一.绪论

1.1课题背景

在我们设计数字集成电路时，有一个非常重要的步骤，逻辑函数化简。逻辑函数化简在本文中特指将逻辑函数化简为最简与或式的过程。将逻辑函数化简为最简与或式，可以减少逻辑门的数量，降低电路复杂度，提高电路的可靠性。如果不化简逻辑函数，直接设计电路，一般会增加使用逻辑门的数量，从而使电路复杂，成本增加。

常用的逻辑函数化简方法有公式法，Q-M化简法，卡诺图法。当使用公式法时，我们需要人为选择合适的公式进行化简，容易出错，化简结果也不统一，不适合计算机使用。

Q-M化简法是由Quine和Mccluskey提出的，在部分参考文献中，也被称为系统列表化简法。Q-M化简法和卡诺图化简法的化简思想基本一致，都是通过逐步合并二进制相邻的项来实现逻辑函数的化简。卡诺图化简法使用图形化的化简方式，通过卷起相邻的(0)项非常直观的完成化简。而Q-M化简法使用函数F的质蕴涵项集来求得最简表达式，更适合计算机编程实现。

1.2国内外相关研究现状及发展趋势

李昂等[1]使用了树这种数据结构来实现Q-M化简法。Schwender[2]选择了用矩阵这种相对简单的数据结构，运用的矩阵代数方法有助于降低Q-M化简法的代码量，更容易使用Matlab实现。朱幼莲[3]采用十进制数来表示多维体，这借鉴了神经网络中的增大批处理大小的思想。这种方法的贡献在于提出了可以从复合多维体中选择一对十进制数来处理最小项，并给出了对应的相邻判别规则和求解最小覆盖的方法。

1.3论文工作安排

本文的第二章介绍了QM化简法的基本理论，并举例描述。在本文的2.1节，我们将介绍QM化简法的基本思路。在本文的第2.2节，我们将介绍QM化简法涉及的相关知识点。在本文的第2.3节，我们将举例说明QM化简法是如何运用在逻辑函数的化简的。

本文的第三章描述了如何使用C语言程序实现QM化简法。描述了函数3个源文件和2个头文件的基本功能，以及具体函数的功能和输入输出数据的方式。同时介绍了本文使用的以链表为基础的数据结构，以及链表的基本单元结构体的构成。

本文的第四章，给出了具有唯一化简的逻辑函数，具有多种化简的逻辑函数，无关项不参与化简的逻辑函数，无关项参与化简的逻辑函数这四组测试数据，并均得到了正确的结果。这证明设计的C语言程序可以实现逻辑函数化简的功能。

本文的第五章，总结了使用C语言实现逻辑函数化简的相关原理，并对程序的优化提出了自己的思考。

二．相关知识理论

2.1课题设计的主体思路

系统化简法(Q-M化简法)化简输入的逻辑函数的过程可以分为以下四步。

步骤一，以最小项的形式输入逻辑函数，并二进制化。

步骤二，找出并保存逻辑函数中的全部质蕴含项，以判断质蕴含项是否必要。

步骤三，舍弃非必要的质蕴含项，保留必要的质蕴含项。

步骤四，可以覆盖输入逻辑函数的最少的变量。

2.2课题设计涉及的知识点

蕴涵项(Implicant)：在函数的“与-或”表达式中,每个“与”项被称为该函数的蕴涵项。在函数卡诺图中，任何一个1方格所对应的最小项或者卡诺圈中的2m个1方格所对应的“与”项都是函数的蕴涵项。

本原蕴含项：以最小项形式列出开状态集合和无关项集合中所有元素，元素用二进制表示。第一列中，根据二进制数含1的个数分组。第一组只能和第二组比较，第二组只能和第三组比较，依此类推。如果比较的元素只有一位不同，那么他们所代表的最小项在n维布尔空间是相邻的，并由这两个元素得到蕴含项，将其放入下一列，如果一个元素对新的蕴含项没有贡献，则其为本原蕴含项，并将其保存。对新的列，重复进行上述操作，直到找到所有本原蕴含项为止。

质蕴涵项(Prime Implicant)：质蕴涵项，也称质项，是指那些不能被其他蕴含项包含的蕴涵项。质蕴含项只代表覆盖了最大的蕴含项，但是不同的质蕴涵项可以覆盖部分重叠的蕴含项。故最小覆盖不一定需要全部的质蕴涵项。在函数卡诺图中，按照最小项合并规律，如果某个卡诺圈不可能被其他更大的卡诺圈包含，那么，该卡诺圈所对应的“与”项为质蕴涵项。

必要质蕴涵项：必要质蕴涵项，也称必要质项，是一种特殊的质蕴涵项。由于其包含了不被其他质蕴涵项包含的最小项，所以在生成最小覆盖时必须包含，所以称为必要质蕴涵项。

最小覆盖：如果某一个最小项只被某个本原蕴含项蕴含一次，则此本原蕴含项为本质本源蕴含项，最终必定要输出。然后寻找能够实现对其余最小项覆盖的本源蕴含项。然后输出。

2.3举例说明

例2.1:测试用例为逻辑函数

第一步，我们需要将给出的最小项用二进制表示，以做出真值表。在我们的程序中使用Reslove\_LNode函数将输入的最小项转为二进制表示，对于最小项x的第i位，按如下公式进行计算。

式(2-1)

其中，1<<i代表将1向左移动i位，&表示按位与。

通过以下方法，得到例1的逻辑函数的真值表如表2-1所示。

|  |  |  |
| --- | --- | --- |
| 项号 | A B C D | F |
| 0 | 0 0 0 0 | 1 |
| 2 | 0 0 1 0 | 1 |
| 4 | 0 1 0 0 | 1 |
| 5 | 0 1 0 1 | 1 |
| 6 | 0 1 1 0 | 1 |
| 7 | 0 1 1 1 | 1 |
| 9 | 1 0 0 1 | 1 |
| 15 | 1 1 1 1 | 1 |

下面我们以最小项的二进制表示中含有1的个数为标准，将最小项进行分组。然后各组按含有1的个数升序排列。通过这种分组方式方式，我们可以更直观，更迅速的找到待合并的最小项。因为待合并的最小项具有以下2个性质。第一，对于两个可以合并的相邻最小项，他们的二进制表示只相差1位。第二，由于已按含有1的个数分组，可合并的最小项只能位于相邻两组中。

对于已被合并的最小项，我们用“o”表示。被合并的最小项相同部分保持不变，合并的1位用“-”表示，并将合并的最小项填写至下一列。同时，在合并最小项时，我们也需要将组号一并合并

对于无法再进行合并的最小项，我们使用“P”表示。并不再参与后续的合并流程。

对新合成的最小项进行下一轮合并，直到新合成的最小项不再存在相邻的情况时，合并流程结束。

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 组号 |  | ABCD | 合并 |  | ABCD | 合并 |  | ABCD | 合并 |
| 0 | 0 | 0000 | o | 0 2 | 00-0 | o | 0 2 4 6 | 0--0 |  |
| 1 | 2 | 0010 | o | 2 6 | 0-10 | o | 4 5 6 7 | 01-- |  |
| 4 | 0100 | o | 4 5 | 010- | o |  |  |  |
| 2 | 5 | 0101 | o | 5 7 | 01-0 | o |  |  |  |
| 6 | 0110 | o | 6 7 | 011- | o |  |  |  |
| 9 | 1001 | o | 9 | 1001 |  |  |  |  |
| 3 | 7 | 0111 | o | 7 15 | -111 |  |  |  |  |
| 4 | 15 | 1111 | o | 7 15 | -111 | o |  |  |  |

如表2-2所示，我们一共得到了4个质蕴含项，，由于上述四个蕴含项不能相互覆盖，故4个质蕴含项均为必要质蕴含项。

最后，我们需要找到最小覆盖。最小覆盖的定义如下，最小覆盖需要包含全部的最小项的同时，使用最少的质蕴含项。由质蕴含项的定义，为了覆盖全部最小项，至少需要选择全部的必要质蕴含项。本例如表2-2所示，选取质蕴含项即可满足上述要求。

综上所述，该逻辑函数通过Q-M化简法，得到结果式2-2。

(式2-2)

三、实验过程

3.1实验环境及程序描述

QM算法演示程序使用C语言描述，标准为C99，使用集成开发环境CLion提供的开发工具CMake构建。开发电脑为MacBook Air(M1,2020)，操作系统为macOS Monterey 12.3.1。同时，已测试过在Windows 1064位系统的兼容性。

本程序共包含2个头文件，3个源文件。头文件为Input.h和List\_Operation.h。源文件为Input.c List\_Operation.c 和 main.c。

Input.c/Input.h部分，主要用于处理输入数据，包含平方计算函数int\_power，由输入最小项判断变量个数的函数Num\_Variate，将输入最小项的二进制值转换为链表储存的函数Reslove\_LNode/Reslove\_Mini，及接受用户输入的函数Receive。

List\_Operation.c/List\_Operation.h部分，主要用于处理相关链表，实现Q-M化简法。creatnode\_LNode函数和creatnode\_Minimum函数描述了创建基本蕴含项和最小质蕴含项链表的方法，与之配套的还有释放相关链表节点的Delete\_LNode和Delete\_Minimum方法。我们使用\_Printf\_Data和\_Printf\_Data\_Mini函数进行输出调试，Compare\_Func,Compare\_record,LabeL\_Min\_Essent以及其他本部分未提及的函数，为处理Q-M化简法的核心函数及核心函数配套函数，我们将会在下一节进行详细的描述。

将项目拆分成多个源文件，有助于降低整个程序的耦合性，在对程序进行修改时，加快重编译的速度，同时提高程序的可读性及可维护性。引用上述头文件时，我们使用条件编译避免头文件被重复定义，见例3.1。

例3.1：使用条件编译避免头文件Input.h重定义

#ifndef SIMPLEPLAYGROUND\_INPUT\_H

#define SIMPLEPLAYGROUND\_INPUT\_H

#endif

// 项目名称为SIMPLEPLAYGROUND

当我们需要使用自定义的头文件时，需要使用引号方式引用，使编译器优先在本目录里面搜索，使用库函数提供的头文件时，需要使用尖括号进行应用，使其优先在编译器目录进行搜索，见例3.2。

例3.2：使用不同方式引用头文件

#include <stdio.h>

#include "Input.h"

页-1 下面，我们提供本程序的流程图见图3-1，并对关键模块进行进一步的解释。

3.1 数据输入

对于本程序，用户需要输入全部最小项和无关项，以及最小项和无关项的数量。

数据输入由主函数打印用户交互界面并调用Receive(LNode \*\*head\_LNode,Minimum \*\*head\_Mini,int Num\_min,int Num\_Irelev)函数实现。输入参数head\_LNode和head\_Mini是指向逻辑变量和最小覆盖链表头结点的二重指针。

对于一个逻辑变量链表的结点LNode，我们使用一个结构体进行描述。

|  |  |  |
| --- | --- | --- |
| 类型 | 名称 | 描述 |
| Int | data | 最小项表达式值 |
| \*Char | Bits | 最小项表达式二进制值 |
| Char | Group\_Sequ | 包含的最小项个数 |
| Char | Mark | 重要性 |
| \*Lnode | next | 链表的下一个结点 |

对于一个最小项链表的结点MInimum，我们也使用了一个结构体进行描述。

|  |  |  |
| --- | --- | --- |
| 类型 | 名称 | 描述 |
| Int | data | 最小项表达式值 |
| \*Char | Bits | 最小项表达式二进制值 |
| Char | Num\_Label | 组号 |
| \*MInimum | next | 链表的下一个结点 |

传统方法是使用数组储存上述两组表格，这样的缺点是需要预分配更多的存储空间。由于Q-M化简法在生成质蕴涵项时，可以释放掉已被合并的部分，所以使用链表这一数据结构更符合Q-M化简法的工作流程。

传统的将十进制数转为m进制数的方法是除m取余数，在将所得结果倒序排列。由于逻辑函数变量是可以互换且不影响最终表示的，所以储存在结构体里的最小项表达式二进制值并没有倒序排列，而且直接储存，以优化时间复杂度。

3.2寻找质蕴含项

先找到组号为i的第一个数据，然后寻找组号为i+1的第一个数据，比较后寻找组号为i+2的第二个数据，直到比较完毕。然后循环寻找组号为i+1的第一个数据，然后寻找组号为i+2的第一个数据，比较后寻找组号为i+1的第二个数据，直到比较完毕。依此递推。比较完毕。

相关函数void Compare\_Func(LNode \*be\_compare,LNode \*Columns\_LNode)。输入函数为两个链表的节点，通过改变节点标志位返回结果。从链表第一个节点开始，遇到组号为0的节点从头开始查找是否有节点组号为1，若有判断是否为相邻项，标记两者为非本源项，并将不同的位置标记为 NONE。判断是否相邻的方法是利用二进制数组，如果只有一位不同，置位NONE，复制到一个新的节点。

3.2寻找最小质蕴含项

为每一位变量设置优先级数，确保例如A的优先级>A’BCD…所有优先级之和，A’的优先级>BCD…所有优先级之和，然后输出优先级最大的本质本源项。设为第i位变量的优先级，为第i位反变量优先级，其数学解释为

（式3-1）

该功能由主函数循环调用Is\_Label\_1 (LNode \*Ultimate\_List,Minimum \*head\_node\_Minimum,LNode \*head\_ESSENT\_PRIME)实现。我们首先重新标记新的本质本源项和最小项。接着，判断包含最多最小项的本质本源项，如果两者包含的一样多，取第一个，复制到最终链表里。然后，删除此本质本源项所包含的其他最小项,删除最小项时头结点和尾节点均要删除。最后，用循环次数测试头结点，尾节点用下一指针为空判断，以及将此本质本愿项Group\_Sequ置位0，循环结束的条件是最小项列表为空。

3.3寻找最小覆盖

此功能由函数Least\_Cover(LNode \*Ultimate\_List,Minimum \*head\_node\_Minimum,LNode \*head\_ESSENT\_PRIME)实现。过程是使去掉本质本源项的剩余本源项对剩余的最小项进行标记，然后寻找包含最小项最多的本源项，输出并删除其所包含的最小项。如果包含的最小项一样多则取第一个本源项输出。直到最小项链表为空。

3.6输出结果

输出结果的函数由Print\_Variate(LNode \*Ultimate\_List)实现，并完成对输出顺序的控制。对每一个变量位创建一个优先级表，优先级大的输出。

|  |  |
| --- | --- |
| 变量位 | 优先级 |
| None | 0 |
| 第一位反变量 | 1 |
| 第一位正变量 | 2 |

之后每一位反变量，优先级为前面所有变量位正变量优先级之和+1，之后每一位正变量优先级这一位反变量优先级+1。

输出结果时，我们使用代替，表示对逻辑变量A取反。

由于上述优先级规则，输出时，每个小项按第一个逻辑变量的降序排列。

四、实验结果分析

我们一共设计了四组逻辑函数对我们设计的程序进行测试，分别是具有唯一化简的逻辑函数，具有多种化简的逻辑函数，无关项不参与化简的逻辑函数，无关项参与化简的逻辑函数，以下为我们使用的测试样例。

例4.1 使用测试程序化简逻辑函数

该逻辑函数只有唯一的最简与-或式，为式4-1，程序输出的最简与-或式是正确的。

(式4-1)

文本

描述已自动生成

例4.2 使用测试程序化简逻辑函数

该逻辑函数有多组最简与-或式，为式4-2和式4-3，程序输出的最简与-或式为式4-2是正确的。

(式4-2)

(式4-3)

文本

描述已自动生成

例4.3 使用测试程序化简逻辑函数&

无关项不参与化简，程序输出的最简与-或式为式4-4是正确的。

(式4-4)

文本

描述已自动生成

例4.4 使用测试程序化简逻辑函数&

无关项不参与化简，程序输出的最简与-或式为式4-5是正确的。

(式4-5)

文本

描述已自动生成

五、总结与展望

综上所述，相较于以总结规律为主的代数化简法和以作图辅助合并的卡诺图法，使用系统列表法在化简逻辑函数过程中可以运用明确的规则，故可以运用升维思想增大批处理，改进数据结构，引入更科学的判别规则等方法进行优化。与前两种方法相比是一种适合计算机使用的逻辑函数化简法。

程序使用链表这一种数据结构，使用C语言实现了Q-M化简法。对于逻辑变量和最小项，使用结构体定义，增加了可读性。同时由于引入多组源文件与头文件，降低了程序的耦合性，提高了程序的可读性和可维护性。实验证明，设计的程序可正确化简含有无关项的逻辑函数。本程序的主要贡献在于，充分利用C语言特性，使用更优的数据结构和程序组织方式，实现了Q-M化简法，降低了空间复杂度。

根据徐俊平等提出的改进Q-M逻辑函数化简方法，本文在算法的时间复杂度上仍有一定的优化空间使用启发式的方法，引入最小项关联图，可以更进一步的降低时间复杂度。