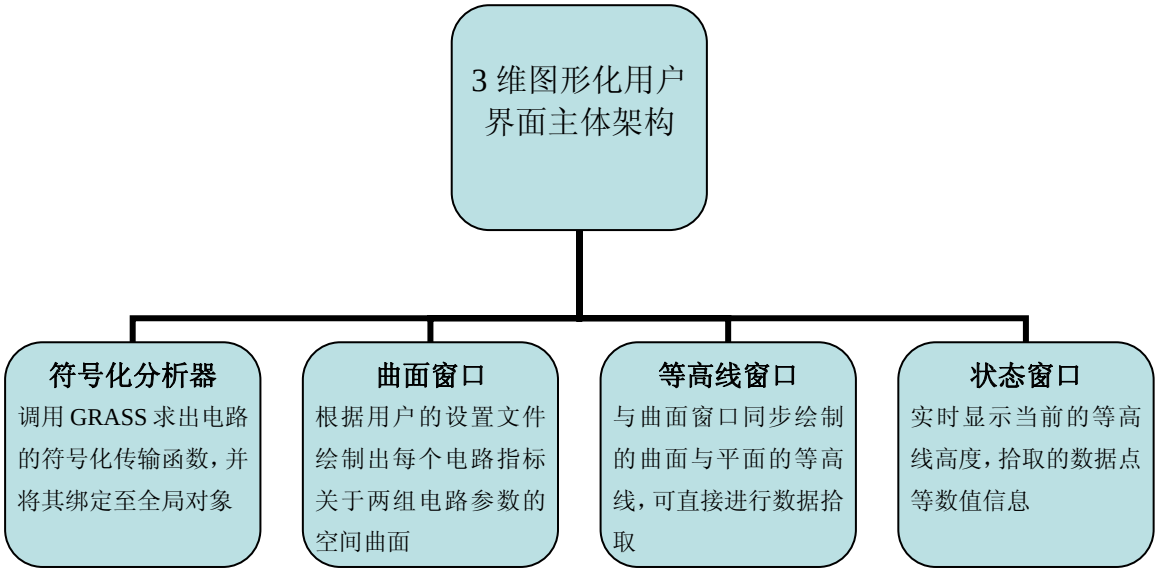


图 3-11 3 维图形化用户界面主体架构

Fig. 3-11 Main structure of the 3D GUI

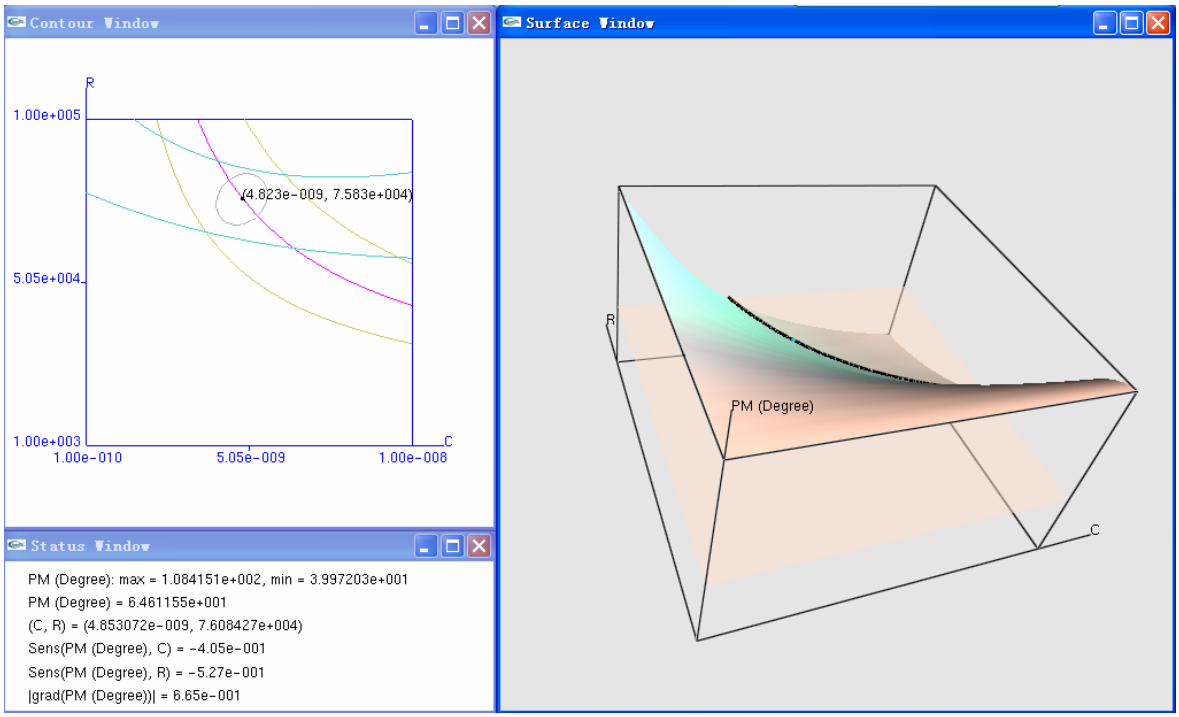


图 3-12　3 维图形化用户界面样例

Fig. 3-12　Sample of the 3D GUI

3.2.3　3 维图形化用户界面的实现

由式(2.5)描述的这一类优化问题本质上可以归结为如下的数学问题：

问题 1：求空间曲面 $z = f(x, y)$ 在 $z = z_0$ 的曲线上使梯度模 $\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$ 最小的点 $P_{opt}(x_{opt}, y_{opt})$。

通常我们认为 x，y 的随机扰动在梯度较小的点对函数值产生的影响较小，所以系统的稳定性就相对较高。

为了以图形方式展示出问题 1 的解，首先需要在特定区域内绘制出空间曲面 $z = f(x, y)$ 。我们采用了 OpenGL 图形系统来实现空间曲面的绘制。

用 OpenGL 绘制二元函数对应的空间曲面可以用三角形条带（GL_TRIANGLE_STRIP，图 3-13）或四边形条带（GL_QUAD_STRIP，图 3-14）两种方式拼接而成。指定条带类型后只需要按图示顺序依次指定每个三维顶点就能自动生成条带。将这些条带拼接起来就得到了近似的空间曲面。如果再为每个顶点指定单位法向量并设置合适的光照就能得到立体感很强的空间曲面。我们选择三角形条带来绘制空间曲面，是因为每小块三角形面片（如图 5 中的 $\triangle V_2V_3V_4$ ）的顶点都落在同一平面上，便于进行后续的图形计算。

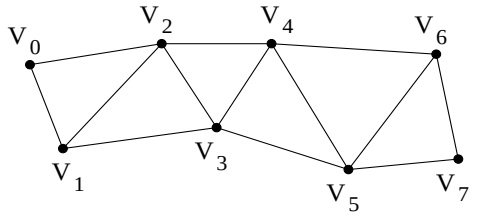


图 3-13　OpenGL 中的三角形条带样例

Fig. 3-13　Sample of a triangle strip in OpenGL

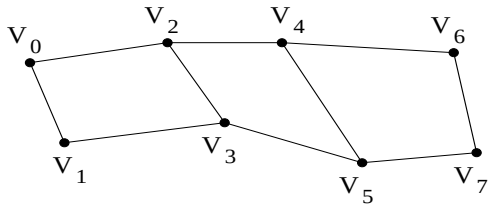


图 3-14　OpenGL 中的四边形条带样例

Fig. 3-14　Sample of a quad strip in OpenGL

为了获得问题 1 的解，直接想法是求出曲面 $z = f(x, y)$ 与平面 $z = z_0$ 的交线方程。对于实际电路网络而言，$f(x, y)$ 就是传输函数 $H(s; x, y)$，通常以高阶有理分式的形式出现，所以直接求解方程 $f(x, y) = z_0$ 并不现实。但可以通过下面的问题 2 得到直观的解决方案：

问题 2：绘制由方程 $f(x, y) = z_0$ 确定的隐函数 $y = y(x)$对应的二维曲线。

通过三角形条带绘制曲面，实际上已经把原本光滑的曲面转换为图 5 所示的大量三角形面片。于是曲面 $z = f(x, y)$ 与平面 $z = z_0$ 的交线完全可以由每一小块三角形面片与平面 $z = z_0$ 的交线段的集合来近似。图 3-15 给出了三维空间中任意一块三角形面片与一个无限大平面 α 的全部 8 种相对位置关系。其中实心点表示位于平面 α 上方（以法线为正向）的顶点，空心点表示位于平面 α 下方的顶点, 虚线为交线段。注意到其中第（i）种和第（7−i）种位置关系总是等价的，可以利用该特性来减少计算量。

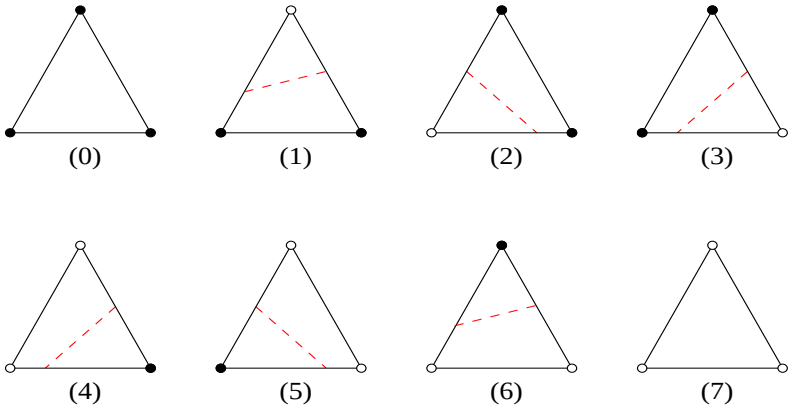


图 3-15　三角形面片与一个无限平面的 8 种相对位置和交线段

Fig. 3-15　8 relative positions and intersecting lines between a triangle piece and an infinite plane

只要每块三角形面片取得足够小并计算出每块小面片与平面 $z = z_0$ 的交线段，再将这些交线段按任意次序摆放在同一平面上就能得到问题 2 中 2 维曲线的近似图样。图 3-16 展示了我们的图形界面用上述方法绘制出的二元函数 $z = \sin(x^2+y^2) / (x^2+y^2)$，$x, y \in [1, 3]$，在 $z = -0.03$ 时的等高线，其中空间曲面由 800 块三角形面片拼接而成。图 3-16 所示的约束平面 $z = z_0$ 的高度可以由用户实时更改（由鼠标拉动），对应的等高线会在左边的窗口中同步更新，便于用户跟踪参数的变化轨迹。

下面将详细介绍 3 维图形化用户界面各主要部件的详细实现。

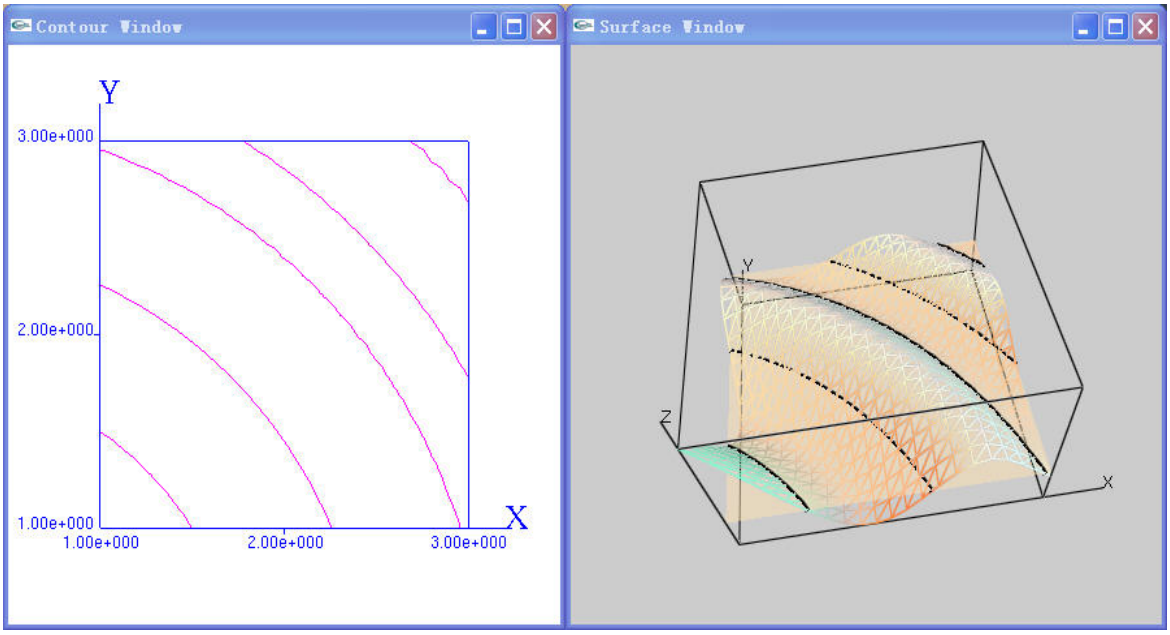


图 3-16　曲面 $z = \sin(x^2+y^2) / (x^2+y^2)$（右）及其在 $z = -0.03$ 时的等高线（左）

Fig. 3-16　$z = \sin(x^2+y^2) / (x^2+y^2)$ (right) and its contour lines on $z = -0.03$ (left)

3.2.3.1　空间曲面的绘制

用 OpenGL 绘制形如 $f(x, y)$ 的空间曲面，通常分三步进行：

(1) 在给定矩形区域$\{(x, y) \mid x_{\min} \le x \le x_{\max}, y_{\min} \le y \le y_{\max}\}$内以均匀采样的方式反复调用符号化表达式求得顶点矩阵；

(2) 由顶点矩阵计算出每个顶点的单位法向量，得到单位法向量矩阵；

(3) 使用顶点矩阵绘制出三角形面片组成的条带，这些条带正好拼接成完整曲面；

(4) 加上适当的环境光，增强图形的立体感。

求顶点矩阵的过程很简单，只需要两重循环即可完成全部计算。由于采样区域是矩阵区域，所以可以使用图 3-17 所示的方法来控制顶点矩阵和法向量矩阵占用的空间，只需要$(n+1)^2-1$个单元就能放一个包含n^3个元素的矩阵，其中$f_{ij} = f(x_i, y_j)$，i, j = 1, 2, …, n。

	$y_1=y_{\min}$	y_2	…	$y_n=y_{\max}$
$x_1=x_{\min}$	f_{11}	f_{12}	…	f_{1n}
x_2	f_{21}	f_{22}	…	f_{2n}
…	…	…	…	…
$x_n=x_{\max}$	f_{n1}	f_{n2}	…	f_{nn}

图 3-17 3 维顶点矩阵和法向量矩阵的 2 维压缩表示

Fig. 3-17 2D compressive representation for 3D vertex matrices and normal vector matrices

要使 OpenGL 中的环境光产生真实的效果，需要为每个顶点指定一个单位法向量。因此顶点矩阵的每个元素都对应一个单位法向量，这些单位法向量构成的矩阵称为法向量矩阵。OpenGL 内部并没有为一般曲面自动计算法向量的工具，所以需要用下面的算法为每个顶点计算近似法向量。

由于整个曲面都是用图 3-13 中的大量小三角形拼接而成，每个小三角形内部的点都有相同的法向量。可以把与一个顶点相连的所有小三角形的法向量都叠加起来作为该顶点的法向量，这样得到的法向量就是一个与该顶点关联的比较合理的单位法向量，如图 3-14 所示。

注意到同一平面的法向量有正负之分，会影响光照对正反表面的表现力，所以应当按某个固定的下标顺序来计算同一曲面上的每一个三角形面片的法向量。对于任意一个三角形面片 $\triangle V_{i,j}V_{i-1,j}V_{i,j+1}$，其法向量如式(3.1)所示

$$\overrightarrow{n_{\triangle V_{i,j}V_{i-1,j}V_{i,j+1}}}=\overrightarrow{V_{i,j+1}V_{i,j}}\times\overrightarrow{V_{i,j}V_{i-1,j}} \tag{3.1}$$

使用这种方法遍历曲面上的所有顶点，就能建立起与顶点矩阵对应的法向量矩阵。最后将法向量矩阵的每个元素单位化后就得到了所需的单位法向量矩阵。

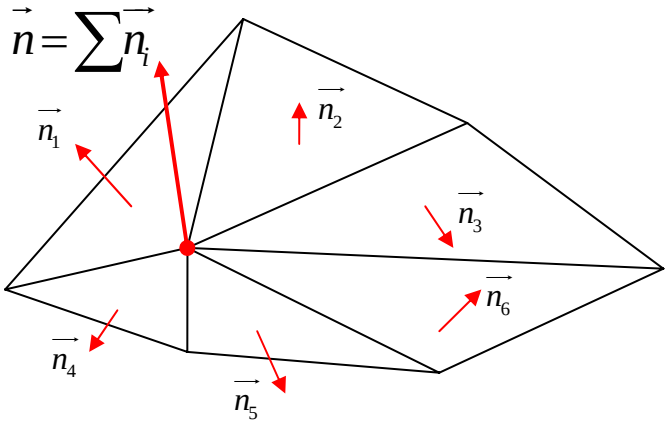


图 3-18 求一个顶点法向量的算法

Fig. 3-18 An algorithm to evaluate a normal vector

得到顶定矩阵和单位法向量矩阵后就能调用 OpenGL 函数在窗口中绘制曲面了。下面的代码将同时开启多盏环境灯，并选择灯光的模型为“双面光照（GL_LIGHT_MODEL_TWO_SIDE）”。双面光照能根据顶点法向量的正负产生不同的光照效果。

```
glEnable(GL_LIGHTING);
for (size_t light_index = 0; light_index < NUM_LIGHTS; ++light_index)
  glEnable(GL_LIGHT0 + light_index);
glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE);
```

下面的代码设置了曲面的材质、高光效果以及对环境光的反散效果等参数，并依次绘制出三角形面片组成的条带，最终形成具有很强立体感的曲面。图 3-19 是用 3 维界面绘制的一张曲面，$z = \sin(x + y^2)$。

```
glShadeModel(GL_SMOOTH);
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
glMaterialfv(GL_FRONT, GL_SPECULAR, front_material_specular);
glMaterialfv(GL_FRONT, GL_SHININESS, front_material_shininess);
glMaterialfv(
  GL_FRONT, GL_AMBIENT_AND_DIFFUSE, front_material_specular);
glMaterialfv(GL_BACK, GL_SPECULAR, back_material_specular);
glMaterialfv(GL_BACK, GL_SHININESS, back_material_shininess);
glMaterialfv(
  GL_BACK, GL_AMBIENT_AND_DIFFUSE, back_material_specular);
```

```
for (size_t i = 0; i < num_ypoints - 1; ++i)
{
   glBegin(GL_TRIANGLE_STRIP);
   for (size_t j = 0; j < num_xpoints; ++j)
   {
      glNormal3dv(normalmat[i+1][j].v());
      glVertex3dv(vertexmat[i+1][j].v());
      glNormal3dv(normalmat[i][j].v());
      glVertex3dv(vertexmat[i][j].v());
   }
   glEnd();
}
```

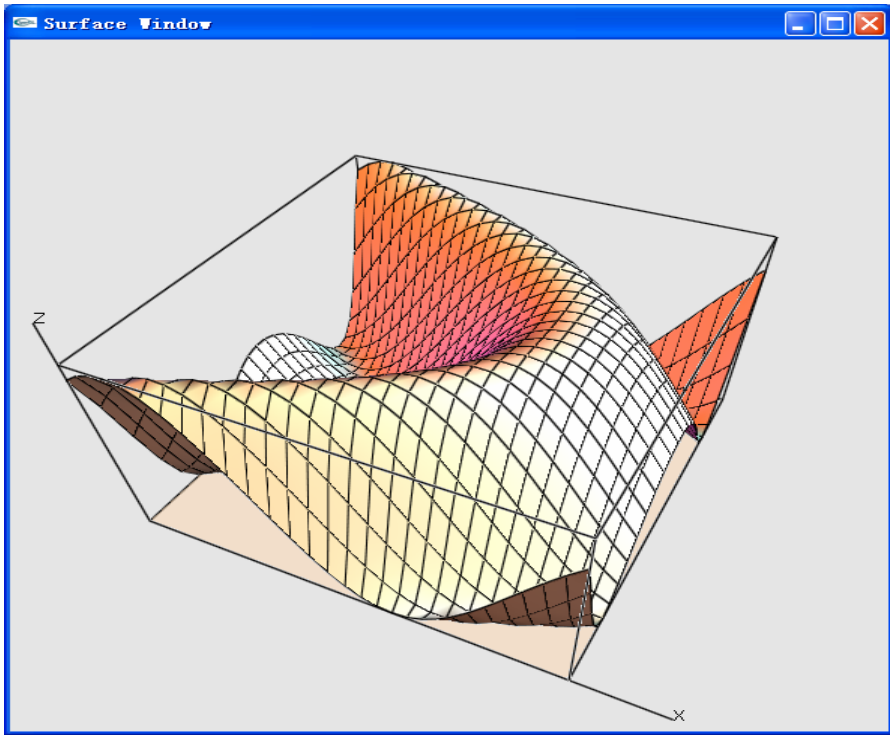


图 3-19 求一个顶点法向量的算法

Fig. 3-19 An algorithm to evaluate a normal vector

3.2.3.2 使用 GLUT 创建窗口系统

OpenGL 的代码虽然已经成型，但真正要产生图 3-19 所示的窗口，还需要借助

GLUT 库的支持。把 OpenGL 代码嵌入利用 GLUT 库生成的窗口代码中才能在用户环境下绘制出精美的图像。

在创建一个 GLUT 窗口前，必须指定窗口的若干特征。例如，刷新窗口的方式是单缓冲还是双缓冲、颜色的存储方式是 RGBA 还是颜色索引值、窗口的初始位置等。一般地，在用 glutCreateWindows()函数创建一个窗口之前，必须依次调用下列函数：

(1) glutInit(int argc, char **argv)

它接收命行可能出现的选项，对整个 GLUT 函数库进行初始化。它的参数值应该与整个程序的 main()函数保持一致。

(2) glutInitdisplayMode(unsigned int mode)

它指定了窗口的显示模式，另外还可以指定与即使创建的窗口相关联的深度缓冲区、模板缓冲区和累积缓冲区。由于有跨平台的需要，3 维用户界面设置的参数是 GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH | GLUT_STENCIL。这样设置得到的窗口运行最稳定，在配制不太差的计算机上运行效率也不会受太大影响。

(3) void glutInitWindowSize(int width, int height)

它指明了新窗口的长宽，以像素为单位。它的设置只是初始设置，以后可以被用户或其它代码更改。

(4) void glutInitWindowPosition(int x, int y)

它指定了新窗口首次出现在屏幕上时的位置，以后也可能被更改。

(5) int glutCreateWindow(char *name)

它使用前面各步完成的设置来创建一个完整窗口。参数 name 用于指定新窗口的标题文字。函数的返回值是保证唯一的整型标识符，用于区别 GLUT 窗口中的不同窗口。可以将这个整数当作窗口的身份代码，在多窗口应用程序中特别重要。

(6) glutMainLoop()

最后程序进入 GLUT 主循环需要调用这个函数。OpenGL 从这里开始进行所有窗口的渲染。

GLUT 库用类似于 GTK+库的回调函数触发方式对窗口中的事件进行响应。不过它注册回调函数的方式要简单许多，只需要在创建窗口后、进入主循环前使用下列函数把回调函数关联到特定的信号上即可，可以每个子窗口注册不同的回调函数：

(1) void glutDisplayFunc(void (*func) (void))

它指定了当窗口中的内容需要重绘时触发的回调函数。在发生下面这些情况时，窗口就会重绘：窗口刚被打开、窗口被弹出、窗口的内容遭到破坏，以及地调用了 glutPostRedisplay()函数时。前面介绍的用 OpenGL 函数绘制图像的代码都写在这个回调函数中，由 GLUT 负责调用。

(2) void glutReshapeFunc(void (*func) (int width, int height))

它指定了当窗口大小发生改变或窗口被移动时所调用的函数。其中的 width 和 height 是窗口大小改变后的最新长度和宽度。在 3 维用户界面中，该回调函数被用于实时控制窗口的纵横比和窗口的可视区域。

(3) glutKeyboardFunc(void (*func) (unsigned char key, int x, int y))

它指定的函数可以获取一个 ASCII 字符按键被按下时按键的字符值和当前鼠标的位置。这对于交互式用户界面十分重要，可以用它设置一些快捷按键，方便进行一些需要频繁进行的操作。

(4) glutSpecialFunc(void (*func) (int key, int x, int y))

它的功能与 glutKeyboardFunc()非常相似。唯一不同之处是，它只响应非 ASCII 字符按键按下的运作。3 维图形化用户界面中，它注册的函数被用于捕捉 PageUp、PageDown，以及 4 个方向按键按下的信号，用于实时控制曲面窗口中观察者所处的角度。

(5) void glutMouseFunc(void (*func) (int button, int state, int x, int y))

它注册的回调函数被用于等高线窗口中用鼠标捕捉数据点。参数中的 button 是指鼠标的左键或右键。下面是这个用户界面回调函数中的一段代码，展示了如果检测到鼠标左键被按下并释放的动作：

if (state == GLUT_DOWN && button == GLUT_RIGHT_BUTTON) {}

(6) glutMotionFunc(void (*func) (int x, int y))

它注册的回调函数负责与用户交互，以操控曲面绘制窗口中的半透明平面上下移动，实时地产生与曲面的交线（一条等高线），如图 3-16 所示。

(7) glutPostRedisplay(void)

该函数不带任何参数，在 glutReshapeFunc()注册的回调函数内部使用。在 3 维用户界面中由于需要与用户交互，某些操作也会调用这个函数实时刷新窗口。

至此，3 维图形化用户界面不仅有了能绘制各种精美图形的窗口，还能处理用户端的输入事件，实现了与用户的实时交互。

3.3 并行计算的初步应用

通常情况下，模拟电路工程师需要同时考虑若干种电路指标。而现在多核 CPU 已经广泛应用于个人电脑中，利用 CPU 的多个核同步计算相互独立的电路指标会极大地提高符号化分析后期应用的速度。

3 维图形化用户界面中使用 OpenMP 对多个电路指标的数值化过程进行同步计算，成倍地提高了整个分析流程的速度。

另外，2 维图形化用户界面中也使用了同样的技术，用多线程按频率分段同步计算传输函数的采样点数值。

3.3.1 多线程指导性注释 OpenMP 介绍

OpenMP 是由 OpenMP Architecture Review Board 牵头提出的，并已被广泛接受的，用于共享内存并行系统的多线程程序设计的一套指导性注释（Compiler Directive）。OpenMP 支持的编程语言包括 C 语言、C++和 Fortran；而支持 OpenMP 的编译器包括 Sun Studio 和 Intel Compiler，以及开放源码的 GCC 和 Open64 编译器。OpenMP 提供了对并行算法的高层的抽象描述，程序员通过在源代码中加入专用的 pragma 来指明自己的意图，由此编译器可以自动将程序进行并行化，并在必要之处加入同步互斥以及通信。当选择忽略这些 pragma，或者编译器不支持 OpenMP 时，程序又可退化为通常的程序（一般为串行），代码仍然可以正常运作，只是不能利用多线程来加速程序执行[40]。

OpenMP 的这种灵活性非常适合一些不需要进行很大改动就能通过并行计算提高运行速度的场合。

3.3.2 在 3 维图形化用户界面中并行计算多个符号化电路指标

对于模拟电路中最常见的运放电路，目前 3 维图形化用户界面可以计算的电路指标有开环差模电压增益（A_{vo}）、开环带宽（f_H）、单位增益带宽（GBW，f_T）和相位裕度（PM）。

在调用 GRASS 得到符号化传输函数的表达式后，以上 4 种电路指标都可以互不影响地同步计算。下面的代码显示了如何使用 OpenMP 的 pragma 编译引导指令来实现多线程同步计算多种相互独立的电路指标的：

```
#include <omp.h>
#pragma omp parallel sections
{
#pragma omp section
{
    // 计算电路指标 1
}
#pragma omp section
{
    // 计算电路指标 2
}
// …
#pragma omp section
{
    // 计算电路指标 n
}
}
```

编译器读取 OpenMP 指令后生成的应用程序会在运行时自动检测当前计算机的CPU 信息，根据实际情况把由#pragma omp section 指令标注的代码块尽可能均匀地调配到空闲的 CPU 上，很方便地实现了代码块的并行计算。头文件 omp.h 包含了OpenMP 支持的函数声明。

对于不支持 OpenMP 的编译器，#pragma 语句会被编译器忽略，编译产生的应用程序就只能串行地依次计算这些电路指标，但最终的结果不会发生改变，从而在不改变代码的前提下可以同时支持并行和串行计算。

3.3.3 在 2 维图形化用户界面中并行计算传输函数的采样点

对于 2 维图形化用户界面，可以进行并行优化之处不多，但一些频繁执行的循环操作是可以很容易地并行化的，这样正好可以用最小的代码改动换取尽可能多的优化。其中反复调用符号化传输函数绘制其幅度和相位图像的循环最适合进行这种优化。

实际的代码结构非常简单：

```
#pragma omp parallel for schedule(static)
for (int i=0; i < num_points_; ++i)
{ // 计算传输函数在每个频率点的值 }
```

只需要在 for 循环前加入一行指令就能让编译生成的程序根据 CPU 核的个数自动将 for 循环划分为多段 for 循环，每段 for 循环的长度大约是 num_points_ / N，N 是 CPU 的核心个数。每段这样的循环被分配到一个核心上单独运行，到所有的小段循环全部计算完成后，程序的多个线程又重新汇合到一起，继续执行其它代码。

实验表明，对于传输函数中电路参数比较多，而且用户需要绘制清晰度较高的图像时，并行计算确实会带来明显的速度提升。

3.4 本章小结

本章详细介绍了 2 维和 3 维图形化用户界面的设计思路和实现方案。所有的设计方案都从电路设计者的角度出发，从多种角度用尽量简洁的方式把符号化仿真器复杂的分析结果呈现在用户面前，试图为用户提供最佳的设计体验。将当前研究的热点：多核心 CPU 并行计算引入到程序设计中，给程序的运行效率带来了不小的提升，也为今后的研究进行了初步尝试。

第 4 章　交互式图形化用户界面的应用案例

本章将着重介绍这套图形化用户界面的应用案例。

4.1 归一化梯度模的计算

3.2.3 节已经指出归一化梯度模作为一类优化问题标准的重要性。虽然符号化模拟电路分析的结果是某种程度上的近似，而且电路设计者往往只需要从电路指标与多个电路参数的曲面关系图上就能判断出某个电路参数对于特定指标的灵敏度，直观上难以理解的归一化梯度模的高精度计算仍然能够为电路设计者的设计方案提供另一方面有力的验证。

3 维图形化用户界面不仅能分析符号化电路传输函数，也能够处理普通数学函数的图像，并进行相关计算。Mathematica[41]是当前比较成熟的一款数学分析工具。与它的结果进行比较可以验证 3 维图形化用户界面在算法上的正确性。

以数学函数(4.1)为例来展示并验证 3 维图形化用户界面计算梯度值的正确性。

$$z=\frac{1}{2x^2+3y^2+1}. \tag{4.1}$$

首先定义归一化灵敏度(4.2)

$$\mathrm{Sens}(f,x)=\frac{x}{f}\frac{\partial f}{\partial x}. \tag{4.2}$$

归一化梯度的定义为

$$\boldsymbol{\Phi} f=\big(\mathrm{Sens}(f,x),\mathrm{Sens}(f,y)\big), \tag{4.3}$$

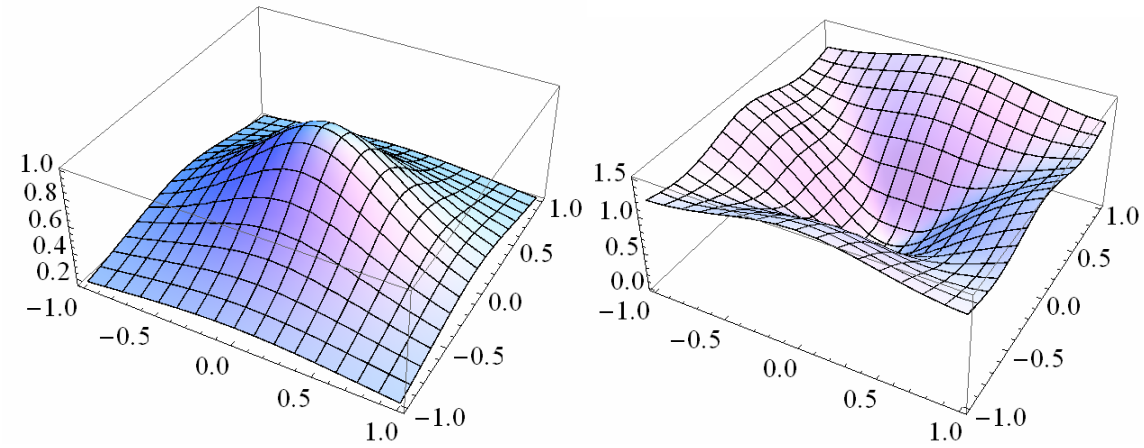
从而归一化梯度的模为

$$\|\boldsymbol{\Phi} f\|=\sqrt{\left(\frac{x}{f}\frac{\partial f}{\partial x}\right)^2+\left(\frac{y}{f}\frac{\partial f}{\partial y}\right)^2}. \tag{4.4}$$

对于(4.1)，其归一化梯度的模为

$$\|\boldsymbol{\Phi} z\|=\frac{2}{1+2x^2+3y^2}\sqrt{4x^4+9y^4}. \tag{4.5}$$

用 Mathematica 得到的(4.1)和(4.5)的图像如图 4-1 所示。

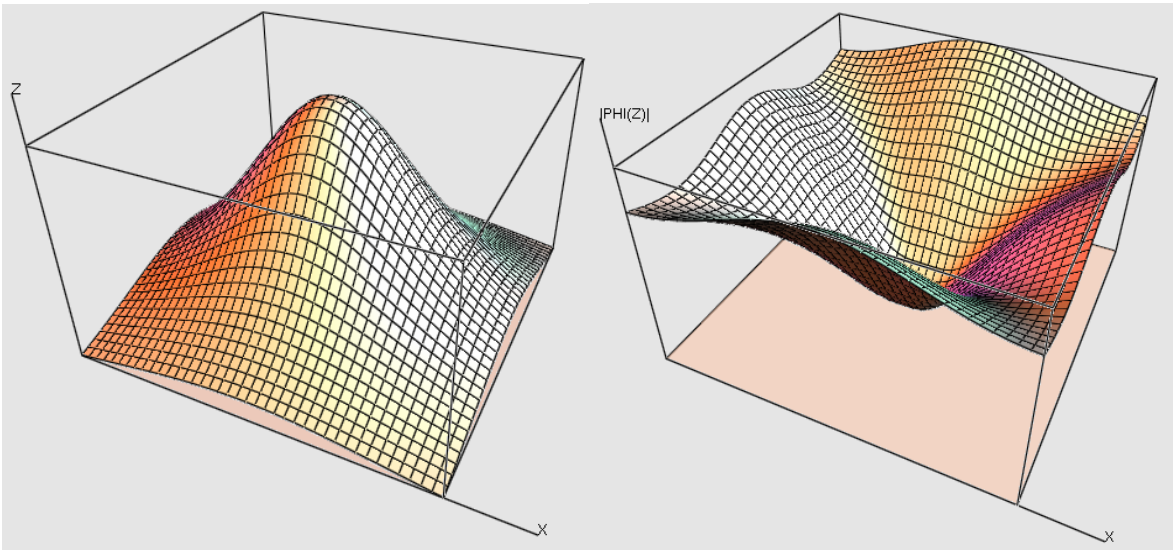


(a)　　(b)

图 4-1　(a) Mathematica 绘制的曲面(4.1)的图像;
(b) Mathematica 绘制的曲面(4.1)的归一化梯度模的图像

Fig. 4-1　(a) Surfaces of (4.1) from Mathematica;
(b) Surfaces of the normalized gradient modulus from Mathematica

用3维图形化用户界面得到的关于同样函数的曲面(4.1)和用数值微分直接从(4.1)计算得到的归一化梯度模的图像如图 4-2 所示。



(a)　　(b)

图 4-2　(a) 3 维图形化用户界面绘制的曲面(4.1)的图像;
(b) 通过数值方法得到的(4.1)的归一化梯度模的图像

Fig. 4-2　(a) Surface of (4.1)

(b) Surface of the normalized gradient modulus of (4.1) evaluated by numerical methods from 3D GUI

比较图 4-1 和 4-2 的图像，除了显示的比例有所不同外，二者并没有显著差异。这验证了数值微分算法的正确性。

4.2 图形化用户界面结合 GRASS 使用的案例

本节我们以图4-3所示三级运放电路为例说明三维图形界面在电路参数优化中的应用。模拟电路设计基本常识告诉我们，反馈网络中元件值的改变往往会对电路性能产生较大的影响。

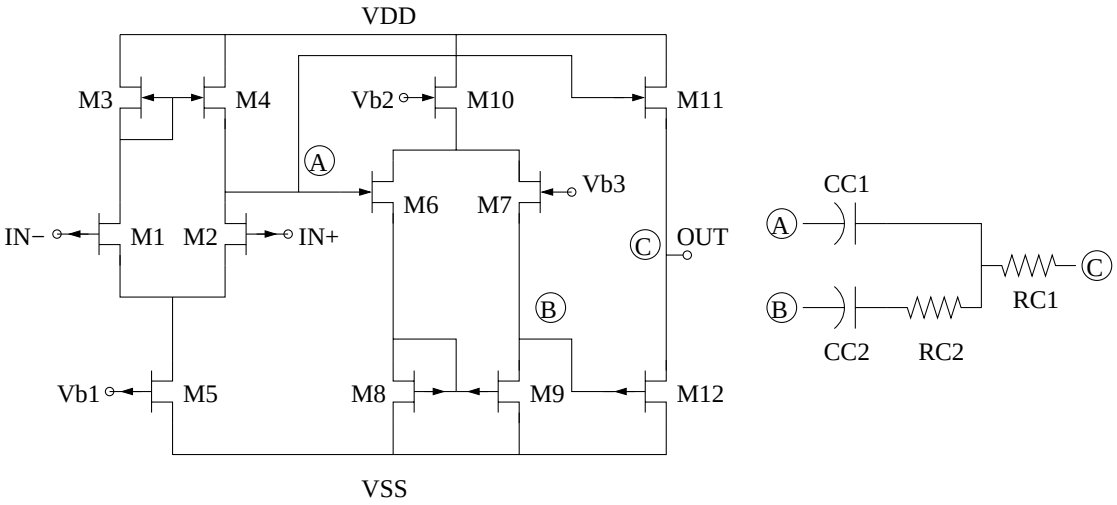


图 4-3　一个含 RC 反馈网络的 3 级运算放大器电路[30]

Fig. 4-3　A 3-stage operational amplifier with an RC feedback network[30]

此电路的特殊结构导致一些性能指标如相位裕度（Phase Margin）非常敏感地依赖于级间反馈器件 C_{C1} 和 R_{C1}。为此设计者需要了解这一对参数如何对相位裕度发生协同影响，并选择一个特殊的组合以在保证相位裕度（Phase Margin）PM_0 要求的前提下获得最大的设计良率。这意味着我们需要建立相位裕度关于参数 C_{C1} 和 R_{C1} 的一个二元函数关系 $PM(C_{C1}, R_{C1})$，最好还能求出二元函数 $PM(C_{C1}, R_{C1})$关于参数 C_{C1} 和 R_{C1} 的梯度（敏感度），从而选择敏感度较小的参数组合，使得电路具有比较好的抗干扰能力。对于这样的设计需求，传统的数值仿真工具不通过反复而冗长的计算是无法得到的，最终绝大多数设计者会放弃这样一种选择。而 GRASS 工具由于其特殊的核心设计和快速反复计算能力，可以轻易地解决此类设计问题。再加上友好的交互式图形化用户界面的支持，使电路设计者在短时间内处理多参数设计优化问题成为可

能。

对于图 4-3 所示的运放电路，给定约束条件：PM = 60°，GBW≥15MHz，$C_{C1} = C_C$，$C_{C2} = 4C_C$，$R_{C1} = 120R_C$，$R_{C2} = R_C$。

首先利用 2 维图形化用户界面，用户通过水平拖拽控件反复调节反馈器件参数，能够迅速地确定这两组参数的大致范围，如图 4-4 所示。取 0.5pF≤C_C≤10pF，4Ω≤R_C≤50Ω。

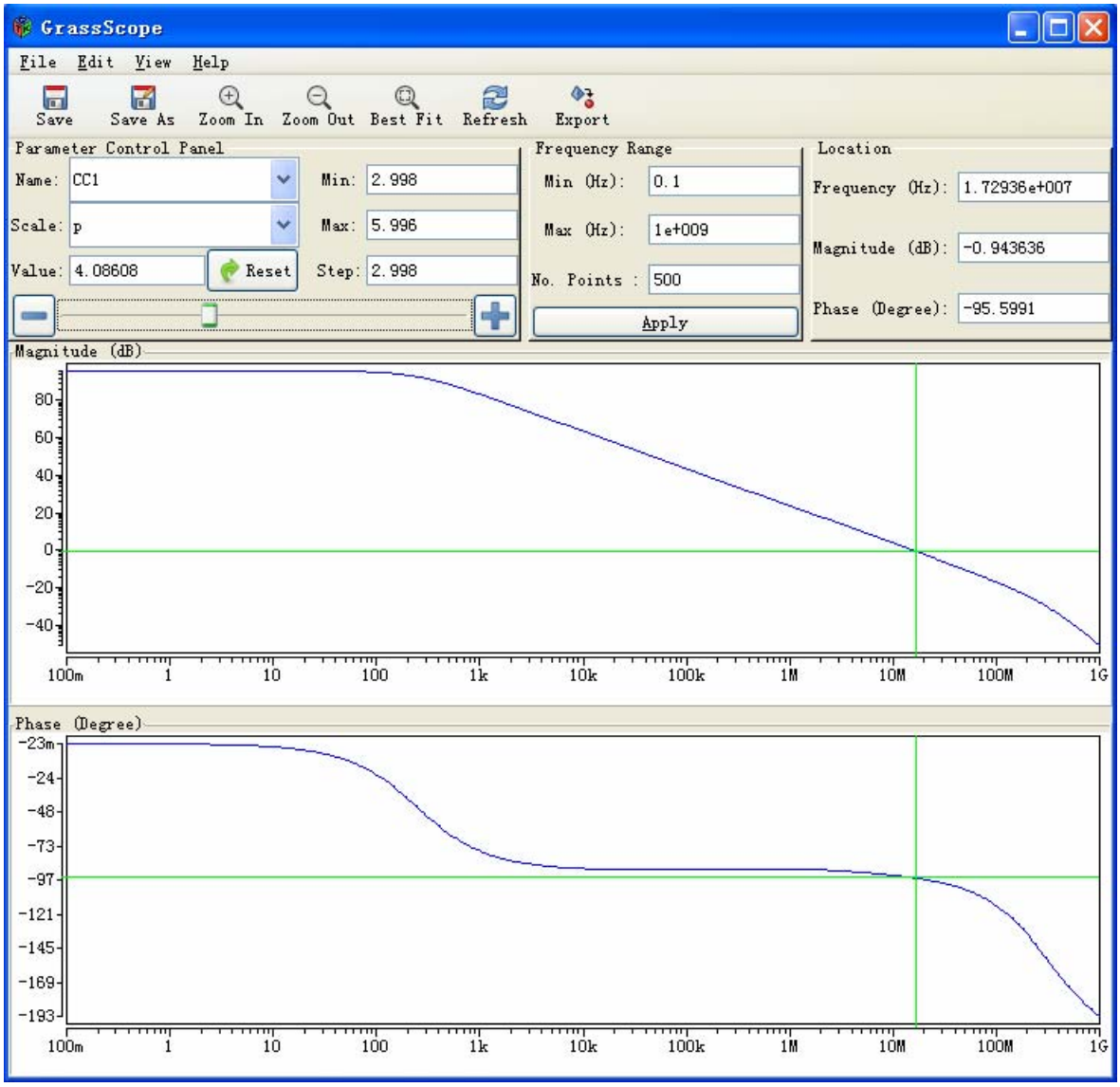


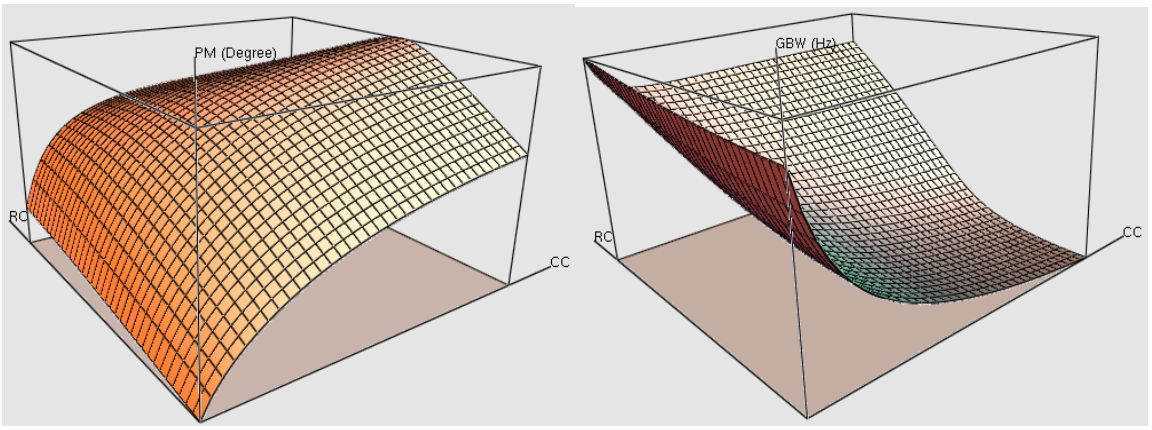
图 4-4　使用 2 维图形化用户界面快速确定主要参数取值范围

Fig. 4-4　Find the range of main parameters fast using 2D GUI

相位裕度（PM）和单位增益带宽（GBW）是运放电路中最重要的一些指标，因

此我们先用图形界面展示一下反馈回路中四个元件 C_{C1}，C_{C2}，R_{C1}，R_{C2} 对于这两个指标的影响情况，如图 4-5。

图 3-16 左边所示的等高线窗口允许用户用鼠标选取曲线上的任何点，并实时计算出该点的两个归一化灵敏度和归一化梯度的模。用户选取的点和等高线也会同步出现在空间曲面上。用户通过观察所取点处曲面的“坡度”就能大致判断出该点处系统性能的抗干扰程度。将不同指标对应的等高线叠加到同一窗口中就很容易找到满足各种约束条件的设计方案。



(a)　　(b)

图 4-5　PM 与(C_C, R_C)图像(a)以及 GBW 与(C_C, R_C)的图像(b)

Fig. 4-5　Surface (a) of PM vs. (C_C, R_C) and surface (b) of GBW vs. (C_C, R_C)

用户只需要用鼠标点击即可得到图 4-6 和图 4-7 所示的图像。图 4-6 中黄线上的点表示满足 PM = 60° 的所有（C_C，R_C）组合，灰色区域内的点（C_C，R_C）满足 GBW ≥15MHz。黄色曲线上位于灰色区域中的点（C_C，R_C）都满足所有约束条件。我们希望找到这段曲线上使系统性能最稳定的点。

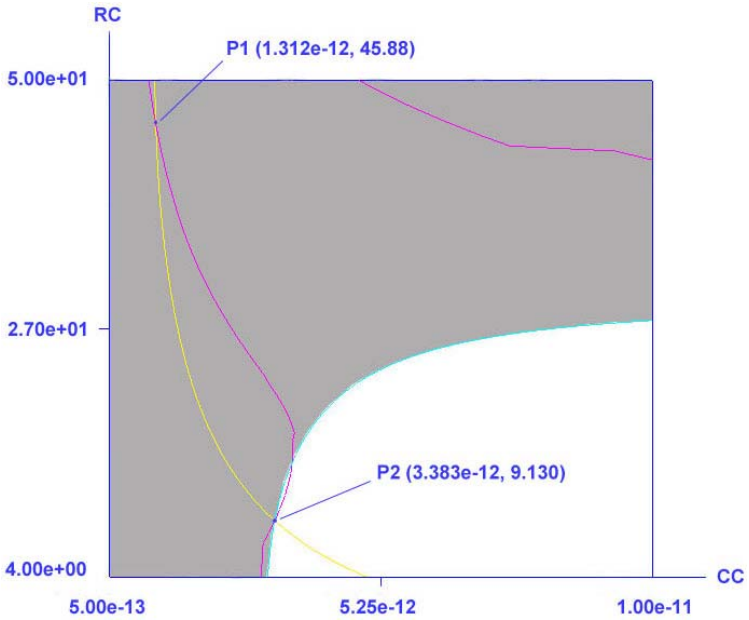


图 4-6　(C_C, R_C)平面上的若干约束区域

Fig. 4-6　Several constraint areas on the (C_C, R_C) plane

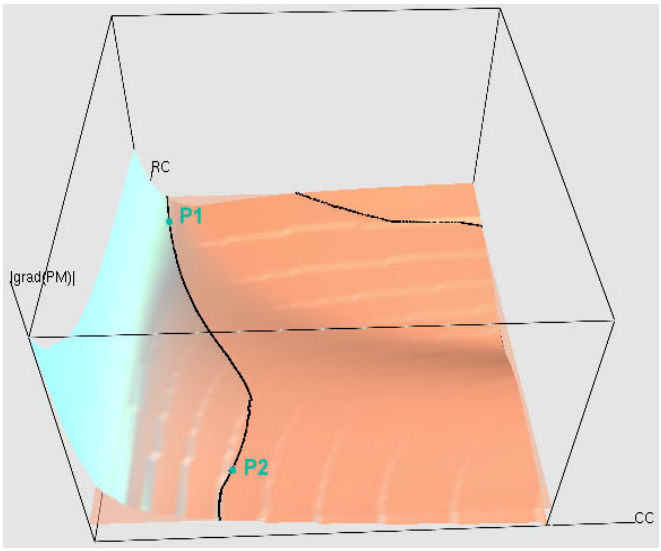


图 4-7　PM 归一化梯度模曲面上的一组等高线

Fig. 4-7　A group of contour lines on the normalized gradient modulus surface of PM

相位裕度（PM）能够表征运放电路的稳定性，所以最佳的设计参数应当使相位裕度随参数变化的敏感度最低，可以用|**Φ**PM|来描述相位裕度的灵敏度。

图 4-6 中的两条紫色曲线与图 4-5 中的两条黑色轮廓线是绘制在不同窗口中的同一组等高线。它们是使|**Φ**PM|数值相等的所有点(C_C，R_C)的集合。在图 4-7 中拖动鼠

标选取不同的等高线平面（也就是用鼠标调节平面$|\boldsymbol{\Phi}\mathrm{PM}| = \Phi_0$中的常数$\Phi_0$）就能在图 4-6 中实时地看到紫色曲线的变化情况。

图 4-6 和 4-7 中的点 P_1 和 P_2 是满足所有约束条件的(C_C，R_C)组合中，使$|\boldsymbol{\Phi}\mathrm{PM}|$最小的点。从图 4-7 还能看出 P_1 和 P_2 附近的灵敏度分布情况：虽然二者有相同的灵敏度，但 P_1 附近的曲面会随 C_C 的减少而剧烈上升，很可能因为实际制造工艺引起的 C_C 的偏差而造成产品系统稳定性的急剧下降；而 P_2 附近的曲面则较为平滑，所以选择 P_2 作为最终设计方案。

为了用传统的数值分析方法验证设计结果，我们使用已经成为业界标准的数值电路仿真器 HSPICE 对图 4-6 中包括 P_1 和 P_2 在内的满足约束条件的若干点进行蒙特卡洛分析（参数采用 Gaussian Distribution，期望值μ为设计值，标准差σ为$2\mu/3$，每组参数进行 3000 次分析），以模拟实际生产中由工艺偏差等随机因素引起的参数偏移对生产良率的影响。比较结果如表 4-1 所示，其中的生产良率是指满 足约束条件 45°≤PM≤85°的产品在所有产品中所占的比例。

表 4-1　图 4-6 中满足所有约束条件的部分(C_C, R_C)组合与 HSPICE 蒙特卡洛分析结果的比较

No.	C_C (pF)	R_C (Ω)	$\|\boldsymbol{\Phi}\mathrm{PM}\|$	相位裕度（PM，°）	单位增益带宽（GBW，MHz）	生产良率（HSPICE）（45°≤PM≤85°）	说明
1	1.312	45.88	0.562	60.1	36.8	93.2%	P_1
2	1.464	34.52	0.693	60.1	30.8	74.5%	
3	1.833	23.92	0.729	60.1	24.7	83.4%	
4	2.371	16.32	0.684	60.0	20.0	90.6%	
5	3.383	9.130	0.562	59.9	15.0	97.7%	P_2

表中数据说明：$|\boldsymbol{\Phi}\mathrm{PM}|$越小，电路越能承受由制造工艺偏差带来的影响，生产良率也相对较高。最终设计方案（P_2）确实能够得到较高的生产良率。

除了可以利用相位裕度的灵敏度对系统稳定性进行优化外，利用这套工具还能用类似方法对其他指标单独进行优化或是若干指标同步优化。

4.3 本章小结

本章通过实验验证了交互式图形化用户界面结合符号化仿真器 GRASS 的设计结果的可靠性，也展示了使用这套工具比传统数值仿真器具有更短的设计周期，为模拟电路设计者提供了的一种新设计思路。

第 5 章 全文总结

5.1 主要结论

本文介绍了用 GTK+和 OpenGL 实现的基于符号化模拟电路仿真器 GRASS 的 2 维和 3 维图形界面的开发和应用。它以模拟电路设计者易于理解的图形方式实时直观地展示出不同电路设计指标关于多组电路参数的变化情况，为电路设计者提供了一个能灵活进行参数选取的互动界面，并能提供多个电路设计指标相互约束情况下的参数优化途径，突破了传统数值电路仿真工具在类似应用中的局限性。

这套工具与 GRASS 结合使用，可应用于模拟集成电路高可靠性设计、生产良率提升，以及辅助芯片面积、功耗等多目标性能优化，能有效地缩短设计周期，提高电路抵御工艺偏移的能力，降低生产成本，为模拟集成电路设计提供了一种新的设计理念。

5.2 研究展望

更深入的研究可从以下方面入手：

(1) 进一步发掘符号化分析结果的应用领域，为模拟集成电路设计提供更为广泛的支持；
(2) 采用更强大的 3 维图形库（如 Qt+OpenGL）编写跨平台的图形化用户界面；
(3) 充分发挥多核 CPU 共享内存的资源优势，利用并行算法进一步加速计算过程；
(4) 对符号化分析结果中不需要频繁变动的部分进行局部数值化，以避免符号化表达式求值过程中大量的冗余计算，可以极大地提升运算效率。

参 考 文 献

[1] L.W. Nagel, "SPICE2: A computer program to simulate semiconductor circuits," Ph.D. dissertation, Univ. California, Berkeley, CA, May. 1975.
[2] L. Chua and P. Lin, "Computer-Aided Analysis of Electronic Circuits: Algorithms and Computational Techniques, " Englewood Cliffs, NJ: Rentice-Hall, ch. 14, 1975.
[3] P. Lin, "Symbolic Network Analysis," Amsterdam, the Netherlands: Elsevier, 1991.
[4] G. Gielen, P. Wambacq, and W. Sansen, "Symbolic analysis methods and applications for analog circuits: A tutorial overview," in proceedings of the IEEE, vol. 82, pp. 287-303, Feb. 1994.
[5] W. Sansen, G. Gielen, and H. Walscharts, "A symbolic simulator for analog circuits," in proceedings of the International Solid-State Circuits Conference, pp. 204-205, 1989.
[6] G. Gielen, H. Walscharts, and W. Sansen, "ISAAC: A symbolic simulator for analog integrated circuits," IEEE Journal of Solid-State Circuits, vol. 24, no. 6, pp. 1587-1597, Dec. 1989.
[7] F. Femandez, A. Rodriguez-Vazquez, and J. Huertas, "A tool for symbolic analysis of analog integrated circuits including pole/zero extraction," in proceedings of European Conference on Circuit Theory and Design, pp.752-761, 1991.
[8] F. Fernández, A. Vázquez, J. Huertas, "Interactive ac modeling and characterization of analog circuits via symbolic analysis," Analog Integrated Circuits and Signal Processing, vol. 1, pp. 183- 208, Nov. 1991.
[9] S. Seda, M. Degrauwe, and W. Fichtner, "A symbolic analysis tool for analog circuit design automation," in proceedings of the International Conference on Computer-Aided Design, pp. 488-491, 1988.
[10] S. Seda, M. Degrauwe, and W. Fichtner, "Lazy-expansion symbolic expression approximation in SYNAP," in proceedings of the International Conference on Computer-Aided Design, pp. 310-317, Nov. 1992.
[11] S. Manetti, "New approaches to automatic symbolic analysis of electric circuits," IEE proceedings G, pp. 22-28, Feb. 1991.
[12] G. Wierzba et al., "SSPICE-A symbolic SPICE program for linear active circuits," in proceedings of the 32nd Midwest Symposium on Circuits and Systems, vol. 2, pp. 1197-1201, Aug. 1989.
[13] A. Konczykowska and M. Bon, "Automated design software for switched-capacitor IC's with symbolic simulator SCYMBAL," in proceedings of the Design Automation Conference, pp. 363-368, 1988.
[14] M. Hassoun and P. Lin, "A new network approach to symbolic simulation of

large-scale networks," in proceedings of the IEEE International Symposium on Circuits and Systems, pp. 806-809, 1989.
[15] L. Huelsman, "Personal computer symbolic analysis programs for undergraduate engineering courses," in proceedings of the IEEE International Symposium on Circuits and Systems, pp. 798-801, 1989.
[16] C.-J. Shi and X.-D. Tan, "Canonical symbolic analysis of large analog circuits with determinant decision diagrams," IEEE Transactions on Computer-Aided Design *of* Integrated Circuits and Systems, vol. 19, no. 1, pp. 1-18, Jan. 2000.
[17] J. A. Starzky and A. Konczykowska, "Flowgraph analysis of large electronic networks," IEEE Transactions on Circuits and Systems, vol. 33, no. 3, pp. 302-315, Mar. 1986.
[18] G. Minty, "A simple algorithm for listing all the trees of a graph," IEEE Transactions on Circuit Theory, vol. 12, pp. 120, 1965.
[19] W. M. Zuberek et al. "Simulation-based parameter extraction - its implementation and some applications," in IEE Proceedings-G - Circuits, Devices and Systems, vol.141, no.2, pp.129-134, Apr. 1994.
[20] Z. Arnautovic and P. M. Lin, "Symbolic analysis of mixed continuous and sampled-data systems," in proceedings of the IEEE International Symposium on Circuits and Systems, Singapore, pp. 798-801, 1991.
[21] W. Chen, G. Shi, "Implementation of a symbolic circuit simulator for topological network analysis," in proceedings of IEEE Asia Pacific Conference on Circuit and System 2006, Singapore, pp.1327-1331, Dec. 2006.
[22] S. B. Akers, "Binary decision diagrams," IEEE Transactions on Computers, vol. C-27(6), pp. 509-516, Jun. 1978.
[23] C. Lee, "Representation of switching circuits by binary-decision programs", Bell System Technical Journal, vol. 38, pp. 985-999, Jul. 1959.
[24] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," IEEE Transactions on Computers, vol. C-35(8), pp. 677-691, Aug. 1986.
[25] K. Brace, R. Rudell, and R. Bryant, "Efficient implementation of a BDD package," in proceedings of 27th IEEE/ACM Design Automation Conference, pp. 40-45, Jun. 1990.
[26] S. Minato, "Zero-suppressed BDDs for set manipulation in combinatorial problems," in proceedings of 30th IEEE/ACM Design Automation Conference, (Dallas, TX), pp. 272–277, 1993.
[27] C.-J. Shi and X.-D. Tan, "Canonical symbolic analysis of large analogcircuits with determinant decision diagrams," IEEE Transactions on Computer-Aided Design, vol. 19, no. 1, pp. 1-18, Jan. 2000.
[28] G. Shi, W. Chen and C.-J. Shi, "A graph reduction approach to symbolic circuit analysis," Asia South-Pacific Design Automation Conference, Yokohama, Japan, pp. 197-202, Jan. 2007.
[29] G. Shi and X. Meng, "Variational analog integrated circuit design via symbolic

sensitivity analysis," International Symposium on Circuits and Systems, Taiwan, pp. 3002-3005, May. 2009.
[30] G. Palumbo, S. Pennisi, "Design Methodology and Advances in Nested-Miller Compensation," IEEE Transaction on Circuits and Systems-I: Fundamental theory and applications, vol. 49, no. 7, pp. 893-903, 2002.
[31] www.gtk.org
[32] www.gimp.org
[33] www.pidgin.im
[34] www.gnome.org
[35] www.gnu.org/copyleft/lesser.html
[36] www.jamesh.id.au/software/libglade
[37] library.gnome.org
[38] en.wikipedia.org/wiki/Name_mangling
[39] www.opengl.org
[40] www.openmp.org
[41] www.wolfram.com

符号与标记（附录 1）

C	电容（Capacitor）
BW	带宽（Band Width）
GBW	增益带宽积（Gain-Band Width）
grad, ∇	梯度算子（Gradient Operator）
GRASS	图约化模拟电路符号化仿真器（Graph Reduction Analog Symbolic Simulator）
PM	相位裕度（Phase Margin
R	电阻（Resistor）
Sens	归一化灵敏度（Normalized Sensitivity）
μ	期望值（Expectation）
σ	标准差（Standard Deviation）
$\mathbf{\Phi}$	归一化梯度算子（Normalized gradient operator）
$\vartriangle$	三角形（Triangle）

用 Makefile 管理代码（附录 2）

1. 自动生成 C++源代码文件（.cpp 文件）与头文件（.h 文件）间的依赖关系

下面的 Makefile 脚本可以在编译代码前自动生成并包含源代码文件和头文件间的依赖关系，这大减少了代码管理的复杂性，避免了软件开发过程中一些不必要的麻烦：

```
ifneq "$(MAKECMDGOALS)" "clean"
ifneq "$(MAKECMDGOALS)" "distclean"
sinclude $(PREREQS)
endif
endif
%.d : %.cpp
  $(CC) -MM $< $(CFLAGS) $(addprefix -I, $(INCLUDE_DIRS)) > $@.$$$$; \
  sed 's,\($*\)\.o[:]*,\1.o $@:,g' < $@.$$$$ > $@; \
  $(RM) $@.$$$$
```

2. 使用层次化的 Makefile 管理代码

把具有不同功能的代码分开放置于不同的目录下会给代码管理和维护带来不少方便。使用 make 的--directory 指令可以进入不同的目录分别编译代码。

使用下面的 Makefile 脚本可以将当前目录下的所有.o 文件链接生成临时静态库，以简化多层 Makefile 管理代码的复杂性：

```
AR := ar
ARFLAGS := rus
$(LIB) : $(patsubst %.cpp,%.o,$(wildcard *.cpp))
    $(AR) $(ARFLAGS) $@ $^
```

致　谢

首先需要感谢我的导师施国勇教授。从两年前刚进入微电子学院对研究方向的迷茫到今天完成这篇硕士学位论文，从当初只知道按个人喜好编写小程序到今天写下了成千上万行的代码，施老师始终引导着我沿着深入探索 EDA 科研领域问题的方向不断前进。在施老师的指导下解决一个个研究中遇到的问题时，我深刻地体会到了进行高水平科研工作的不易，同时也尽情地享受着在一次次失败之后每一次获得成功的喜悦。施老师对学术研究的一贯热情和严谨的态度在这两年间一直感染着我，让我能够充满信心地去面对种种困难和挑战。

感谢李章全老师对我的研究课题给予的大力支持。他对模拟电路的理解深度和他对体育运动的热爱程度一样，着实令我惊讶，让我在每一次与他的交谈中都受益匪浅。我的论文中涉及的大多数功能都是为了满足他从模拟电路设计者角度提出的需求而实现的。没有他的帮助，我就不会了解到资深模拟电路设计者对 EDA 工具最迫切的需求。

我必须感谢未曾谋面的陈微微师姐，我今天使用的符号化模拟电路仿真器引擎的代码绝大部分仍然来自她 2007 年前的工作。至今我仍然时常感叹她当初处理代码的能力之强，非常人所能比拟。

另外还要感谢谭焜元同学为我的研究课题提供了很多第一手的资料，还为我设计的图形化用户界面提出了不少建设性的方案。感谢孙亚男同学在我的研究课题起步阶段提供了第一个测试完整的设计案例，我确信这个案例将会永远保留在代码的测试目录中。感谢曾媚、王婷和陈硕同学在我刚开始学习使用 GTK+图形库时无私提供给我很多精妙的小例子，后来我的主体程序正是从这些例子出发逐步完善而成的。感谢郝志刚师兄和谢边村同学常常耐心和我一起讨论程序设计中的细节问题。在那些每天晚上很晚才离开实验室的日子里，他们总是我回寝室的伙伴，在路上谈论的一些话题也让我对学术和工作有了更多的感悟。感谢孟晓旋同学在我刚接触符号化模拟电路仿真器的时候耐心解答了我提出的无数琐碎的问题。

最后要感谢一直在背后支持着我的家人，感谢他们这些年来对我的一贯支持和鼓励，感谢他们和我一起分享了这一切。

攻读硕士学位期间已发表或录用的论文

[1] 李骥, 符号化模拟电路仿真器的图形界面开发与应用. 信息技术. 2010（3-5）（已录用）