Advances in Applied Mathematics 应用数学进展, **2021, 10(10), 3495-3504**
Published Online October 2021 in Hans. http://www.hanspub.org/journal/aam
https://doi.org/10.12677/aam.2021.1010369

Hans汉斯

Gröbner基法验证乘法器的设计与实现

吕妍颖，张小盈，江建国*

辽宁师范大学数学学院，辽宁 大连

收稿日期：2021年9月22日；录用日期：2021年10月15日；发布日期：2021年10月26日

摘 要

当今形式化验证工具在大规模集成电路的设计过程中起着非常重要的作用。最有效的方法是以Gröbner基方法为基本原理，将乘法器电路建模为一组伪布尔多项式，通过Gröbner基来既约由多项式表示的字级规范。本文将基于Gröbner基方法，使用C++语言重新实现验证工具，将整个代码分成多个模块的形式，并利用容器类对变量进行分类存储。实验结果表明：C++语言设计的验证工具不仅可以实现成功，而且也为之后的研究提供了有利的验证工具。

关键词

Gröbner基，C++，形式化验证

Design and Implementation of Gröbner Basis Verification Multiplier

Yanying Lv, Xiaoying Zhang, Jianguo Jiang*

School of Mathematics, Liaoning Normal University, Dalian Liaoning

Received: Sep. 22nd, 2021; accepted: Oct. 15th, 2021; published: Oct. 26th, 2021

Abstract

Today's formal verification tools play a very important role in the design of large-scale integrated circuits. The most effective method is based on the Gröbner basis method. The multiplier circuit is modeled as a set of pseudo-Boolean polynomials, and the word-level specification represented by the polynomial is reduced through the Gröbner basis. This article will re-implement the verification tool based on the Gröbner base method and use the C++ language, divide the entire code into multiple modules, and use the container class to classify and store the variables. The experimental

*通讯作者。

文章引用：吕妍颖，张小盈，江建国．Gröbner 基法验证乘法器的设计与实现[J]．应用数学进展，2021，10(10): 3495-3504. DOI: 10.12677/aam.2021.1010369

results show that the verification tool designed by C++ language can not only achieve success, but also provide a favorable verification tool for subsequent research.

Keywords

Gröbner Base, C++, Formal Verification

Copyright © 2021 by author(s) and Hans Publishers Inc.
This work is licensed under the Creative Commons Attribution International License (CC BY 4.0).
http://creativecommons.org/licenses/by/4.0/

Open Access

1. 引言

验证数字电路的正确性是超大规模集成电路(VLSI)设计领域中的一个重要问题[1]。随着工艺技术的进步，人们已经可以制造出相当复杂的电路，但这种复杂性也带来越来越多的错误。为了保证错误被及时的发现，需要对电路的正确性进行验证。但由于芯片越来越复杂，设计周期越来越短，产品竞争越来越激烈等诸多因素，导致电路验证成为 VLSI 设计领域内的一个瓶颈。传统的电路验证方法有动态测试法[2]，它创建测试平台，在设计的可执行软件模型上运行这些测试。但测试平台的开发是一项漫长的过程，不仅容易出错，而且需要高昂的成本。

而当今最流行的电路验证方法是形式化验证，它利用数学方法来解决电路设计的问题。当它第一次被引入时，很少有人相信它是可行的。但现在形式化验证已在众多重大项目的各个领域中使用，不仅提高了生产力和识别棘手的极端情况错误，而且缩短了上市时间，并取得了宝贵的成果。

数字电路执行逻辑运算，这使其成为计算机和数字系统中的重要组成部分。如果电路执行算术运算，则称为算术电路。算术电路尤其是乘法器电路的验证在设计中是至关重要的一环，但即使到今天，也没有一个完美的解决方案。例如二叉判定图会出现指数级膨胀[3]。布尔可满足性方法和可满足性模理论求解方法都无法验证更大规模的乘法器电路[4]。使用定理证明器需要大量的手工工作和专业知识[5]。

近几年来，基于 Gröbner 基理论的形式化方法受到了学术界的广泛关注与认同，并在验证整数乘法器方面已显示出非常好的结果[6] [7]。Ritirc 等人提出了逐列验证方法[8]，该方法将问题分为几个较小的、易于计算的子问题，不需要既约整个字级规范，以此来提高验证速度。Ritirc 等人并在此基础上提出了加法器重写优化[9]，通过提取乘法器中的全加器和半加器，既约了 Gröbner 基，进一步加快了验证速度。

虽然已经研发出多种乘法器电路的验证工具，但都处于实验阶段，没有进入工业界。因为现在的验证工具仅仅热衷于寻找验证方法，对于验证工具的代码实现草草了之。在实现代码中，他们将上千行代码写在一个文件中，没有考虑过代码的架构，也不利于后期的开发和代码的调试。

因此，本文利用 C++语言[10]重新实现乘法器的验证工具，相比于一个 C 语言编写的程序，该程序被拆分为多个模块，不仅有利于代码的调试与运行，而且为之后的优化提供了方便。其中添加了顺序容器对其进行优化，并且实现成功。

2. 验证原理

2.1. Gröbner 基理论

本文主要给出 Gröbner 基理论的定义，其它相关定义、定理及证明可以参考[11] [12]。

设 k 是数域，$k[X]$ 或 $k[x_1,\cdots,x_n]$ 是一个多项式环。单项式常见项序有字典序、分次字典序和分次逆字典序等。单项式序可以定义出任意多项式的首项，并保证多项式除法得到的余式唯一。

定义 2.1 设 I 是 $k[x_1,\cdots,x_n]$ 的非零理想，给定一个字典序。若存在 I 的有限子集 $G=\{g_1,\cdots,g_m\}$ 满足

$$\langle LT(g_1),\cdots,LT(g_m)\rangle=\langle LT(I)\rangle$$

则 G 是 I 的一个 Gröbner 基。

理想 I 的 Gröbner 基一般不是唯一的，它与项序的选择密切相关。

2.2. 代数模型

乘法器电路具有 $2n$ 个布尔输入 $a_0,\cdots,a_{n-1},b_0,\cdots,b_{n-1}$ 和 $2n$ 个布尔输出 $s_0,\cdots,s_{2n-1}$，并且每个内部门(门输出)由一个门变量 $g_0,\cdots,g_{k-1}$ 表示。该电路由许多逻辑门表示，其中某些门的输出可能是其它门的输入，但不允许在电路中循环[13]。我们假设

$$X=a_0,\cdots,a_{n-1},b_0,\cdots,b_{n-1},g_0,\cdots,g_{k-1},s_0,\cdots,s_{2n-1}$$

将门输出 u 与门输入 v,w 用以下多项式表示：

$$\begin{aligned} u=\neg v &\quad\Leftrightarrow\quad -u+1-v=0\\ u=v\wedge w &\quad\Leftrightarrow\quad -u+vw=0\\ u=v\vee w &\quad\Leftrightarrow\quad -u+v+w-vw=0\\ u=v\oplus w &\quad\Leftrightarrow\quad -u+v+w-2vw=0 \end{aligned} \tag{2.1}$$

电路中的所有变量均为布尔变量，为确保仅找到多项式的布尔解，我们为每个变量 u 添加关系式 $u(u-1)=0$，并称这种关系式为域多项式。

定义 2.2 设 C 是一个电路，$G\subseteq Q[X]$ 为包含电路 C 中每一个门对应的多项式(见(2.1))和输入域多项式 $a_i(a_i-1)$ 以及 $b_i(b_i-1)$ 构成的多项式集合，其中 $0\le i<n$。环 $Q[X]$ 中 G 生成的理想记为 $J(C)$。

定理 2.1 设 C 是一个电路，G 是定义 2.2 中的多项式集合，给定一个字典序 $>_{lex}$，那么 G 为 $J(C)$ 的一个 Gröbner 基。

定义 2.3 设 C 是一个电路，如果将每个 $(a_0,\cdots,a_{n-1},b_0,\cdots,b_{n-1})\in\{0,1\}^{2n}$ 和 $g_0,\cdots,g_{k-1},s_0,\cdots,s_{2n-1}$ 代入到多项式 p 中结果为零，则多项式 $p\in Q[X]$ 称为电路 C 的规范多项式(PCC)。

以下推论表明 $J(C)$ 包含电路 C 中变量之间的所有关系。

推论 2.1 任一多项式 $p\in[x_1,\cdots,x_n]$，p 是 PPC 当且仅当 $p\in J(C)$。

一个给定的字级规范适用于 C，当且仅当它是理想 $J(C)$ 中的元素。因此，我们可以使用多元除法来确定理想成员。因此，验证乘法器的正确性可以归为理想成员问题。

定义 2.4 若

$$\sum_{i=0}^{2n-1}2^i s_i-\left(\sum_{i=0}^{n-1}2^i a_i\right)\left(\sum_{i=0}^{n-1}2^i b_i\right)$$

是一个 PPC，则电路 C 被称为乘法器。

2.3. Gröbner 基法

当 G 是理想 $I=\langle G\rangle=\langle g_1,\cdots,g_m\rangle\subseteq Q[X]$ 的一个 Gröbner 基，多项式 $q\in Q[X]$，则 $q\in I$ 当且仅当 q 除以 G 的余式 r 为 0。结合多元多项式除法，Gröbner 基可以用来判定任意多项式是否属于该理想。

在 Gröbner 基方法中，SP 是根据乘法器的输入和输出确定的规范多项式。例如，2 位乘法器的规范多项式为 $SP = 8S[3]+4S[2]+2S[1]+S[0]-(2A[1]+A[0])(2B[1]+B[0])$，其中 $8S[3]+4S[2]+2S[1]+S[0]$ 表示 2 位乘法器的输出，而 $(2A[1]+A[0])(2B[1]+B[0])$ 表示 2 位乘法器的输入。

Gröbner 基方法的一般思想：

1) 将乘法器中变量之间的关系生成一个 Gröbner 基 $G=\{g_1,\cdots,g_m\}$，并将乘法器的输入和输出表示为规范多项式 SP。

2) 将 G 中的多项式代入 SP 中或 G 对 SP 进行逐步除法(称为向后重写)，余式为 r。

3) 如果 SP 为零或 $r=0$，则乘法器是正确的；否则，乘法器是错误的。

3. 架构设计

3.1. 需求分析

验证电路的正确性在设计领域是非常重要的，正确的电路设计可避免因重新设计浪费开发成本和设计周期。对于任意给定的乘法器，验证工具都应读取输入文件掌握乘法器的结构，并通过遍历乘法器的节点变量给出需要的信息，最后得出输出文件。但由于大乘法器的结构复杂，在验证乘法器的过程中，可能导致验证失败，因此，验证工具不仅需要正确的验证乘法器，而且还应该具有优化等功能，进一步提高验证速度以及验证更大的乘法器。

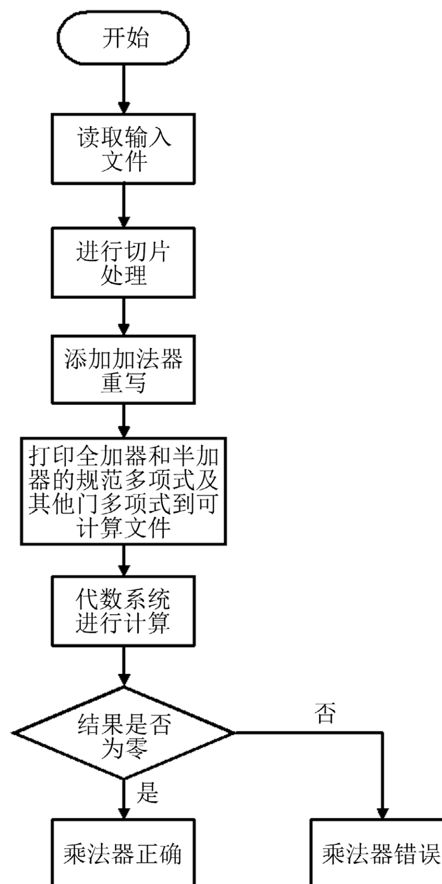

Figure 1. Flow chart of the verification tool
图 1. 验证工具的流程图

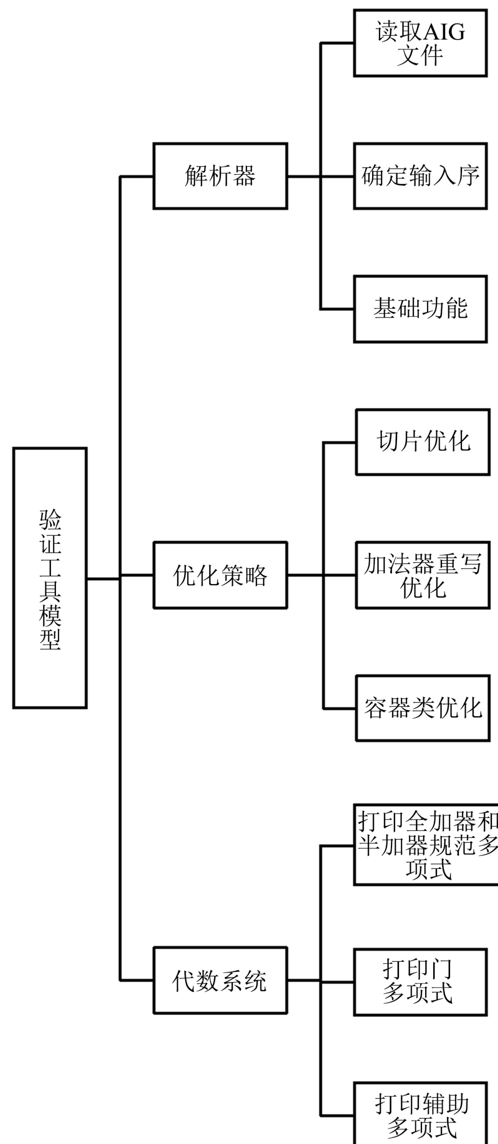

Figure 2. Overall architecture of the verification tool
图 2. 验证工具的总体架构

3.2. 总体架构

验证工具是为了满足乘法器验证的需求而设计的，该验证工具的目标是将 Gröbner 基方法与形式化验证相结合，最终完成一个独特的、能够解决乘法器正确性问题和满足实际需求的验证工具。

根据需求分析，验证工具应具备以下功能：

1) 可以读取输入文件；

2) 可选择进行的操作；

3) 能直观的给出节点的相关信息；

4) 具有优化功能，可以提高验证速度；

5) 可以验证乘法器的正确性。

形式化验证主要是将乘法器建模为多项式，利用 Gröbner 基方法既约规范多项式，当既约结果为零

时，则乘法器是正确的，否则是错误的。根据验证工具的需求分析，给出验证工具的一般工作流程：首先选择代数系统，随后读取 AIG 文件[14]。针对 AIG 文件中的变量文字，确定输入、输出变量及与门节点信息。根据节点排列位置，对乘法器进行切片优化，并将切片中对应的门多项式打印到可计算文件中。如果选择了加法器重写优化，则需要查找乘法器中所包含的加法器和半加器，将全加器和半加器的规范多项式及其他门多项式打印到可计算文件中。最后将可计算文件传递给代数系统 Mathematica [15]和 Singular [16]进行计算，如果计算结果为零，则乘法器是正确的，否则是错误的，进而验证乘法器的正确性。验证工具的工作流程如图 1 所示。

结合验证工具的工作流程，设计了验证工具的总体架构。如图 2 所示，包括解析器、优化策略和代数系统。解析器实现读取 AIG 文件并确定输入序；将输入、输出和中间变量进行标记，并为其命名；通过遍历节点，给出所有节点的层次关系。优化策略包括切片优化、加法器重写优化和容器类优化，是验证工具的核心，对验证工具进行优化可提高验证速度，是验证乘法器中重要的一环。代数系统将全部门多项式打印到可计算文件中，如果进行了加法器重写，将全加器和半加器的规范多项式打印到可计算文件中；随后将可计算文件传递给代数系统进行计算，进而得出计算结果。

4. 模块的设计与实现

本文的验证工具将采用 C++语言开发，因为虽然 C 语言的运行速度比较快，但当验证工具的代码达到一定长度时，很容易崩溃。而 Python 是一种脚本语言[17]，简单易学，但机器运行时间很长。Java 代码需由 Java 虚拟机读取运行，所以效率偏低[18]。而 C++语言不仅拥有计算机高效运行的实用性特征，同时还致力于提高大规模程序的编程质量，在核心的功能和需要大量代码运行的部分更倾向于选择 C++语言，并且在验证乘法器方面也更倾向于用 C++语言进行编译。下面将给出各模块的设计与实现。

4.1. 解析器

与非图(And-Inverter-Graphs)是电路的常见表示，也称为 AIG。它是由表示逻辑与的两个输入节点组成的有向无环图，边可以包含指示逻辑否定的标记，每个节点的语义蕴含了多项式关系。通过多个 AIG 描述电路形成 AIGER 格式，也就是 AIG 文件。解析器应实现 AIG 文件的读取和处理，包括 3 方面功能：读取 AIG 文件、确定输入序和给出基础信息。

AIG 文件具有 ASCII 和二进制两种版本，通常以带有“aag”标头的 ASCII 格式进行查看。在 ASCII 格式的 AIG 文件中，列出了输入、输出以及与门的文字表示，获取文字的符号位需将其无符号整数表示取模 2，符号位为 1 表示取反，符号位为 0 表示取正。因此，如果与门右侧的文字为奇数，则符号位为 1，表示取反。而读取 AIG 文件需要确定文字之间的联系，其中与门是由三个正整数组成，第一个整数是偶数，代表文字或与门左侧(LHS)，另外两个整数表示“与”门右侧(RHS)的文字。通过 AIG 文件给出的输入，确定输入序，进而判断乘法器的基准类型。本文主要验证由 boolector 基准生成的“btor”乘法器和由 AOKI 基准生成的“sp-ar-rc”乘法器，因此需要进行判断。读取 AIG 文件和确定输入序等功能通过 parser_aig()和 determine_input_order()函数实现，如表 1 所示。

通过遍历 AIG 文件中的每个节点，掌握乘法器的结构，并针对需要的功能进行设计。实现基础功能为后续的设计打下坚实的基础。遍历全部节点时，为该节点的变量赋予唯一的名字和文字，并确定变量是否是输入、输出变量。而且每个节点在乘法器中的层次是固定的，通过节点所在层次，可以确定节点所在位置，为后续尽快找到该节点提供了基础。乘法器中应当有很多的异或门，给出一个节点如何判断该节点是否是异或门是一个关键。在运行验证工具时，给出运行的时间和内存，当出现错误时，给出错误说明也是很有必要的。这些功能在此验证工具中都已经实现。

Table 1. Partial function implementation of the parser
表 1. 解析器的部分函数实现

实现函数	函数描述
parser_aig()	读取 AIG 文件，并打印第一行
determine_input_order()	确定输入序
level()	确定节点的所在层次
find_partial_products()	标记部分乘积的变量及数量

4.2. 优化策略

一个验证工具不仅要能正常运行，而且在速度上也有一定的需求。在验证大的乘法器时，常常因为乘法器过大，导致运行时间过长，进而验证失败。因此，验证工具的运行效率也是一个值得重视并为之付出努力的问题，其性能的优化也是一门复杂的学问。

首先，对乘法器进行切片优化，将一个大的乘法器问题分为更小更易于解决的子问题，逐步验证这些子问题进而验证整个乘法器，该方法也被称为逐位验证方法。如图 3 所示，是一个 3 位乘法器的结构图，按输出位开始，将每一个输出划分为一个切片。因此，在 Gröbner 基方法的基础上，该方法将 Gröbner 基 G 重写为变量较少的切片 Gröbner 基 G_i。通过 G_i 对 SP_i 进行逐步除法，将得到的余式 r_i 添加到下一个切片 Gröbner 基 G_i 中，重复此操作。如果最后得到的余式为零，则乘法器是正确的，否则是错误的。

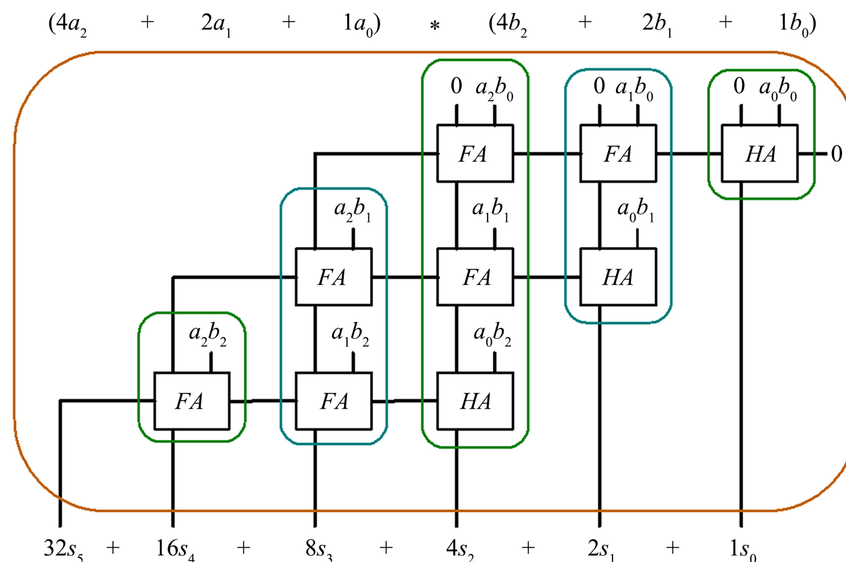

Figure 3. Schematic diagram of the bit-by-bit verification method of the 3-bit multiplier
图 3. 3 位乘法器的逐位验证方法示意图

其次，由于在遍历变量时，每次都需要从第一个变量进行遍历直到最后一个变量，其中包含许多无效变量，减慢了验证速度。因此本文利用 C++命名空间中的顺序容器类对乘法器中的变量和文字进行分类存储，以便更快的找到符合条件的变量。vector 容器相当于动态数组，它的随机访问效率高；list 容器相当于列表，它的插入和删除效率高。因此我们将 vector 和 list 结合起来，实现变量随机存储的最佳状态。利用 vector 和 list 创建一个二维数组，list 容器对象有一个 push_back()成员函数，用于在 list 容器的

最尾部中插入元素使用。通过此函数成员，在 traverse_column()函数中完成对 slices 的赋值，即返回各个切片中包含的变量。例如 std::vector<std::list<Var*>> slices。表 2 中给出了相关函数的实现。

最后，通过图 3 可以发现该乘法器是完全由半加器和全加器组成的，因此，可以对其进行加法器重写。找到乘法器电路门级表示中的全加器和半加器，并把它们视为独立的电路。利用全加器和半加器的规范表示消去内部门多项式，使乘法器的 Gröbner 基更小、更紧凑，从而加快约化过程。下面给出全加器和半加器的规范多项式以及结构图，如图 4 所示。

定义 4.1 给定输出 c,s 和输入 a,b,i。如果 $-2c-s+a+b+i$ 是一个 PCC，则电路 C 称为全加器。如果 $-2c-s+a+b$ 是一个 PCC，则电路 C 称为半加器。

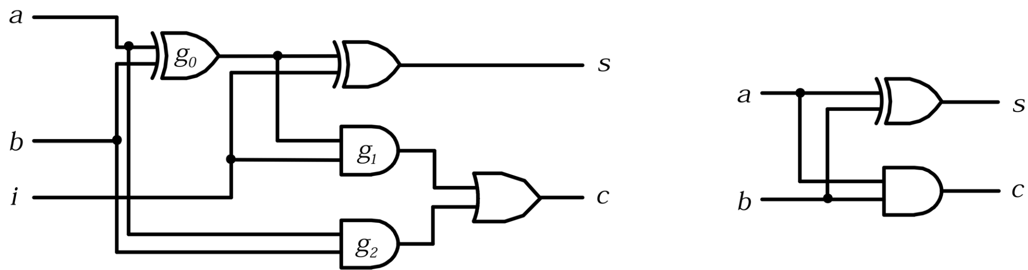

Figure 4. Structure diagram of full adder (left) and half adder (right)
图 4. 全加器(左)和半加器(右)的结构图

寻找乘法器中的全加器需要确定进位变量，表 2 给出了寻找进位变量的函数实现。在函数 collect_carry()中，将切片中的进位变量赋给数组 carry[k]，以便在寻找全加器时，直接从数组 carry[k]中的变量进行遍历。确定了进位变量之后，通过进位变量的位置找到全加器的其他变量，进而找到乘法器电路中的全部全加器和半加器，其函数实现在表 2 中给出。

Table 2. Partial function implementation of optimization strategy
表 2. 优化策略的部分函数实现

实现函数	函数描述
init_slice()	初始化二维数组 slice
traverse_column()	为变量标记所在切片，根据所在切片依次放入二维数组 slice 中
collect_carry()	确定全加器和半加器的进位变量
fulladder_reduction()	确定全加器和半加器的所有变量

4.3. 代数系统

根据选择的优化方法和代数系统，进行综合运行，输出一个可计算文件。该文件需要传递给代数系统进行计算，得出验证结果。因此，输出的计算文件是至关重要的。首先，通过解析器找到乘法器中的与门、或门、非门和异或门，通过电路建模将乘法器中的所有门用多项式进行表示。在表 3 中，实现了一个具有这样功能的函数。由于代数系统 Mathematica 和 Singular 的计算方式不同，需要其他的辅助多项式。因此，在表 3 中实现了一个函数 print_slice()，它将各个切片中的辅助多项式打印在计算文件中，并将两个切片之间的联系，通过余多项式进行了连接，使得验证正确的乘法器时，最后余式为零。不仅如此，当进行加法器重写优化时，需将乘法器中的全加器和半加器都用规范多项式表示，并打印到可计算

文件中。当打印全部完成，将该文件生成代数系统的可计算文件，随后传递给 Mathematica 或 Singular 进行计算。

Table 3. Partial function realization of algebraic system
表 3. 代数系统的部分函数实现

实现函数	函数描述
print_adder()	打印全加器和半加器的规范多项式
print_lit()	打印门多项式
print_slice()	打印代数系统需要的辅助多项式

4.4. 实验结果

在实验中，本文使用了具有 $2n$ 个输出位和两个大小为 n 的输入位向量的整数乘法器。使用了与[9]相同的乘法器类型“btor”乘法器和“sp-ar-rc”乘法器。

本次实验使用了一台带有 Ubuntu18.04 虚拟机的电脑，配备 Intel i5-6200U 2.3GHz CPU 和 4GB 主内存。并将时间限制为 1200 秒，主内存限制为 4GB，并且时间都以秒为单位。我们从启动验证工具 AIGMULTOPOLY 开始计算到 Mathematica 或 Singular 完成计算为止，所包含的时间为验证工具生成 Mathematica 和 Singular 计算文件所需的时间和使用计算机代数系统计算的时间。本次实验选择 Mathematica 作为代数系统进行计算，其中 n 表示乘法器的位数，AR 表示加法器重写。实验结果如下表(见表 4)所示：

Table 4. Implementation of verification tools
表 4. 验证工具的实现

mult	n	AR
btor	16	1.56
btor	32	10.08
btor	64	24.54
btor	128	257.06
sp-ar-rc	16	2.82
sp-ar-rc	32	8.23
sp-ar-rc	64	32.54
sp-ar-rc	128	267.27

如表 4 所示，验证 16~128 位乘法器时，利用 C++语言重新实现乘法器的验证工具可以对乘法器进行验证，通过得出可计算文件，将其传递给代数系统进行计算，得出总体验证时间。并且本文的验证工具使用模块的形式，使程序更简单易读，代码的运行与调试也是更方便了。由于 C++语言编程的多样性，我们将来应该进一步优化验证工具和代数系统以提高乘法器的验证效率。但由于计算机的不同，软件版本的不同，得出的结果具有差异性。实验结果表明：通过 C++语言实现的验证工具比 C 语言实现的验证工具更具有可读性和可行性，并为后期验证乘法器提供了更好的平台。

5. 结论

本文基于 Gröbner 基的验证原理，通过 C++语言重新实现了乘法器的验证工具，并通过验证工具读取 AIG 文件，得出可传递到 Mathematica 或 Singular 计算的文件。不仅将整个繁琐的 C 语言代码整理成多文件的形式，使程序调试起来方便，也为后续验证乘法器提供了一个不错的工具。而且利用 C++命名空间中的顺序容器类对代码进行了优化，并且可以正确地实现出来。

使用验证工具和代数系统可以有效验证简单的乘法器，而更复杂和优化的乘法器还需要更复杂的方法。在以后的工作中，我们希望将该方法扩展到浮点运算符，甚至扩展到更具挑战性的乘法器体系结构和其他算术电路。

参考文献

[1] 杨宗凯. 数字专用集成电路的设计与验证[M]. 北京：电子工业出版社, 2004.

[2] Seligman, E., Schubert, T. and Kirankumar, M. (2015) Formal Verification: An Essential Toolkit for Modern VLSI Design. Elsevier Science, Amsterdam.

[3] Randal, Y. and Bryant, E. (1995) Verification of Arithmetic Circuits with Binary Moment Diagrams. 32*nd Design Automation Conference*, San Francisco, 12-16 June 1995, 1-7.

[4] Kaufmann, D., Biere, A. and Kauers, M. (2019) Verifying Large Multipliers by Combining SAT and Computer Algebra. *FMCAD* 2019: 2019 *Formal Methods in Computer Aided Design*, San Jose, 22-25 October 2019, 28-36. https://doi.org/10.23919/FMCAD.2019.8894250

[5] 游珍，薛锦云. 基于 Isabelle 定理证明器算法程序的形式化验证[J]. 计算机工程与科学, 2009, 31(10): 89-93.

[6] Mahzoon, A., Groβe, D. and Drechsler, R. (2018) PolyCleaner: Clean Your Polynomials before Backward Rewriting to Verify Million-Gate Multipliers. *Proceedings of the International Conference on Computer-Aided Design*, San Diego, 5-8 November 2018, Article No. 129. https://doi.org/10.1145/3240765.3240837

[7] Mahzoon, A., Groβe, D. and Drechsler, R. (2020) Towards Formal Verification of Optimized and Industrial Multipliers. *DATE* 2020: 2020 *Design*, *Automation & Test in Europe Conference & Exhibition*, Grenoble, 9-13 March 2020, 544-549. https://doi.org/10.23919/DATE48585.2020.9116485

[8] Ritirc, D., Biere, A. and Kauers, M. (2017) Column-Wise Verification of Multipliers Using Computer Algebra. *FMCAD* 2017: 2017 *Formal Methods in Computer Aided Design*, Vienna, 2-6 October 2017, 23-30. https://doi.org/10.23919/FMCAD.2017.8102237

[9] Ritirc, D., Biere, A. and Kauers, M. (2018) Improving and Extending the Algebraic Approach for Verifying Gate-Level Multipliers. *Design*, *Automation & Test in Europe Conference & Exhibition*, Dresden, 19-23 March 2018, 1556-1561. https://doi.org/10.23919/DATE.2018.8342263

[10] 李雁妮，陈平，王献青. C++程序设计语言[M]. 西安：西安电子科技大学出版社, 2009.

[11] Cox, D., Little, J. and O'Shea, D. (2015) Ideals, Varieties and Algorithms: An Introduction to Computational Algebraic Geometry & Commutative Algebra, Springer International Publishing, Switzerland, 307-338. https://doi.org/10.1007/978-3-319-16721-3

[12] 陈玉福，张智勇. 计算机代数[M]. 北京：科学出版社, 2020.

[13] Sayed-Ahmed, A., Groβe, D., Kühne, U., Soeken, M. and Drechsler, R. (2016) Formal Verification of integer Multipliers by Combining Gröbner basis with Logic Reduction. *DATE* 2016: 2016 *Design, Automation & Test in Europe Conference & Exhibition*, Dresden, 14-18 March 2016, 1048-1053. https://doi.org/10.3850/9783981537079_0248

[14] Biere, A. (2007) The AIGER And-Inverter Graph(AIG) Format Version 20071012. http://fmv.jku.at/aiger

[15] Wolfram Research (1991) Writed, I. Mathematica: A System for Doing Mathematics by Computer. Wolfram Research, Inc., Champaign.

[16] Decker, W., Greuel, G.M., Pfifister, G. and Schonemann, H. (2016) SINGULAR.

[17] Lutz, M., Ascher, D., 陈革. Python 语言入门[J]. Internet：共创软件, 2002(10): 86-86.

[18] 吴建平，尹霞，冯晓冬. Java 程序设计语言[M]. 北京：清华大学出版社, 1997.