**2024届本科生学士学位论文 学校代码：10269**



**集成电路中互连寄生测试结构设计及自动生成**

**Design and Automatic Generation of Interconnect Parasitic Test Structure in Integrated Circuit**

**姓 名： 许天一**

**学 号： 10202150422**

**学 院： 通信与电子工程学院**

**专 业： 微电子科学与工程**

**指导教师： 孙亚宾**

**职 称： 教授**

**2024年4月**

**华东师范大学学位论文诚信承诺**

本毕业论文是本人在导师指导下独立完成的，内容真实、可靠。本人在撰写毕业论文过程中不存在请人代写、抄袭或者剽窃他人作品、伪造或者篡改数据以及其他学位论文作假行为。

本人清楚知道学位论文作假行为将会导致行为人受到不授予/撤销学位、开除学籍等处理（处分）决定。本人如果被查证在撰写本毕业论文过程中存在学位论文作假行为，愿意接受学校依法作出的处理（处分）决定。

承诺人签名： 日期： 年 月 日

**华东师范大学学位论文使用授权说明**

本论文的研究成果归华东师范大学所有，本论文的研究内容不得以其它单位的名义发表。本学位论文作者和指导教师完全了解华东师范大学有关保留、使用学位论文的规定，即：学校有权保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅；本人授权华东师范大学可以将论文的全部或部分内容编入有关数据库进行检索、交流，可以采用影印、缩印或其他复制手段保存论文和汇编本学位论文。

保密的毕业论文（设计）在解密后应遵守此规定。

作者签名： 导师签名： 日期： 年 月 日

目录

[摘要： I](#_Toc163569166)

[Abstract: II](#_Toc163569167)

[1. 绪论 1](#_Toc163569168)

[1.1 研究背景 1](#_Toc163569169)

[1.2 国内外现状 2](#_Toc163569170)

[1.3 后道寄生测试模型自动生成及寄生参数提取的意义 2](#_Toc163569171)

[2. 互连寄生参数 3](#_Toc163569172)

[2.1 寄生电阻 3](#_Toc163569173)

[2.2 寄生电感 4](#_Toc163569174)

[2.3 寄生电容 4](#_Toc163569175)

[3. Python GUI设计 4](#_Toc163569176)

[3.1 设计功能概述 5](#_Toc163569177)

[3.2 依赖软件包及介绍 6](#_Toc163569178)

[3.3 程序运行流程及分析 7](#_Toc163569179)

[4. 结果测试 12](#_Toc163569180)

[4.1 程序运行流程及分析 12](#_Toc163569181)

[4.2 模型版图展示 13](#_Toc163569182)

[5. 总结与展望 15](#_Toc163569183)

[参考文献 17](#_Toc163569184)

[附录 18](#_Toc163569185)

[致谢 26](#_Toc163569186)

**集成电路互连寄生测试结构设计及自动生成**

摘要：

随着集成电路产业的发展，深纳米工艺不断发展，片上器件的特征尺寸不断缩小，电路频率逐渐提高，即使尽可能考虑器件材料以及在布局布线中不断优化，集成电路中后道寄生，即互联线所带来的寄生对电路性能的影响日趋明显。因此，对集成电路互连寄生结构的搭建和测试逐渐成为集成电路制造中重要的一环。

同时，计算机软件的发展使集成电路设计和制造更加便利。其中，图形用户界面(GUI)的应用使用户可以直观地进行电路的搭建和测试，这项技术可以用于对集成电路互连寄生结构的搭建和自动生成。

本文设计结合Python下的GUI界面，通过读取文件夹内特定结构编写的模版文件，使用户可以在图形界面输入参数，自动生成skill脚本。用户可将skill脚本导入cadence中完成测试结构的搭建。本文设计允许用户自定义模型，同时，图形化界面可以直观地生成模型，为后道寄生测试模型的搭建和测试提供了流程上的优化。

**关键词：**深纳米工艺，GUI，skill，自动生成

**Design and Automatic Generation of Interconnect Parasitic Test Structure in Integrated Circuit**

Abstract:

With the development of the integrated circuit industry and the continuous development of deep nanotechnology, the feature size of devices continues to shrink, and the circuit frequency gradually increases. Even if device materials are considered as much as possible and optimization is continuously carried out in layout and wiring, the parasitic effects of interconnect lines on circuit performance in integrated circuits are becoming increasingly apparent. Therefore, the construction and testing of parasitic structures in integrated circuit interconnection has gradually become an important part of integrated circuit manufacturing.

Meanwhile, the development of computer software has made integrated circuit design and manufacturing more convenient. Among them, the application of graphical user interface (GUI) allows users to intuitively build and test circuits, and this technology can be used to build and automatically generate parasitic structures for integrated circuit interconnection.

This article designs a GUI interface based on Python, which allows users to input parameters and automatically generate skill scripts by reading template files written for specific structures in folders. Users can import skill scripts into cadence to complete the construction of testing structures. This article allows users to customize models, and the graphical interface can intuitively generate models, providing process optimization for the construction and testing of post parasitic testing models.

**Keywords:** Deep Nanotechnology, GUI, skill, Automatic generati

1. 绪论
   1. 研究背景

摩尔定律是计算机科学中一个著名的观察结论，最初由英特尔公司的创始人之一戈登·摩尔在1965年提出，预测在未来的集成电路上，晶体管数量将每两年翻一番。摩尔定律后来被修正为每18个月翻一番，反映了半导体行业的发展更加迅速。摩尔定律表明，随着技术的进步，我们可以在相同的物理空间内集成更多的晶体管，从而提高计算机的性能和存储容量。集成电路产业发展至今，摩尔定律一直准确预测了行业的发展。

随着深纳米技术的不断进步，半导体器件的尺寸不断缩小，在同一块芯片上所包含的晶体管数量逐渐增多，其表现为超大规模集成电路(Very Large Scale Integrated Circuit, VLSI)的出现，是指在芯片上集成超过10万个以上元件的电路。而如今的集成电路更是能容纳数十亿晶体管。诚然，深纳米技术的发展给芯片的性能上的提高带来了革命性的变化，但它也由于其逐渐多样复杂的工艺，让电路设计者面临着更大的挑战。由于特征尺寸的缩小，一些由于工艺和制造所产生的影响，如寄生效应，也随着深纳米工艺的发展不断放大，成为集成电路产业界不得不考虑的问题。

早期集成电路设计确实主要依赖于传统的手工制图方法，这种方法的效率相对较低，成本较高，且制造周期长，这在一定程度上限制了集成电路产业的快速发展。然而，随着中小规模集成电路的出现，人们对设计效率和精度的要求日益提高，传统的手工制图方法已无法满足这种需求。在这种情况下，计算机辅助设计(Computer Aided Design, CAD)技术开始被引入到集成电路产业中。CAD技术的引入，极大地提高了集成电路设计的效率、精度和可靠性。利用计算机强大的计算和图形处理能力，设计师可以更快速、更准确地完成集成电路的版图设计、布局布线、仿真验证等工作。

在测试集成电路寄生效应的影响时，计算机辅助设计也发挥了重要作用。寄生效应是指由于电路元件之间的相互作用而产生的非理想效应。其中，互连寄生，也被称为后道寄生，是指集成电路中互连线之间相互作用所产生的寄生效应。通过CAD工具，可以对电路进行精确的建模和仿真，进而预测和评估寄生效应对电路性能的影响，提高电路的性能和可靠性。因此，对于如何快速精准构建寄生模型，在CAD中如何提取互连寄生参数，成为了集成电路产业关注的重点问题之一。

* 1. 国内外现状

早在20世纪60年代，内部等效技术(integral-equation, IE)就应用于二维几何结构电容值的计算[1]，在20世纪70年代初，IBM公司首次提出关于互连参数提取的若干问题，引发了业界的广泛关注[2]。随着20世纪80年代左右化学气相沉积(Chemical Vapor Doping, CVD)和隔离层定义(Isolation Layer Definition, ILD)技术的出现，再到化学机械抛光技术(Chemical Mechanical Polishing, CMP)在产业界运用，集成电路逐步进入了多层金属互连的时代[3]。然而，直到90年代，在深亚微米工艺兴起之后，相关的算法研究和软件开发才开始逐渐被关注。进入21世纪，西方各国加快了对于后道寄生模型的研究工作。对于互连寄生效应的模拟出现在在美国硅谷的大部分EDA软件中。这一功能的实现需要包括数据分析、数理方程建立、计算机科学等多个学科进行交叉，并且随着集成电路复杂度的提高，模型也逐渐复杂。

国内对寄生抽取的相关研究起步较晚。中国科学院微电子研究所才首次提出后道互连寄生测试概念。随后，国内众多高校和科研机构也开展了相关技术研究。在电子设计自动化(EDA)领域的重要会议上，众多三维电容提取方法被提出，并在IEEE Transactions on Computer-Aided Design等期刊上发表了相关论文。基于这些研究，多家EDA公司研发了针对互连寄生参数提取的EDA测试工具。尽管寄生参数提取技术已趋于成熟，但在集成电路设计中，其提取和测试要求仍至关重要，需持续改进与提升。虽然研究热潮已减退，但相关研究依然具有重要意义。

* 1. 后道寄生测试模型自动生成及寄生参数提取的意义

在集成电路产业起步阶段，器件的性能如开关速度是影响电路性能的主要矛盾，此时的后道寄生效应对电路性能的影响几乎可以忽略不计，体现在设计领域即设计者往往把互连线看作是理想导线。但随着器件尺寸的缩小，器件的开关速度不断提高，电路的频率已经达到了千兆赫兹(kM Hz)。此时，器件的性能已经不是影响电路性能的主要矛盾，而是器件缩小导致互连线缩小所引起的更加显著的寄生效应成为影响电路性能的关键因素。并且随着特征尺寸的减小，互连线因素影响占总延迟的比例逐渐增大，寄生效应对电路的串扰和功耗也产生了严重影响[4]。并且随着电路的频率升高，互连线之间将出现很强的寄生耦合：互连线发热表明有分布电阻效应；线与地之间会出现漏电流表明存在分布漏电导；不同互连线之间电场的相互作用增强，体现其还具有分布电容效应，这时，耦合效应导致的噪声也开始明显[5]。

因此在当今集成电路领域，关于后道寄生参数的提取是值得研究的问题。在传统设计中，寄生参数测试通常依赖于手动建模和测试，这一过程繁琐且耗时，需要手动创建测试模型，设定测试参数，并逐一进行验证。随着集成电路规模的不断增大和复杂度的提升，这种手动测试方法难以高效和准确，引入计算机辅助测试进行模型自动生成成为了一种有效的解决方案。计算机自动生成测试模型可以极大地提高测试效率和准确性。通过一定算法和数据处理，计算机辅助软件能够自动分析模型特征，生成对应的脚本框架，并根据用户输入的数据生成相应的测试模型，不仅减少了手动操作的繁琐程度，还提高了测试的一致性和可重复性。此外，计算机辅助进行自动生成测试模型还具有更高的灵活性。它可以根据不同结构进行自定义。这使电路设计者能够更专注于电路的创新和优化，而不必过多关注繁琐的测试工作。

通过自动生成的后道寄生测试模型，制造商可以更有效地检验芯片的性能和可靠性，确保产品符合设计规格和质量要求。这有助于在进行大规模生产时，降低产品故障率，提高客户满意度，增强市场竞争力。

1. 互连寄生参数
   1. 寄生电阻

理想情况下，互连线的寄生电阻由材料和尺寸决定。理想条件下计算公式如下：

其中，为材料的电导率，l,w,h分别为导线的长度、宽度和高度。

不过随着集成电路的规模越来越大，互连线越来越复杂，理想条件的公式不再能准确描述互连线的电阻，出现一些非理想因素，如在铜互连线工艺中，由于阻挡层导致的互连线横截面积减小，增加了互连线的阻值[6]。并且，寄生电阻还会受到趋肤效应(Skin Effect)的影响。由于趋肤效应，在高频下，互连线的电阻还会上升，且互连线存在最大长度[7]。

* 1. 寄生电感

在以往集成电路设计过程中，往往忽略寄生电感的影响。在传统集成电路设计中，正因为采用了多种降低RC时间参数的方法，使得电感所引发的延迟效应更为突出。仿真数据表明，特别是在高速应用中，当信号边沿陡峭、导线较宽且间距较近时，传统的RC模型在模拟电路行为时表现出明显的不足[8]。

值得注意的是，电感对误差的敏感程度相较于RC模型来说较小，且寄生电感值受材料尺寸变化的影响也较小。这意味着，即使在电感值存在高达20%的误差时，电路的波形误差依然能够保持在较小的范围内。如果忽视寄生电感的影响，那么电路仿真和实际表现之间将会出现显著偏差[8]。

因此，在高频集成电路的设计和分析中，必须充分考虑电感效应，以确保电路性能的准确性和可靠性。同时，通过合理的材料选择和布线设计，可以在一定程度上控制寄生电感的大小，从而优化电路的整体性能。

* 1. 寄生电容

产业界和学术界很早就开始了针对寄生电容的研究。在互连线系统中，主要可分为交叉电容、对地电容和耦合电容。针对三维寄生电容的提取，主要有间接边界元法、直接边界元法、半解析法、MEI法[9]。

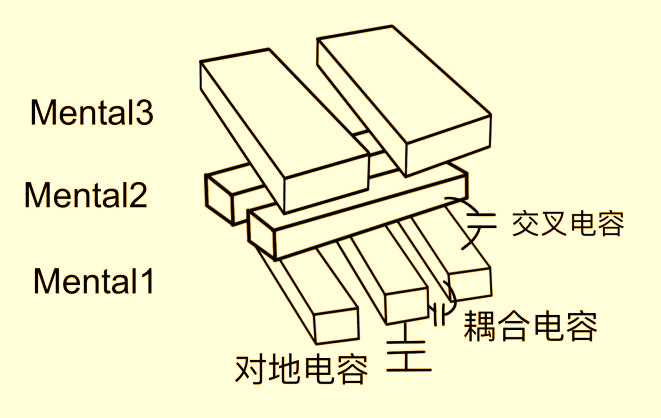


图2.1 互连线系统寄生电容示意图

Figure 2.1 Schematic diagram of parasitic capacitance of interconnect system

1. Python GUI设计

GUI[10][11]，即图形用户界面(Graphical User Interface)，表现为人机交互的界面显示格式，允许用户通过图形化的界面来操作程序。与早期的命令行界面(CLI)相比，GUI更为直观、易于理解和操作。

Python是一种解释型、交互式、面向对象的编程语言。其设计哲学强调代码的可读性，开发者可以用少量代码实现功能，让代码尽量易写和易读。Python语法简洁而清晰，具有丰富和强大的类库，非常适合快速开发应用。其代码规范也符合人类思维习惯，因此，Python又被称为“可读性最强的编程语言”。

* 1. 设计功能概述

程序的使用需要如下步骤：1、安装相关依赖：customtkinter；2、运行Python程序；3、输入所需参数；4、点击生成按钮，在output文件夹下生成.il文件。

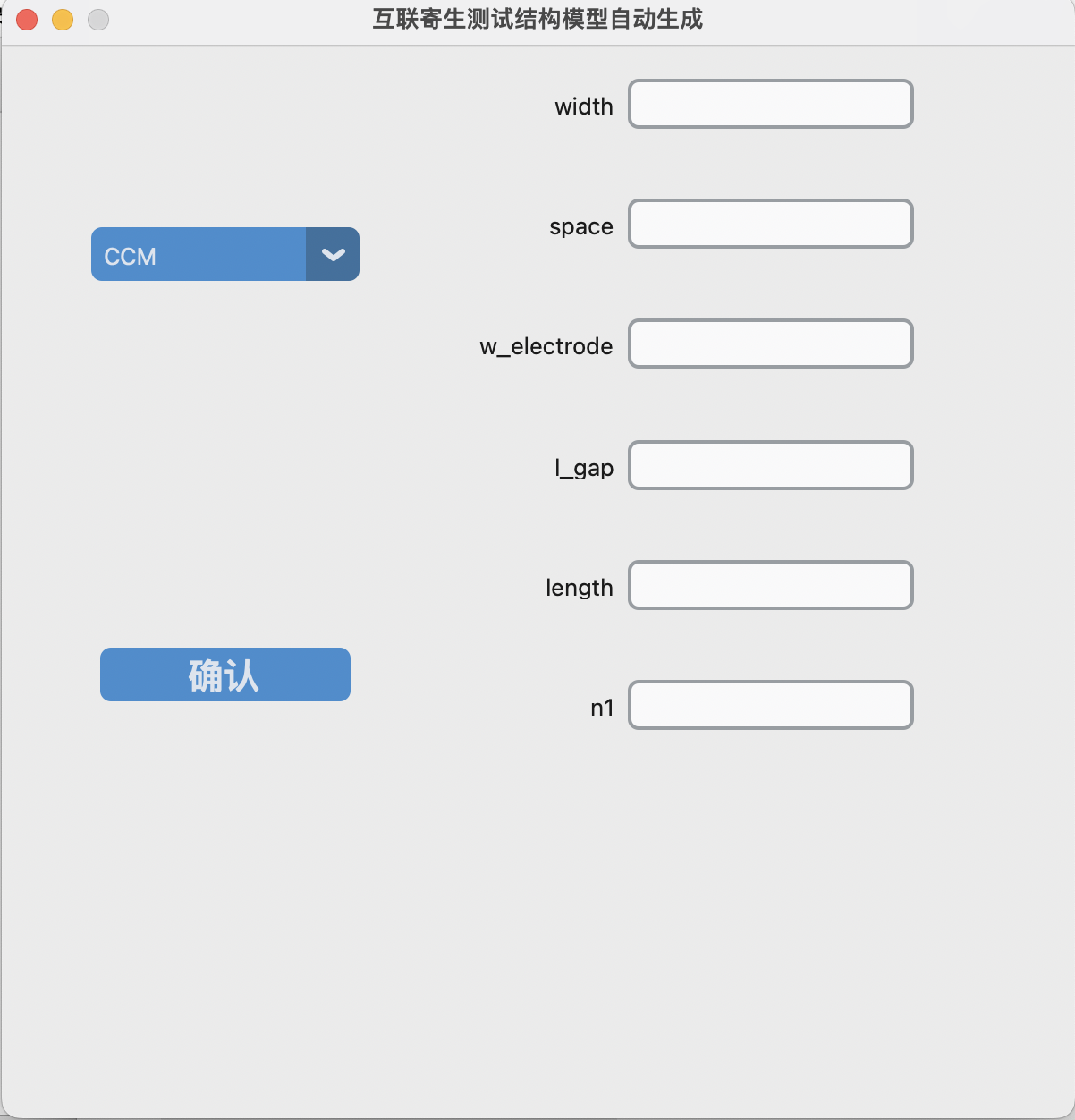


图3.1 GUI 界面

Figure 3.1 GUI interface

程序可以读取当前文件夹下template文件夹下的按照特定格式编写的用户自定义的.txt文件作为模版文件，在GUI界面生成模型选择多选框和参数输入框，程序可以根据用户输入的参数，运算模版文件中的表达式，自动生成需要计算的参数值，并填入程序内文本，在用户点击生成按钮后，程序在当前文件夹内的output文件夹内自动生成相应的.il文件。

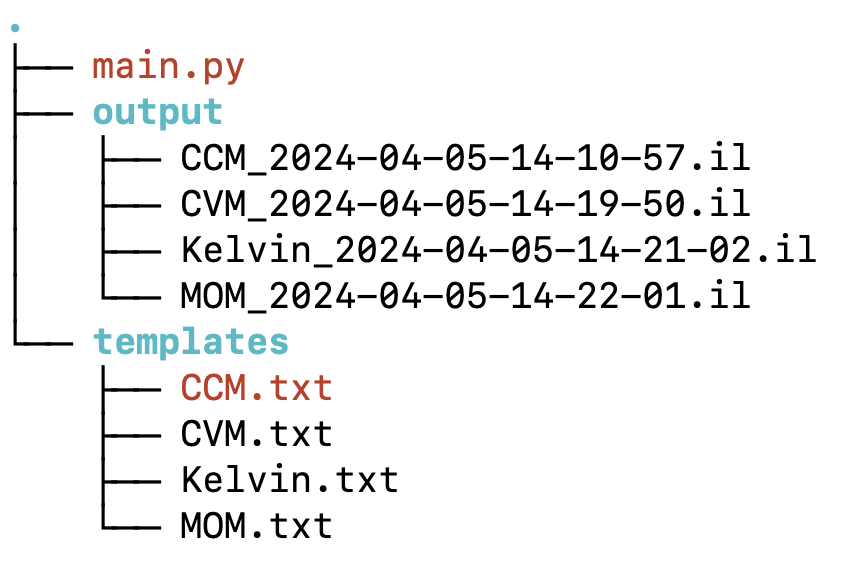


图3.2 文件夹结构

Figure 3.2 folder structure

在template文件夹内的.txt文本文件需要按照以下格式编写：将文本内容分为两部分，上半部分为json格式，下半部分为含有可替换信息的文本，实际上按照skill脚本语法编写。在json格式中，含有两组键值对：parameter和expression，分别对应着自动生成过程中用户需要输入的参数和根据用户输入参数值进行计算的函数。其中，parameter的值是一个列表，元素为需要用户输入的变量名，元素格式为字符串；expression的值是按照json格式书写，包含若干键值对：键为函数名，值为表达式的字符串形式。目前，表达式支持加(+)、减(-)、乘(\*)、除(/)、整除(%)和括号运算，操作数和运算符之间需要用空格隔开。

* 1. 依赖软件包及介绍

customtkinter：基于Tkinter的Python GUI库。它提供了完全可定制的部件，与 Tkinter相同，为开发者提供了一个强大、灵活的GUI库。

tkinter：Python的标准图形用户界面（GUI）工具包。它封装了Tk GUI工具包的接口，使得Python程序员能够使用Tk的部件来创建GUI应用程序。

json：软件包是一个用于处理JSON数据的标准库。提供的函数可以完成Python对象与json之间的解码与编码。

operator：提供了一系列用于执行各种操作的函数，用于实现Python中的内建操作符。在本程序中，operator用于进行表达式的堆栈计算。

os：Python的一个标准库模块，它提供了很多与操作系统交互的函数，如读取环境变量、操作文件和目录、执行系统命令等。程序中用于文件相关操作。

re：Python的正则表达式函数库，用于处理字符串中的模式匹配和搜索替换等操作。正则表达式(Regular Expression)是一种用特定语法来描述字符串的模式，实现对字符串的匹配、查找、替换等操作。程序中用来识别表达式中的变量名并替换成用户输入的数值。

time：模块提供了各种与时间相关的函数。函数功能包括获取当前时间、测量代码执行时间、格式化时间、将时间转换为特定格式的字符串等。程序中用于读取当前时间作为生成文件名的一部分。

* 1. 程序运行流程及分析

用户在安装完整的依赖后，开始运行程序。程序首先执行读取当前文件夹下template文件夹中所有的.txt文件作为模版文件。在程序中，按照如下的数据结构存储读取的信息：创建一个字典，将.txt的文件名（模版名）作为字典的值，将文件的内容作为值；文件内容上半部分为json格式，包括需要用户填写的变量名(parameter)以及根据用户输入变量值进行计算的表达式(expression)，下半部分为生成.il文件时包含可替换部分的文本。



图 3.3 文件读取函数部分代码

Figure 3.3 Partial code of file reading function

接下来，main函数中会运行app = App(filed)，将app定义为APP类，传入的参数filed为文件读取函数返回的字典。App类进行初始化时，会为每一个模版创建一个loadframe的类，这个类是Ctk的子类，也就是本程序中PythonGUI界面生成框架的类。App类在为模型创建loadframe类时，会创建frame\_dict字典，键为模版名，值为根据文本读取函数返回的字典生成的GUI框架（数据类型为<class '\_\_main\_\_.loadframe'>），包括输入框、执行生成文件命令的按钮等。创建的loadframe类应用于GUI界面的右侧，根据App类在窗口左侧的多选下拉菜单的选择值不同，选择显示窗口右侧的内容。App类的功能是使用户可以在GUI界面的左侧选择界面右侧的内容。

用户在GUI界面左侧选择使用的模版，在界面右侧的输入框填入需要的参数后，点击“生成”按钮。程序的关键是如何在用户选择框架后，在支持用户自定义模板的前提下，准确生成对应的.il文件，具体于程序设计上是准确的作出数据结构的转变。在本程序中，具体是由loadframe类中output\_file函数实现的，它是GUI界面中“生成”按钮所执行的命令。以下是output\_file函数的代码：

图3.4 output\_file函数代码

Figure 3.4 Code of output\_file function

函数中，self.parameter、self.expression和self.content分别是文件读取函数返回字典中的parameter、expression和content的值，函数创建parameter\_value字典作为存储变量值的字典，首先将文本读取文件返回字典中parameter中的变量作为键，输入框中用户输入的值作为值存入parameter\_value字典中。

随后函数开始处理expression中的表达式。self.expression的数据类型是字典，其中的键是函数名，值是其对应的表达式，为字符串格式，字符串表达式中的变量可以是用户输入的值（模版文件中的parameter的值），也可以是模版文件exprssion中已经计算过的函数名（函数按照文件中函数名定义顺序进行计算）。以下是一个模版文件的部分示例：

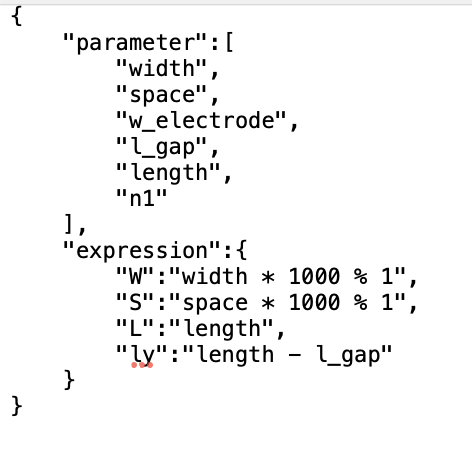


图3.5 模版文件json部分示例

Figure 3.5 Sample json portion of the template file

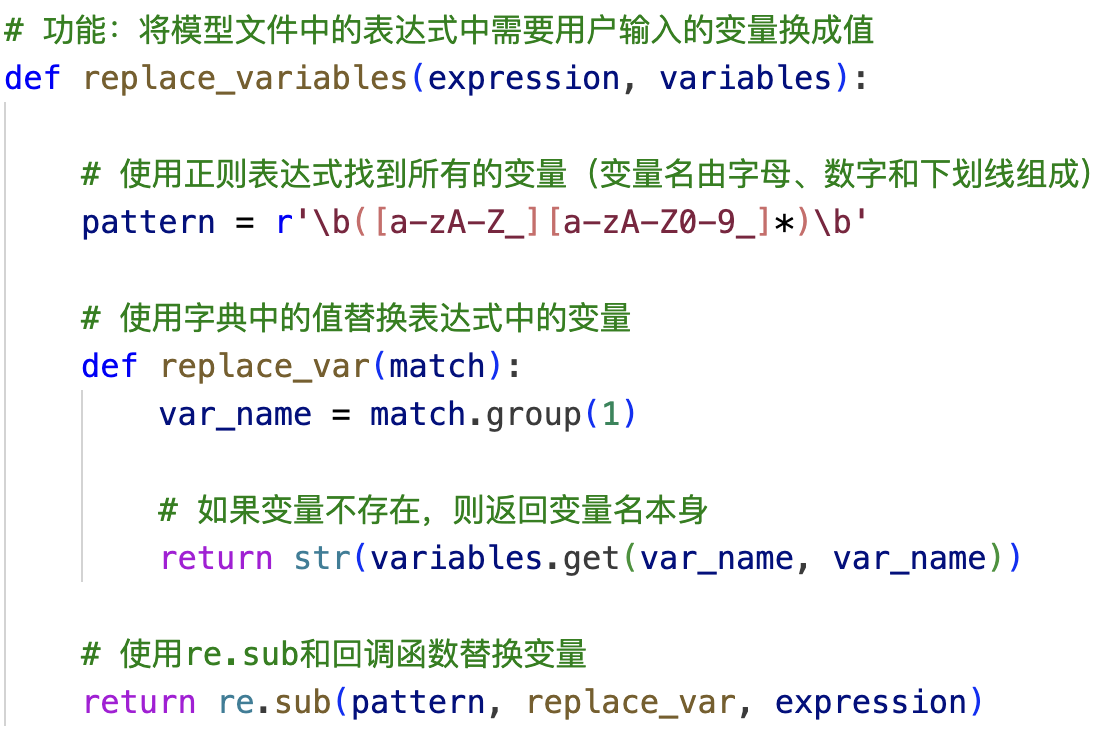
针对每个表达式，函数模仿计算机对于表达式的计算进行处理。先执行预处理器的功能，函数根据正则表达式将变量名替换成相应具体的值，按照字符串格式处理。

图3.6 函数变量替换函数

Figure 3.6 Function variable replacement function

replace\_variables将返回值传入infix\_to\_postfix函数，将中缀表达式转换成逆波兰表达式。按照如下规则处理：操作数直接压入输出栈，运算符优先级大于栈顶运算符优先级直接入运算符栈，反之则将栈内运算符顺序弹出压入输出栈，直到运算符栈空或遇到左括号，遇到左括号直接压入运算符栈，遇到右括号弹出，直到遇到左括号。

随后将infix\_to\_postfix函数的返回值传入计算函数evaluate\_postfix。计算函数根据运算符不同，执行不同的运算，直到栈内只剩下一个数，即表达式结果。evaluate\_postfix部分代码如下：

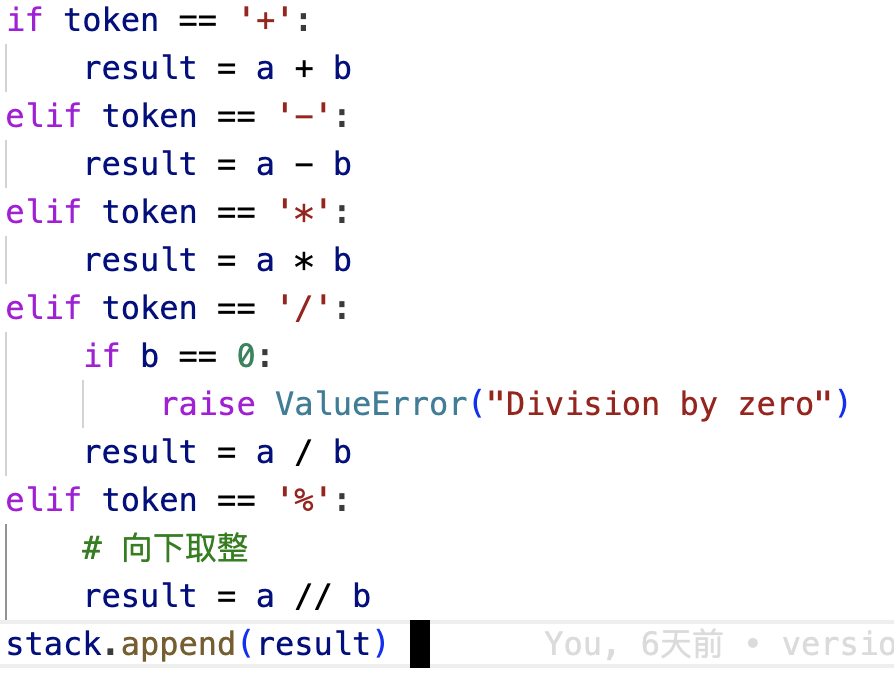


图3.7 evaluate\_postfix函数部分代码

Figure 3.7 Partial code of the evaluate\_postfix function

计算完成后，将结果也按照键值对的形式存入parameter\_value字典中，随后利用format方法将self.content文本中的可替换部分替换成对应变量的值，再写入.il文件中，实现测试结构的及自动生成。以下是程序的流程图：

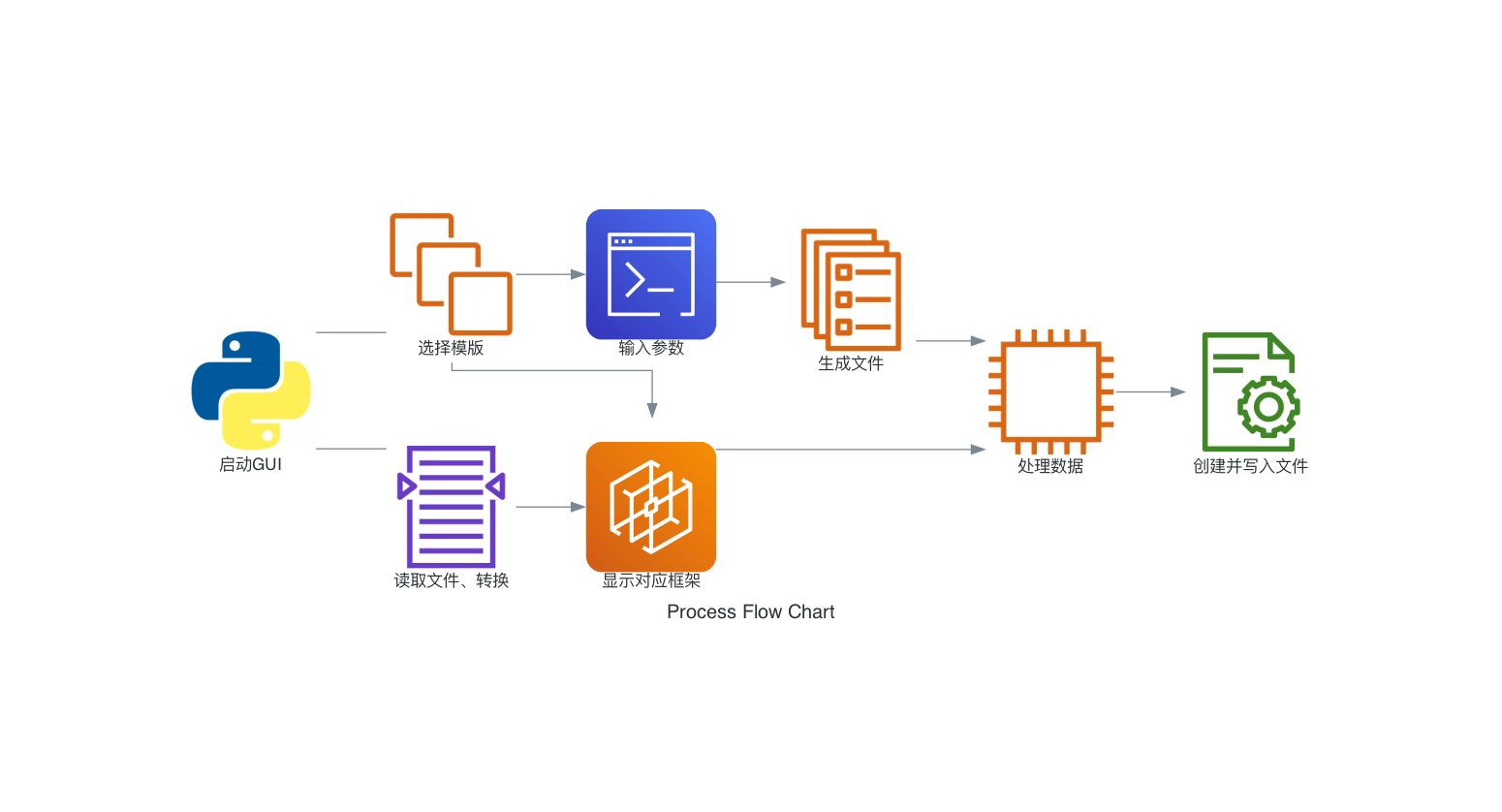


图3.8 程序流程图

Figure 3.8 Program flow chart

1. 结果测试
   1. 程序运行流程及分析

在virtuoso中，新建自定义库（名称与模版中名称相同）并导入相应的pdk库，其中示例CCM.il如下所示：

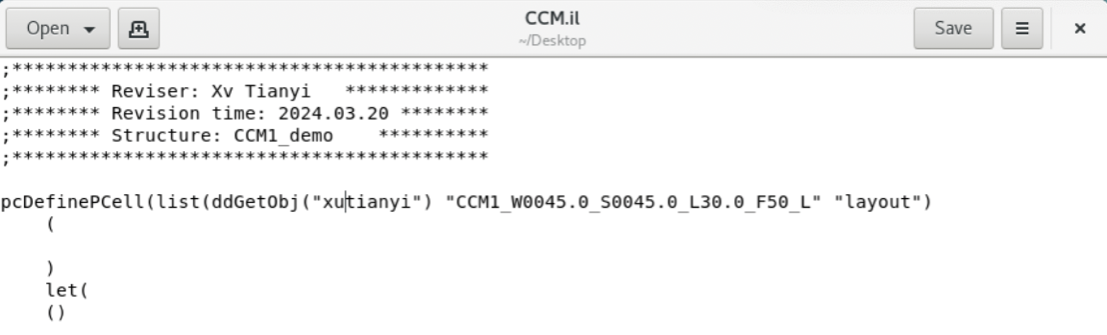


图4.1 CCM.il自建库名

Figure4.1 CCM.il self-built library name

随后在virtuoso界面输入load “path of .il”，相应的Cell就会生成在新建的库下：

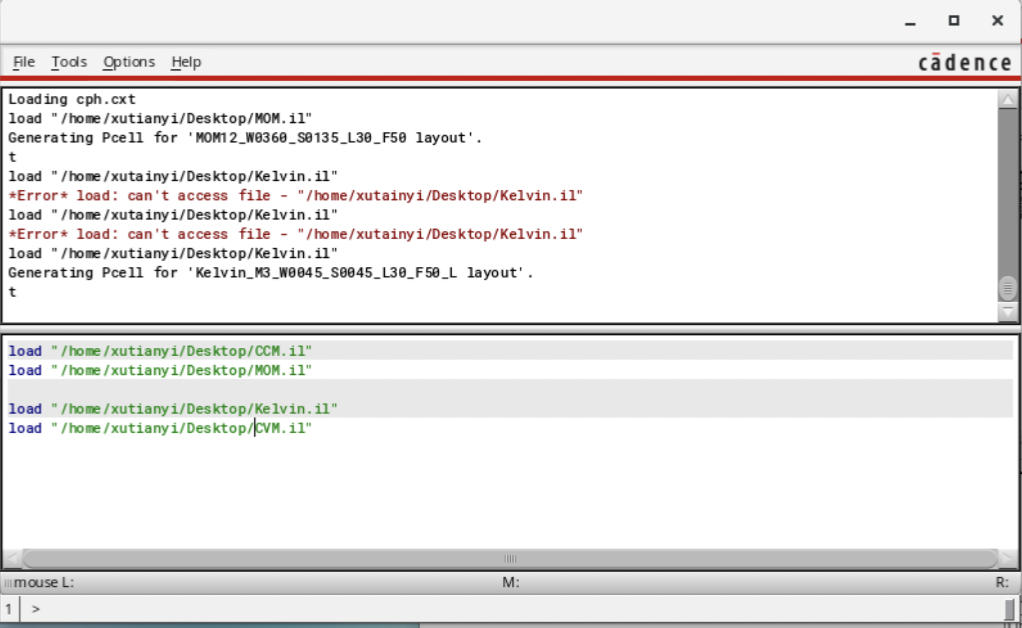


图4.2 加载.il文件

Figure4.2 load .il files

* 1. 模型版图展示

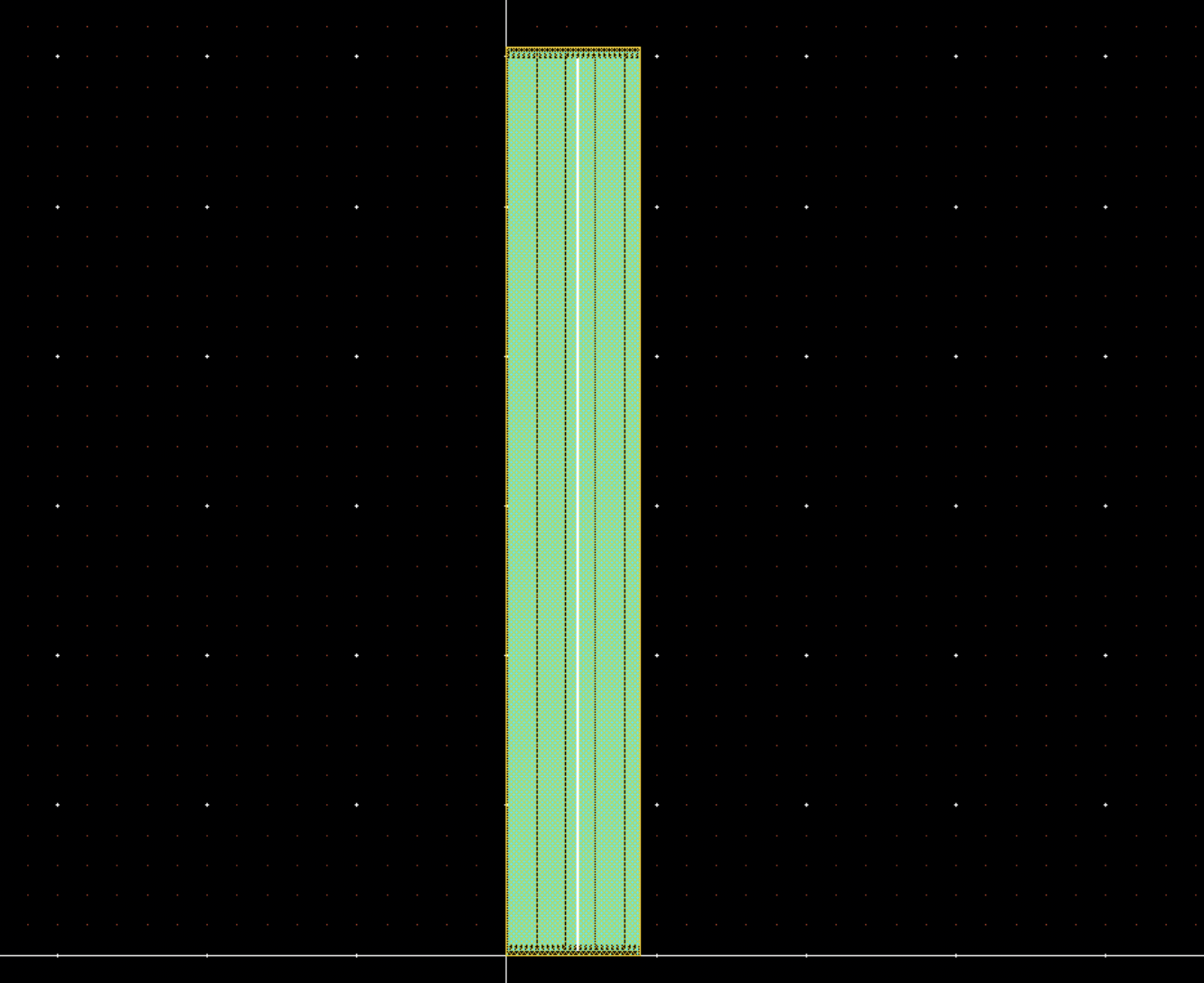


图4.3 CCM模型版图

Figure4.3 CCM model layout



图4.4 CVM模型版图

Figure4.4 CVM model layout

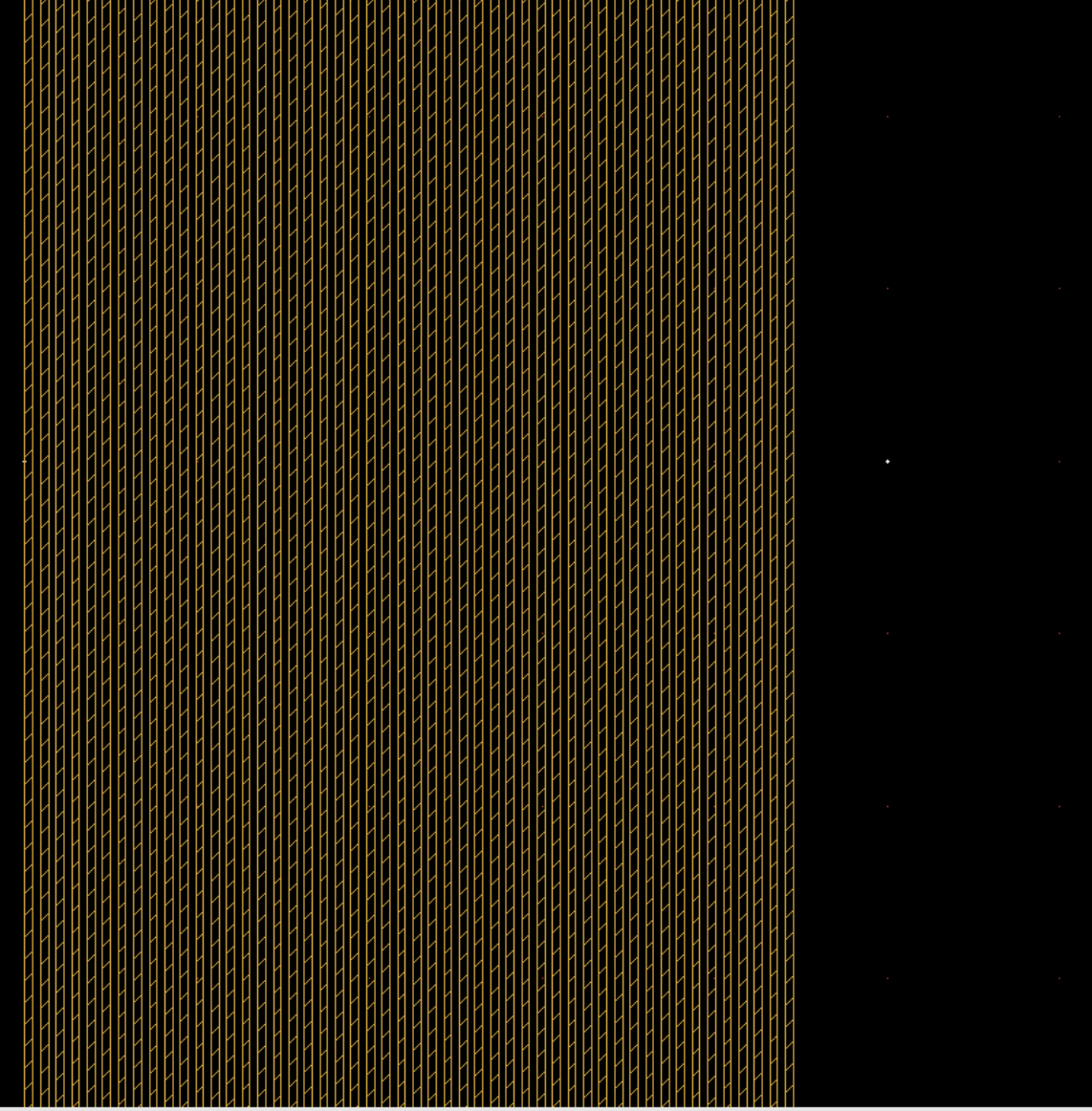


图4.5 Kelvin模型版图

Figure4.5 Kelvin model layout

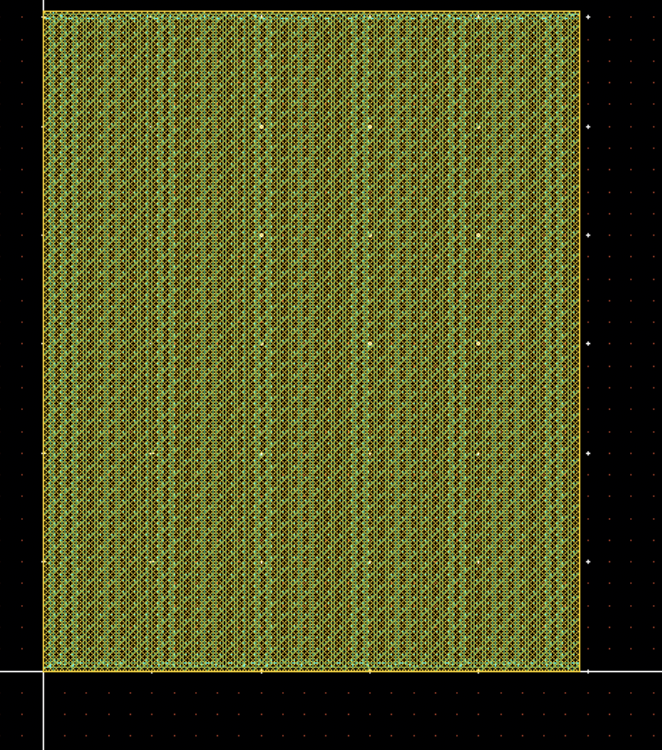


图4.6 MOM模型版图

Figure MOM model layout

1. 总结与展望

本文详细设计了一个具备自定义功能的互连寄生测试结构脚本生成工具，程序在集成电路设计领域具有应用价值，在后道寄生提取过程中，能够快速而准确地完成既定模型的搭建，大大提高工作效率。

从计算机辅助设计的理念出发，本设计充分利用了Python语言的强大功能和丰富的软件包，实现对skill脚本模版中可替换部分的快速替换。Python作为一种高级编程语言，具有简洁易读、易于维护的特点，其强大的第三方库支持则使得开发者能够轻松地实现各种复杂的操作。

通过GUI界面，用户可以便捷地选择模版、输入参数，并自动生成脚本文件。在程序的实现过程中，采用模块化的设计思想，将程序划分为多个独立的模块，不仅可以降低代码的耦合度，提高代码的可维护性，还可以方便地扩展程序的功能。此外，本设计还采用了面向对象的编程方法，通过定义类和对象来实现对程序功能的封装和复用。

通过本文所设计的程序，用户可以快速生成同一类型结构在不同参数下的版图，从而大大缩短了模型搭建的时间。同时，由于程序支持用户自定义模版，因此可以灵活地适应各种复杂的集成电路设计需求。

本设计也存在不足之处，如当用户只改变某一个或某几个参数进行测试时，需用重新输入所有参数，不具备输入记录保存的功能；生成的文件名默认为模型名加生成时间名，不支持用户自定义；程序生成的文件还需用户手动导入测试模型建模文件中，操作仍然繁琐。本设计仍有较大的改进空间，可以增加对输入记录保存、自定义生成文件名等功能，进一步提高测试模型搭建和自动生成的效率。

参考文献

1. D. W. Kammler, "Calculation of Characteristic Admittances and Coupling Coefficients for Strip Transmission Lines," in IEEE Transactions on Microwave Theory and Techniques, vol. 16, no. 11, pp. 925-937, November 1968, doi: 10.1109/TMTT.1968.1126828.
2. A. E. Ruehli and P. A. Brennan, "Efficient Capacitance Calculations for Three-Dimensional Multiconductor Systems," in IEEE Transactions on Microwave Theory and Techniques, vol. 21, no. 2, pp. 76-82, Feb. 1973, doi: 10.1109/TMTT.1973.1127927.
3. T. Smedes, N. P. van der Meijs, A. J. van Genderen, P. J. H. Elias and R. R. J. Vanoppen, "Layout Extraction of 3D Models for Interconnect and Substrate Parasitics," ESSDERC '95: Proceedings of the 25th European Solid State Device Research Conference, The Hague, Netherlands, 1995, pp. 397-400.
4. 喻文健,王泽毅.三维VLSI互连寄生电容提取的研究进展[J].计算机辅助设计与图形学学报,2003(01):21-28.
5. 张瑛. VLSI中互连线工艺变化的若干问题研究[D].南京理工大学,2007.
6. 陈宝君. 集成电路互连线电阻电感参数提取方法研究[D].大连理工大学,2013.
7. L. . -T. Hwang, D. Nayak, I. Turlik and A. Reisman, "Thin-film pulse propagation analysis using frequency techniques," in IEEE Transactions on Components, Hybrids, and Manufacturing Technology, vol. 14, no. 1, pp. 192-198, March 1991, doi: 10.1109/33.76531.
8. Y. I. Ismail, "On-chip inductance cons and pros," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 10, no. 6, pp. 685-694, Dec. 2002, doi: 10.1109/TVLSI.2002.808445.
9. K. K. Mei, R. Pous, Zhaoqing Chen, Yao-Wu Liu and M. D. Prouty, "Measured equation of invariance: a new concept in field computations," in IEEE Transactions on Antennas and Propagation, vol. 42, no. 3, pp. 320-328, March 1994, doi: 10.1109/8.280717.
10. 冯桂尔.基于Python的GUI(Tkinter)实例开发[J].信息与电脑(理论版),2023,35(01):175-178.
11. 李刚. Python:基于实例的GUI(Tkinter)编程[J]. 电脑编程技巧与维护,2022(6):7-9. DOI:10.3969/j.issn.1006-4052.2022.06.002.

附录

main.py

from customtkinter import CTk, CTkFrame, CTkOptionMenu, CTkButton, CTkLabel, CTkEntry, CTkFont

from tkinter import messagebox

import json

import operator as op

import os

import re

import time

# 功能：导入模版文件

def load\_file():

# 当前目录pwd

current\_dir = os.getcwd()

# 获得绝对路径

templates\_dir = os.path.join(current\_dir, 'templates')

template\_dict = {}

for filename in os.listdir(templates\_dir):

if filename.endswith('.txt'):

# 构建文件的完整路径

file\_path = os.path.join(templates\_dir, filename)

# 读取文件内容

with open(file\_path, 'r') as file:

content = file.read()

# 按照文件中空两行划分，上半部分为json，下半部分为文本

json\_part, template\_part = content.split('\n\n\n', 1)

# json读取

json\_data = json.loads(json\_part)

parameter = json\_data['parameter']

expression = json\_data['expression']

# 文本清除前后空白后写入程序

template\_str = template\_part.strip()

# 读取后存入第一级字典，内容为原始的值

template\_dict[filename.rstrip('.txt')] = {

'parameter': parameter,

'expression': expression,

'template\_str': template\_str

}

return template\_dict

# 功能：将模型文件中的表达式中需要用户输入的变量换成值

def replace\_variables(expression, variables):

# 使用正则表达式找到所有的变量（变量名由字母、数字和下划线组成）

pattern = r'\b([a-zA-Z\_][a-zA-Z0-9\_]\*)\b'

# 使用字典中的值替换表达式中的变量

def replace\_var(match):

var\_name = match.group(1)

# 如果变量不存在，则返回变量名本身

return str(variables.get(var\_name, var\_name))

# 使用re.sub和回调函数替换变量

return re.sub(pattern, replace\_var, expression)

# 功能：判断是否是运算符

def is\_operator(token):

return token in ['+', '-', '\*', '/', '%']

# 功能：定义运算符优先级

def get\_precedence(op):

precedences = {'+': 1, '-': 1, '\*': 2, '/': 2, '%': 2}

return precedences[op]

# 功能：支持浮点数，判断是否是数字

def is\_number(s):

try:

# 尝试将字符串转换为浮点数

float(s)

return True

except ValueError:

return False

# 功能：将中缀表达式转换成逆波兰表达式

def infix\_to\_postfix(expression):

# 定义了两个栈

# output：输出，stack：临时

output = []

stack = []

tokens = expression.replace('(', ' ( ').replace(')', ' ) ').split()

for token in tokens:

if is\_number(token):

# 操作数直接压入output

output.append(token)

elif is\_operator(token):

while stack and is\_operator(stack[-1]) and get\_precedence(stack[-1]) >= get\_precedence(token):

# 判断stack中优先级和待处理运算符优先级的关系

# 弹出最后一个运算符，直到stack中的优先级都小于待处理运算符

output.append(stack.pop())

stack.append(token)

elif token == '(':

# 遇到左括号直接压入

stack.append(token)

elif token == ')':

# 右括号一直弹出，直到遇到左括号，相当于堆栈中的堆栈

top\_token = stack.pop()

while top\_token != '(':

output.append(top\_token)

top\_token = stack.pop()

else:

raise ValueError(f"Invalid token {token}")

while stack:

if stack[-1] == '(':

raise ValueError("Mismatched parentheses in expression")

output.append(stack.pop())

# 返回的结果是转换出的逆波兰表达式

return ' '.join(output)

# 功能：计算逆波兰表达式的值

def evaluate\_postfix(postfix\_expression):

stack = []

tokens = postfix\_expression.split()

for token in tokens:

if is\_number(token):

stack.append(float(token))

elif token in ['+', '-', '\*', '/', '%']:

if len(stack) < 2:

raise ValueError("Invalid postfix expression")

b = stack.pop()

a = stack.pop()

if token == '+':

result = a + b

elif token == '-':

result = a - b

elif token == '\*':

result = a \* b

elif token == '/':

if b == 0:

raise ValueError("Division by zero")

result = a / b

elif token == '%':

# 向下取整

result = a // b

stack.append(result)

else:

raise ValueError(f"Invalid token {token} in postfix expression")

if len(stack) != 1:

raise ValueError("Invalid postfix expression")

# 堆栈中最后一个值就是结果

return stack[0]

# 加载框架的类，父类：CTkFrame

class loadframe(CTkFrame):

# 功能：框架初始化；形参：选择的框架名(frame\_name)，框架内容(frame)

def \_\_init\_\_(self,master:CTkFrame,frame\_name,frame: dict):

super().\_\_init\_\_(master = master, corner\_radius = 0, fg\_color="transparent")

self.rowconfigure((0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10), weight=1)

self.columnconfigure(0, weight=1)

self.columnconfigure(1, weight=4)

self.frame\_name = frame\_name

self.parameter = frame['parameter']

self.expression = frame['expression']

self.content = frame['template\_str']

# 存储所有参数值的字典，包括需要输入的值和表达式计算出的值

self.parameter\_value = {}

self.entry = {}

# 遍历标签文本列表，创建标签和输入框

for i, text in enumerate(self.parameter):

# 创建标签

label = CTkLabel(self, text=f"{text} ", anchor="e", width=100)

# 将标签放置到网格中

label.grid(row=i, column=0)

# 创建输入框

self.entry[text] = CTkEntry(self, width=160)

# 将输入框放置到网格中

self.entry[text].grid(row=i, column=1)

# 功能：输出文件，作为按钮的命令

def output\_file(self):

for item in self.parameter:

self.parameter\_value[item] = float(self.entry[item].get())

for key in self.expression:

# 含变量中缀表达式 > 只含数字的中缀表达式 > 逆波兰表达式 > 计算结果

self.parameter\_value[key] = round(evaluate\_postfix(infix\_to\_postfix(replace\_variables(self.expression[key],self.parameter\_value))), 4)

for key in self.parameter\_value:

self.parameter\_value[key] = float(self.parameter\_value[key])

if isinstance(self.parameter\_value[key], float) and self.parameter\_value[key].is\_integer():

self.parameter\_value[key] = int(self.parameter\_value[key])

self.content\_processed = self.content.format(\*\*self.parameter\_value)

try:

with open(f"./output/{self.frame\_name}\_{time.strftime('%Y-%m-%d-%H-%M-%S')}.il", mode="w", encoding="utf-8") as fp:

fp.write(self.content\_processed)

# 写入成功后，提示成功

messagebox.showinfo(title="提示弹窗", message="文件生成成功")

# 如果写入失败，则弹窗提示异常信息

except Exception as exp:

messagebox.showerror(title="提示弹窗", message=f"保存文件异常: {exp}")

return

class App(CTk):

# 功能：GUI的初始化；形参：所有模版的内容字典

def \_\_init\_\_(self, filed: dict):

super().\_\_init\_\_()

self.\_calc\_window\_pos() # 生成窗口

self.title("互联寄生测试结构模型自动生成")

self.rowconfigure(0, weight=1)

self.template\_dict = filed

# 框架字典：键为框架名，值为创建的框架

self.frame\_dict = {}

self.frame\_left = CTkFrame(self, fg\_color="transparent")

self.frame\_left.grid(row=0, column=0, sticky="nsew", padx=50)

self.frame\_left.rowconfigure(0, weight=1)

self.frame\_left.rowconfigure(1, weight=1)

self.frame\_left.rowconfigure(2, weight=1)

self.frame\_left.rowconfigure(3, weight=1)

self.frame\_left.columnconfigure(0, weight=1)

self.option\_menu = CTkOptionMenu(self.frame\_left, width=150, height=30, values=list(self.template\_dict.keys()),command=self.frame\_select)

self.option\_menu.grid(row=0, column=0, pady=70)

btn\_font\_obj = CTkFont(size=20, weight="bold", family="微软雅黑")

self.btn\_gen = CTkButton(self.frame\_left, text="确认", font=btn\_font\_obj)

self.btn\_gen.grid(row=1, column=0, pady=70)

# 各个模型GUI的部分定义

self.frame\_right = CTkFrame(self, fg\_color="transparent")

self.frame\_right.grid(row=0, column=1, sticky="nsew")

self.frame\_right.rowconfigure(0, weight=1)

self.frame\_right.columnconfigure(0, weight=1)

for key in self.template\_dict:

self.frame\_dict[key] = loadframe(self.frame\_right,key,self.template\_dict[key])

print(type(self.frame\_dict[key]))

self.frame\_select()

self.init\_dir()

# 功能：检查是否创建output文件夹，如果没有则创建一个output文件夹

def init\_dir(self):

output\_dir\_path = os.path.join(os.getcwd(), "output")

if not os.path.exists(output\_dir\_path):

try:

os.mkdir(output\_dir\_path)

except Exception as exp:

messagebox.showerror(title="", message=f"创建文件夹失败: {exp}")

return

# 功能：设置窗体大小和位置

def \_calc\_window\_pos(self):

root\_width = 600

root\_height = 600

screen\_width = self.winfo\_screenwidth()

screen\_height = self.winfo\_screenheight()

root\_pos\_x = int((screen\_width - root\_width) / 2)

root\_pos\_y = int((screen\_height - root\_height) / 2)

self.geometry(f'{root\_width}x{root\_height}+{root\_pos\_x}+{root\_pos\_y}')

self.resizable(False, False)

# 功能：改变GUI框架

def frame\_change(self,frame\_name: str):

# 切换下拉选项，同时切换下拉选项对应的输入参数

frame\_conf= dict(row = 0, column = 0, sticky = "nsew")

for key in self.frame\_dict:

self.frame\_dict[key].grid\_forget()

self.frame\_dict[frame\_name].grid(\*\*frame\_conf)

self.btn\_gen.configure(command = self.frame\_dict[frame\_name].output\_file)

# 功能：选择GUI框架

def frame\_select(self,\*args):

self.frame\_change(self.option\_menu.get())

# 程序入口

if \_\_name\_\_ == '\_\_main\_\_':

filed = load\_file()

app = App(filed)

app.mainloop()

致谢

在这个春暖花开、生机勃勃的季节，我的大学生活即将画上圆满的句号，四年的本科学习经历，也将成为我人生中宝贵的财富。在此，我要向所有给予过我帮助和支持的人表示感谢。

首先，我要衷心感谢我的导师孙亚宾教授。从论文选题到最终定稿，您都给予了我悉心的指导和帮助。在您严谨的学术态度、深厚的专业知识以及无微不至的关怀下，我才能不断解决程序设计中出现的问题和不足，逐步推进项目进度。

其次，我要感谢我的父母，是你们的支持和爱给了我不断学习、不断进步的动力，你们是我坚强的后盾，是我学生时代最好的人生和思维的启蒙老师，教会了我为什么去学习和工作、该怎样学习和工作。

此外，我还要感谢我的老师和同学们。微电子科学与工程专业的老师们认真授课，对学生悉心指导，同学们相互陪伴，彼此交流问题、共同进步，也正是这样和谐的学习环境为我以后的学习和工作打下了坚实的基础。

最后，我要感谢母校为我提供的良好学术环境和丰富的学习资源。在本科四年期间，我不仅学到了专业知识，还锻炼了自己的能力，提高了学科素养，获得了独立思考、独立处理问题的能力。

在未来的工作和学习中，我将继续努力，为实现自己的梦想而奋斗。有了本科四年宝贵的经历，我将有足够的底气去地迎接未来的挑战，书写更加精彩的人生篇章。再次感谢所有给予过我帮助的人！