

DOI: 10.20098/j.cnki.1001-5132.2023.0602

基于二叉树的 Verilog 多路分支语句综合算法

廖俊鸿, 刘 森, 马铨昱, 储著飞*

(宁波大学 信息科学与工程学院, 浙江 宁波 315211)

摘要: Verilog 多路分支语句是硬件描述语言的一种条件语句, 在处理器、网络交换和数字信号处理等领域应用广泛, 且可通过数据选择器(Multiplexer, MUX)实现资源的极低消耗. 现有基于 And-Inverter Graph 结构的综合工具 ABC 无法有效综合此类电路. 因此, 提出了一种新型逻辑网络表达形式 MAIG (MUX-And-Inverter Graph), 针对 Verilog 多路分支语句中的显式电路给出了基于二叉树的综合算法. 为提高算法的运行效率以及综合质量, 首先提取电路特征参数并进行矩阵列变换, 进而实现 MUX 门的个数和层级减少; 然后根据矩阵的 0、1 取值, 通过二叉树优化算法划分矩阵递归生成面积小、时延低的 MAIG. 与学术界综合工具 ABC 相比, 所提算法在工艺映射前电路逻辑门的个数和深度平均优化 72%和 52%, 工艺映射后电路面积和时延平均改善 67%和 33%.

关键词: Verilog 多路分支语句; 数据选择器; 二叉树; MAIG

中图分类号: TP391.41

文献标志码: A

文章编号: 1001-5132 (2024) 02-0010-08

逻辑综合是电子设计自动化(Electronic Design Automation, EDA)流程中的重要组成部分, 属于前端设计的核心^[1]. 逻辑综合经过翻译、逻辑优化和工艺映射等步骤, 将用户设计的数字电路转换成工艺库网表. Verilog 多路分支语句的综合是影响专用集成电路(Application Specific Integrated Circuit, ASIC)和现场可编程门阵列(Field Programmable Gate Array, FPGA)逻辑综合质量的重要环节. Verilog 多路分支语句是电路的行为级表示, 其逻辑功能等价于多选一数据选择器(N -to-1 Multiplexer, $N:1$ MUX)树, MUX 广泛应用于处理器、网络交换和数字信号处理等领域^[2], 因此, 针对多路分支语句的综合优化往往离不开 $N:1$ MUX 的综合优化.

现有研究中针对 $N:1$ MUX 的综合优化方法可分为 3 类: 基于二元决策图(Binary Decision Diagrams, BDD)的方法^[3-5]、数据选择器变换法^[6-8]

和数据选择器结构分解法^[9-10]. 第 1 类基于 BDD 的方法将 BDD 节点看成 MUX, 路径对应 MUX 连接关系, 但此类方法只适用于优化较小规模电路, 还未推广至逻辑综合的翻译步骤. 第 2 类数据选择器变换法通过结构设计替换原有 MUX 结构来优化电路, 但受到额外开销的限制. 第 3 类结构分解法通过搜索算法寻找最优 MUX 结构, 搜索 $N:1$ MUX 最佳结构已被证明是 NP-hard 问题, 故此类方法无法推广至大规模电路. Verilog 多路分支语句根据电路特征可分为显式电路和隐式电路, 现有综合工具 ABC 在综合 Verilog 多路分支语句时, 将其综合成与非图(And-Inverter Graph, AIG)表示的门级电路, 但显式电路存在紧凑的 MUX 结构, 因此现有的研究或综合工具还无法有效解决 Verilog 多路分支语句中显式电路的综合优化.

针对 Verilog 多路分支语句中显式电路的综合优化问题, 本文提出了一种新型逻辑网络表达形

收稿日期: 2023-06-09.

宁波大学学报(理工版)网址: <http://journalg.nbu.edu.cn/>

基金项目: 国家自然科学基金(62274100); 宁波市重点研发计划(2023Z071).

第一作者: 廖俊鸿, 硕士研究生, 主要研究方向: 逻辑综合与优化. E-mail: liaojunhong5725@foxmail.com

*通信作者: 储著飞, 教授, 主要研究方向: 逻辑综合与优化. E-mail: chuzhufei@nbu.edu.cn

式 MAIG (MUX-And-Inverter Graph) 以及基于二叉树的综合算法. 为提高算法的运行效率以及综合质量, 首先提取电路特征参数并将数据进行一系列的预处理, 然后通过综合算法中的二叉树优化算法翻译优化显式电路, 最后生成面积小、时延低的门级网表. 与目前学术界的综合工具 ABC 相比, 本文算法生成的电路在工艺映射之前电路逻辑门个数、层级分别改善 72% 和 52%, 在工艺映射之后电路面积、时延平均优化 67% 和 33%.

1 研究背景

1.1 逻辑函数表达方式

逻辑函数表达方式与逻辑综合优化算法密切相关. 随着集成电路的发展, 电路规模增加, 逻辑函数表达方式从传统的真值表/卡诺图, 到 SOP、BDD, 再发展到现今的有向无环图 (Directed Acyclic Graph, DAG). 基于不同逻辑门表述的 DAG 发展出多种逻辑优化方法, 例如与门和异或门驱动的 XAIG (Xor-And-Inverter Graph)^[11] 在异或和与或密集型电路上应用广泛; 与非门驱动的 AIG^[12-13] 和 Majority 门驱动的 MIG (Majority Inverter Graph)^[14] 在电路优化中同样取得了不错的效果.

BDD 是一个 DAG, BDD 中每一个节点表示一个布尔变量, 每条边表示该变量在布尔函数中的取值, BDD 根节点表示该布尔函数的输出. 1986 年提出的 ROBDD (Reduced Ordered BDD) 通过固定变量顺序并进行约简得到逻辑函数的典范表达^[15], 通常称作基于 BDD 的逻辑综合大多指的是 ROBDD.

图 1 所示是 $f = x_1x_2 + x_1x_3 + x_2x_3$ 的 BDD 和化简后的 ROBDD.

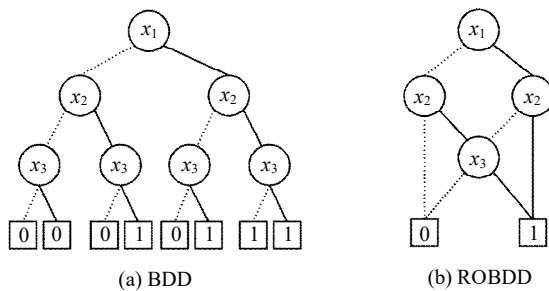


图 1 函数 f 的 BDD 和 ROBDD

图 1 中内部节点 x_1 、 x_2 、 x_3 表示输入变量, 实线表示布尔变量取“1”, 虚线表示布尔变量取“0”, 0 和 1 节点表示网络输出.

1.2 Verilog 多路分支语句和 MUX

1.2.1 MUX 概述

MUX 是数字电路的基本单元, McCluskey^[16] 将 MUX 定义为“从多个输入端选择一个输入信号并将其传递至输出端的电路”. MUX 的输出信号取决于一个或多个控制信号, 这些信号可以控制 MUX 将哪个输入信号传递到输出端. MUX 通常表示为 “MUX($a, b, c, d, \dots, sel$)”, 其中 a, b, c, d 等表示输入信号, sel 表示控制信号. 本文所用 MUX 为 2 选 1 数据选择器, $f = \text{MUX}(a, b, s)$ 的逻辑表达式为 $f = as + b\bar{s}$.

1.2.2 多路分支语句综合成 MUX 树

在 Verilog 中, 多路分支语句通常被用来实现数据选择的功能, 且多路分支语句中的显式电路存在紧凑的 MUX 结构, 现今的综合工具 ABC 在综合优化这类电路时, 主要将其翻译成 AIG, 破坏了其极为紧凑的 MUX 结构, 不利于电路综合优化. 多路分支语句根据控制条件是否蕴含在 “default” 或 “other” 项中, 将电路分为隐式电路和显式电路. 显式电路其控制条件均已显式地列出, 不需要使用 “default” 或 “other” 项进行描述, 如图 2 所示.

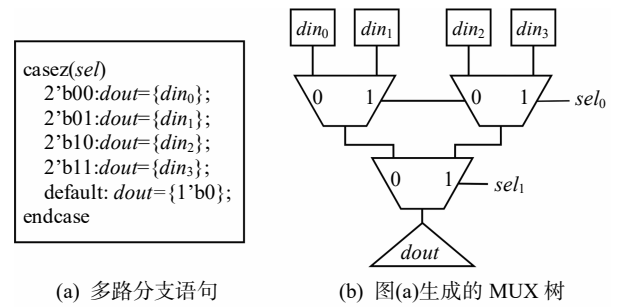


图 2 Verilog 多路分支语句综合成 MUX 树

图 2(a) 中, 多路分支语句的所有条件均已列出, 是典型的显式电路, 对该电路的控制信号取 0、取 1 枚举即可生成逻辑紧凑的 MUX 树, 如图 2(b) 所示. 图 2(b) 表示的 MUX 树实际上实现的是一个 4 选 1 数据选择器的功能, 电路输出为 $dout$, 输入信号为 din_0 、 din_1 、 din_2 、 din_3 , 控制信号为 sel_0 、 sel_1 . 若 $sel_0 = 0$, $sel_1 = 0$, 则 $dout = din_0$, 对应图 2(a) 中的 “2'b00:dout={ din_0 }” 项.

2 基于二叉树的综合算法

基于二叉树的综合算法针对 Verilog 多路分支语句中的显式电路进行翻译和优化,在逻辑功能一致的前提下将多路分支语句综合成面积小、时延低的门级网表.本节分别介绍二叉树综合算法总体流程、多路分支语句解析、MAIG 以及本文的核心算法——二叉树优化算法.

2.1 二叉树综合算法总体流程

二叉树综合算法输入是 Verilog 多路分支语句,输出为 MAIG 表示的门级网表.多路分支语句经过接口解析电路控制模块生成矩阵 addr_mat , 输出模块生成矩阵 data_mat ; 为避免多路分支语句控制信号的无关项(don't care)和“0, 1”状态分布不均的影响,对矩阵进行重新排列;最后通过二叉树优化算法生成门级网表.

2.2 多路分支语句解析和 MAIG

2.2.1 语句解析和矩阵处理

多路分支语句通常由控制模块和输出模块组成.控制模块包含所有选择条件,选择条件由控制信号状态集组成,控制信号存在“0”“1”和“don't care”状态,“don't care”状态表示该变量取值任意,通常使用“?”表示.输出模块根据控制模块输出对应的输入信号,如图 3(a)所示,若条件“3b'?10”为真,电路输出对应的输入信号 din_1 .为方便接口解析,多路分支语句的条件之间存在互斥性,且带有“default”项.

语句解析:将控制信号“0”“1”和“don't care”状态分别取值为 0、1 和 -1;输出信号的 1'b0 和 1'b1 分别由 0 和 1 表示, $\text{din}_0, \text{din}_1, \dots, \text{din}_n$ 分别表示为 $2, 4, \dots, 2 \times n + 2$, $\overline{\text{din}_0}, \overline{\text{din}_1}, \dots, \overline{\text{din}_n}$ 分别表示为

$3, 5, \dots, 2 \times n + 3$. 例如图 3(a)中“3'b001”解析成“001”,“ din_0 ”解析成“2”,最终生成图 3(b)所示的 addr_mat 和 data_mat 矩阵.

矩阵处理:二叉树优化算法通过地址矩阵取值进行划分,若取值为 0,则将该行划分至左子矩阵;若取值为 1,则将该行划分至右子矩阵;若取值为 -1,则将该行分别划分至左和右子矩阵,即增加了子矩阵的行数影响电路综合质量.为了避免控制变量的“don't care”项和某列“0, 1”状态个数差较大而影响电路的综合质量,一般通过矩阵列变换改变矩阵特性,即将“don't care”项较多的列移至地址矩阵的后几列,将“0, 1”状态个数差小的列往前移,例如图 3(b)的第 1 列存在“don't care”项,需移至图 3(c)的最后一列,图 3(b)的第 2 列“0, 1”个数差为 0,则移至图 3(c)的第 1 列,每列对应的控制变量也随之改变,由原来的 $\text{sel}_2, \text{sel}_1, \text{sel}_0$ 变成 $\text{sel}_1, \text{sel}_0, \text{sel}_2$.

2.2.2 MAIG

MAIG 是一种新型逻辑网络,属于 DAG,网络中的逻辑函数基于 2:1 MUX、2:1 AND 和 NOT 门进行逻辑表达.基于 MAIG 逻辑结构,本文提出了二叉树优化算法,用于翻译优化多路分支语句.图 4 所示是通过图 3(a)多路分支语句综合生成的 MAIG,图 4(a)和图 4(b)分别表示通过本文算法在矩阵处理前和矩阵处理后综合生成的 MAIG.图中虚线表示 MUX 门的控制信号;○表示变量取反; dout 为输出;节点 1~3 为控制信号 $\text{sel}_0 \sim \text{sel}_2$,节点 4~8 为输入信号 $\text{din}_0 \sim \text{din}_4$.图 4(a)生成网络需要 6 个 MUX 逻辑门和 3 个 AND 门,图 4(b)只需 5 个 MUX 逻辑门.结果比对证明本文算法通过矩阵处理有利于优化电路性能.矩阵变换后综合过程如

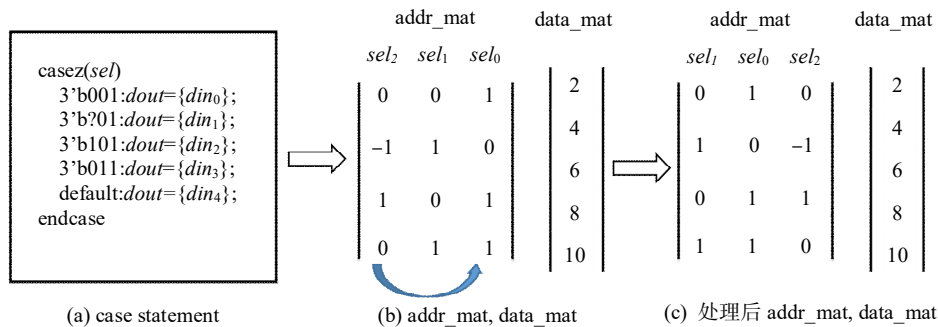


图 3 单输出多路分支语句解析图

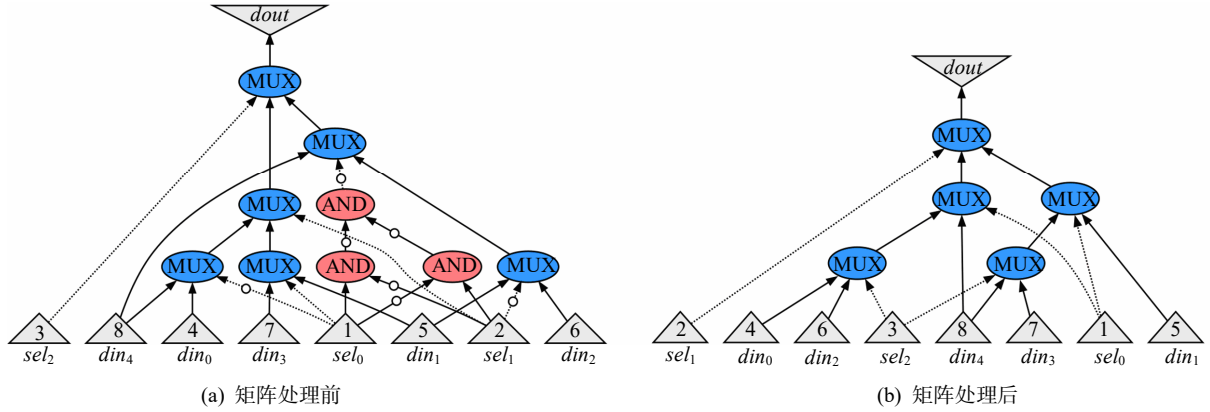


图4 多路分支语句综合成 MAIG

下: 首先对图 3(c)第 1 列地址矩阵进行 0、1 划分, 生成 2 个子矩阵(1, 3 行)和(2, 4 行), 并在图 4(b)中创建最高层 MUX 逻辑门, MUX 门控制逻辑端连接 sel_1 ; 然后对 2 个子矩阵第 2 列进行二次划分, 生成 3 个子矩阵(1, 3 行)、(2 行)和(4 行), 并在次高层级创建 2 个 MUX 逻辑门, MUX 门控制逻辑端连接 sel_0 ; 最后只需对(1, 3 行)进行划分和第 4 行进行划分并补充“default”条件, 同理创建最底层的 2 个 MUX 逻辑门, MUX 门控制逻辑端为 sel_2 , 将变量连接 MUX 逻辑门的 0、1 输入端。

2.3 二叉树优化算法

多路分支语句可由条件集 S_i 、输出信号 $dout_j$ 、输入信号 din_x 描述。条件集 S_i 由控制信号 sel_y 表示, 见式(1); 若某一条件下 sel_y 全取真, S_i 可由式(2)表示; 多路分支语句逻辑函数表达见式(3)。

$$S_i = F(sel_{y-1}, sel_{y-2}, \dots, sel_1, sel_0), \quad (1)$$

$$S_i = sel_{y-1} \wedge sel_{y-2} \wedge \dots \wedge sel_1 \wedge sel_0, \quad (2)$$

$$dout_j = S_i \wedge din_x, \quad (3)$$

$$1 \leq i \leq m_1, 1 \leq j \leq n_1, 1 \leq x \leq X, 1 \leq y \leq n,$$

式中: m_1 、 n_1 为 $data_mat$ 行列数; X 为输入(被选)信号位宽(din_x); n 为控制信号位宽。式(1)简要表述了 MAIG 中所有 MUX 门控制逻辑函数, 即根据控制端将输入索引至输出。式(3)中 $dout_j$ 通过二叉树实现极为方便, 即二叉树的根节点、子节点可以表示控制变量, 叶子节点可以表示输入信号 din_x 。根据上述特性可将 Verilog 多路分支语句通过控制变量的 0 或 1 取值生成二叉树的根节点、子节点和叶子节点, 最后将节点替换成 MUX 逻辑门、AND 逻辑门以及相应的连接关系。

2.3.1 二叉树优化算法流程

二叉树优化算法是通过二叉树节点特性, 将根节点、子节点看成 MUX 门, 节点变量对应 MUX 的控制信号, 节点左枝表示变量取非, 节点右枝表示变量取真, 对应 MUX 的 0、1 输入端, 图 5 中通过虚实线表示。二叉树叶子节点对应被选信号, 其综合过程如图 5 所示。

图 5(a)是多路分支语句解析得到的矩阵 $addr_mat$ 、 $data_mat$, 其中控制信号位宽为 n , 条件集合任意; “default”条件只用于补充某些 MUX 门的输入端, 故不加以处理。二叉树优化算法通过对 $addr_mat$ 进行 0、1 分割同时生成 MAIG。例如通过变量 sel_{n-1} 取值 0 或 1, 即 $addr_mat$ 第 1 列 0、1 取值, $addr_mat$ 和 $data_mat$ 可分别生成两个子矩阵; 同时生成图 5(b)中的根节点 sel_{n-1} , 其中虚线表示该变量取值为 0, 实线为 1; 同理对子矩阵 sel_{n-2} 取值进行 0、1 划分, 生成新的子矩阵, 并生成节点 sel_{n-2} ; 以此类推, 矩阵划分至只剩一行则算法终止, 生成 MAIG。

图 6 所示是二叉树优化算法伪代码。输入: 矩阵 $addr_mat$ 、 $data_mat$ 。输出: MAIG。第 1 行声明二叉树优化函数 OABT, 其输入参数有矩阵 $addr_mat$ 、 $data_mat$, 控制变量 j 。第 2 行判断矩阵是否需要划分。第 3~9 行遍历 $addr_mat$ 矩阵的列, 遍历至第 j 列时, 根据控制信号 0、1 取值划分生成 2 个子矩阵。第 10 行调用 MAIGgen 函数生成 MUX 门, 并确定控制端、输入端的连接关系。第 11 行根据变量 j 递增变化, 确保网络自顶向下创建。第 12~13 行以递归方式划分矩阵, 以达到电路自顶向

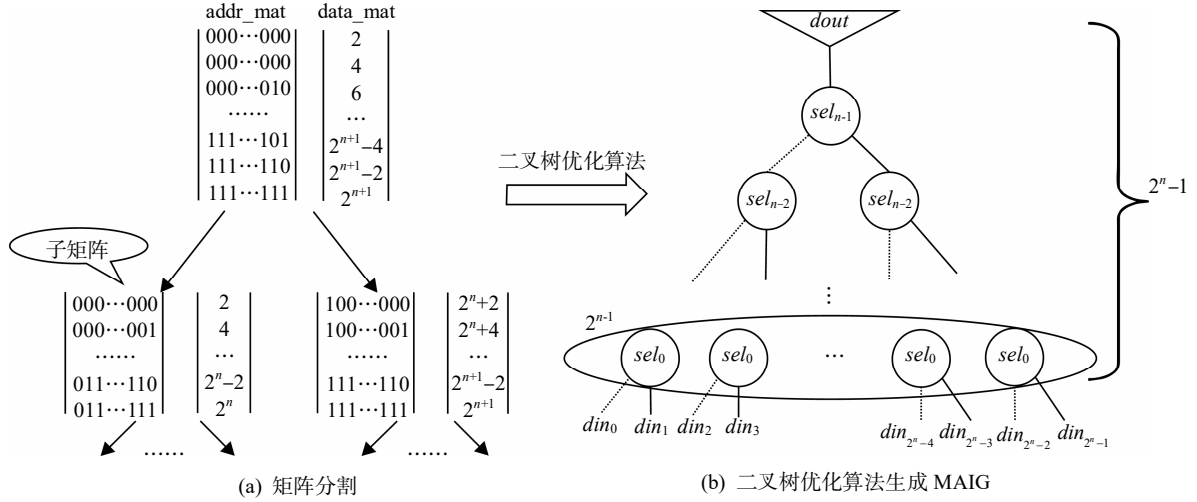


图5 二叉树优化算法生成 MAIG

下创建网络的目标. 第14~16行表示 addr_mat 划分至只剩一行, 则通过函数 OABTend 生成最底层网络; 若 $j = n-1$, 最底层 MUX 只需一位控制信号连接控制逻辑端, 若 $j < n-1$, MUX 控制端由多位控制信号相与生成; 最后算法终止, 生成 MAIG.

Algorithm 1

```

Input:  $\text{addr\_mat}$ ,  $\text{data\_mat}$ 
Output: MAIG
1 procedure OABT( $\text{addr\_mat}$ ,  $\text{data\_mat}$ ,  $j$ )
2   if  $\text{addr\_mat.size} \geq 2$ 
3     for  $i$  from 0 to  $\text{addr\_mat.size}$  do
4       if  $\text{addr\_mat}[i][j] = \text{flag}$  then
5          $\text{addr0.insert}$ ,  $\text{data0.insert}$ ;
6       else
7          $\text{addr1.insert}$ ,  $\text{data1.insert}$ ;
8     end
9   end
10  MAIGgen( $j$ );
11   $j = j + 1$ ;
12  OABT( $\text{addr0}$ ,  $\text{data0}$ ,  $j$ );
13  OABT( $\text{addr1}$ ,  $\text{data1}$ ,  $j$ );
14 else
15  OABTend( $\text{addr\_mat}$ ,  $\text{data\_mat}$ ,  $j$ );
16 end
17 return MAIG;

```

图6 二叉树优化算法

2.3.2 算法分析

二叉树优化算法优势在于生成的 MUX 门控制逻辑简单, 对显式电路或控制条件集趋近 2^n 的电路优化效果显著. 对于显式电路, 二叉树优化算法通过控制变量的 0、1 状态进行划分可以极简描述所有条件, 生成结构紧凑的 MAIG. 二叉树优化算法对控制条件集远小于 2^n 的电路优化效果不佳, 故此方法只适用于综合显式或接近显式的电路.

该算法通过对矩阵行列进行划分, 划分次数与电路输出数目成正比, 故此算法的时间复杂度为 $O(m \times n \times n_1)$, 空间复杂度为 $O(m \times n_1)$, 变量 m 、 n 是 addr_mat 行列数, 变量 n_1 是 data_mat 列数.

3 实验结果

所提算法用 C++ 语言实现, 在 Ubuntu 20.04 操作系统下, 通过 g++ 编译后运行, 程序运行环境为 Intel Core i5-9500, 3.00 GHz CPU, 4 GB RAM, 本文问题来源于第四届中国 EDA 精英挑战赛.

本文算法读入一个 Verilog 文件, 通过接口解析出 addr_mat 和 data_mat , 再通过算法优化电路. 为了说明本文方法的优化效果, 分别与综合工具 ABC 对比工艺映射前和工艺映射后的结果, 见表 1 和表 2. 表 1 为工艺映射之前结果, 表中“电路”列为显式电路名称; “输入/输出”列为相应电路输入/输出数目; “逻辑门”表示映射前电路所需逻辑门个数; “逻辑深度”表示电路逻辑深度; “ABC”表示应用 ABC 优化所得结果, “本文方法”表示应用本文算法所得优化结果; “GEOMEAN”表示 15 个测试案例优化后的几何平均值. 其中, “ r_1 ” “ r_2 ”分别表示本文方法相对 ABC 方法在逻辑门个数和逻辑深度上对电路改善的百分比, 计算方法见式 (4), 式中 N_x 表示采用 ABC 方法优化后电路逻辑门个数或逻辑深度; N_y 表示采用本文优化方法所得电路逻辑门个数或逻辑深度.

$$r_i = (N_x - N_y) / N_x \times 100\%. \quad (4)$$

表 1 工艺映射前优化情况

电路	输入/输出	ABC ^[17]		本文算法			
		逻辑门	逻辑深度	逻辑门	逻辑深度	$r_1/\%$	$r_2/\%$
b32_com_c0	261/8	584	12	248	5	58	58
b32_com_c10	261/8	579	12	259	7	55	42
b32_repeat_c0	133/8	559	12	245	5	56	58
b32_shift1_c0	44/8	513	11	130	5	75	55
b32_shift3_c0	106/8	565	11	218	5	61	55
b32_simp3_c0	69/8	191	9	56	3	71	67
b1024_all_const	8 202/8	2 846	25	1 332	15	53	40
b2048_all_const	16 395/8	5 357	24	2 373	16	56	33
b2048_com_c0	16 395/8	37 409	21	16 376	11	56	48
b2048_com_c10	16 395/8	37 394	21	16 384	13	56	38
b2048_repeat_c0	8 203/8	37 478	21	16 375	11	56	48
b2048_shift1_c0	2 066/8	39 616	21	8 194	11	79	48
b2048_shift3_c0	6 160/8	37 711	21	14 330	11	62	48
b2048_sim8_c0	2 059/8	37 334	21	2 040	8	95	62
b4096_sim8_c0	2 060/8	61 649	24	2 040	8	97	67
GEOMEAN		4 980.3	16.8	1 408.0	8.1	72	52

表 2 工艺映射后优化情况

电路	输入/输出	ABC ^[17]		本文算法			
		面积	时延	面积	时延	$r_3/\%$	$r_4/\%$
b32_com_c0	261/8	691.6	9.3	347.2	6.0	50	35
b32_com_c10	261/8	686.6	9.3	347.6	6.0	49	35
b32_repeat_c0	133/8	666.6	8.6	343.0	6.0	49	30
b32_shift1_c0	44/8	619.8	8.6	182.0	6.0	71	30
b32_shift3_c0	106/8	673.8	8.6	305.2	6.0	55	30
b32_simp3_c0	69/8	233.8	7.8	78.4	3.6	66	54
b1024_all_const	8 202/8	3 314.0	16.0	1 756.8	11.7	47	27
b2048_all_const	16 395/8	6 236.2	16.2	3 187.0	12.9	49	20
b2048_com_c0	16 395/8	44 828.2	17.4	22 926.4	13.2	49	24
b2048_com_c10	16 395/8	44 798.4	17.6	22 925.6	13.2	49	25
b2048_repeat_c0	8 203/8	44 924.0	17.4	22 925.0	13.2	49	24
b2048_shift1_c0	2 066/8	47 826.4	17.4	11 471.6	13.2	76	24
b2048_shift3_c0	6 160/8	45 195.4	17.8	20 062.0	13.2	56	26
b2048_sim8_c0	2 059/8	44 766.0	17.6	2 856.0	9.6	94	45
b4096_sim8_c0	2 060/8	74 857.8	18.7	2 856.0	9.6	96	49
GEOMEAN		5 952.6	13.1	1 952.4	8.8	67	33

注: ABC 优化命令 “resyn2;&dc2; &syn2 -d” 等; ABC 映射命令 “&nf -F 10 -A 10”。

由表 1 数据可知, 对于显式电路, 本文算法的优化效果明显优于综合工具 ABC。ABC 对多路分支语句进行综合时先生成 AIG, 但显式电路存在紧

凑的 MUX 结构, ABC 在综合过程中破坏了这一结构, 最终影响综合质量, 这也是本文算法在工艺映射前比 ABC 在逻辑门个数和逻辑深度上改善

72%、52%的根本原因. 测试案例中电路 b2048_sim8_c0、b4096_sim8_c0 电路优化尤为明显, 逻辑门个数分别改善 95%、97%, 其原因是多路分支语句中存在大量逻辑等价的模块, 这些模块在 MAIG 网络中呈现高度对称, 即 MUX 门 0、1 输入端逻辑功能等价, 通过结构哈希所得电路所需逻辑门个数极少, 但综合工具 ABC 无法直接提取这些电路信息, 故导致优化后的电路极其庞大.

表 2 是工艺映射后所得结果, 将本文算法与综合工具 ABC 所得结果进行对比. “面积”列代表映射后电路面积; “时延”表示映射后电路时延; “ r_3 ”和 “ r_4 ”分别表示本文方法相比 ABC 方法对电路面积和时延改善的百分比, 计算方法见公式(4), 式中 $i=3, 4$; N_x 表示采用 ABC 方法所得电路面积或时延; N_y 表示采用本文优化方法所得电路面积或时延. 测试案例中电路 b2048_sim8_c0、b4096_sim8_c0 面积优化尤为明显, 面积分别改善 94%、96%, 电路时延最高改善 54%. 平均而言, 本文算法与综合工具 ABC 相比在面积和时延上改善 67%、33%.

4 结束语

本文提出了一种基于二叉树的综合算法, 用于实现行为级电路翻译成门级网表. 本文的主要贡献是将 Verilog 多路分支语句中的显式电路综合优化成面积小、时延低的 MAIG. 本文算法与综合工具 ABC 相比, 在 15 个测试案例中工艺映射前电路逻辑门个数和逻辑深度平均改善 72%和 52%, 最高优化 97%和 67%; 工艺映射后电路“面积”和“时延”平均优化 67%和 33%, 最高优化 96%和 54%. 实验结果表明, 本文算法与综合工具 ABC 相比, 所得结果在映射前后逻辑门个数、逻辑深度、面积和时延上均更优. 在未来的研究中, 可考虑优化 Verilog 多路分支语句中的隐式电路, 即原先 addr_mat 矩阵划分至最后一行, 算法终止, 变成多行, 或许能取得更好的结果.

参考文献:

- [1] 储著飞, 潘鸿洋. 基于布尔可满足性的精确逻辑综合综述[J]. 电子与信息学报, 2023, 45(1):14-23.
- [2] METZGEN P. A high performance 32-bit ALU for

- programmable logic[C]//Proceedings of the 2004 ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays. New York: ACM, 2004:61-70.
- [3] YANG C G, CIESIELSKI M, SINGHAL V. BDS: a BDD-based logic optimization system[C]//Proceedings of the 37th Annual Design Automation Conference. New York: ACM, 2000:92-97.
- [4] FEY G, DRECHSLER R. Minimizing the number of paths in BDDs: theory and algorithm[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2005, 25(1):4-11.
- [5] KOHUTKA L, PISTEK P. Faster synthesis of combinational logic based on multiplexer trees and binary decision diagrams[C]//2014 IEEE 12th IEEE International Conference on Emerging eLearning Technologies and Applications (ICETA), 2014:239-244.
- [6] METZGEN P, NANCEKIEVILL D. Multiplexer restructuring for FPGA implementation cost reduction [C]//Proceedings of the 42nd Annual Design Automation Conference. New York: ACM, 2005:421-426.
- [7] LI N S, HUANG J D, HUANG H J. Low power multiplexer tree design using dynamic propagation path control[C]//APCCAS 2008 - 2008 IEEE Asia Pacific Conference on Circuits and Systems, 2008:838- 841.
- [8] PISTEK P, KOLESÁR M, JELEMENSKÁ K. Optimization of multiplexer trees using modified truth table[C]//2010 International Conference on Applied Electronics, 2010:1-4.
- [9] MITRA S, AVYA L J, MCCLUSKEY E J. Efficient multiplexer synthesis techniques[J]. IEEE Design & Test of Computers, 2000, 17(4):90-97.
- [10] BASIRI M M A, MAHAMMAD SK N. High speed multiplexer design using tree based decomposition algorithm[J]. Microelectronics Journal, 2016, 51:99-111.
- [11] HÁLECEK I, FISER P, SCHMIDT J. On XAIG rewriting [C]//Proceeding of 26th International Workshop on Logic & Synthesis (IWLS), Austin, 2017:89-96.
- [12] YU C X, CIESIELSKI M, CHOUDHURY M, et al. DAG-aware logic synthesis of datapaths[C]//Proceedings of the 53rd Annual Design Automation Conference, 2016: 1-6.
- [13] SOEKEN M, CHATTOPADHYAY A. Unlocking efficiency and scalability of reversible logic synthesis using conventional logic synthesis[C]//Proceedings of the 53rd Annual Design Automation Conference, 2016:1-6.
- [14] CHU Z F, HAASWIJK W, SOEKEN M, et al. Exact

- synthesis of Boolean functions in majority-of-five forms [C]//2019 IEEE International Symposium on Circuits and Systems (ISCAS), 2019:1-5.
- [15] BRYANT R E. Graph-based algorithms for Boolean function manipulation[J]. IEEE Transactions on Computers, 1986, 100(8):677-691.
- [16] MCCLUSKEY E J. Logic design principles with emphasis on testable semicustom circuits[M]. Englewood NJ: Prentice-Hall, Inc., 1986.
- [17] BRAYTON R, MISHCHENKO A. ABC: an academic industrial-strength verification tool[C]//International Conference on Computer Aided Verification, 2010:24-40.

Binary tree-based synthesis algorithm for Verilog case statement

LIAO Junhong, LIU Sen, MA Chengyu, CHU Zhufei

(Faculty of Electrical Engineering and Computer Science, Ningbo University, Ningbo 315211, China)

Abstract: Verilog case statements are conditional statements in the hardware description language. They are widely used in fields such as processors, network switches, and digital signal processing. They can optimize resource distribution in terms of efficiency through the use of multiplexers. However, the existing synthesis tool ABC which is based on the And-Inverter Graph logic representation cannot effectively synthesize such circuits. Therefore, in this paper we propose a novel logic representation named MUX-And-Inverter Graph (MAIG), and present a binary tree-based synthesis algorithm specifically for explicit circuits within Verilog case statements. In order to improve the efficiency of the algorithm and the quality of synthesis, the first step is to extract circuit feature parameters and perform matrix column transformation. The proposed process as a result reduces the number of MUX gates and levels. Next, depending on the 0 and 1 values of the matrix, a binary tree optimization algorithm is applied to partition the matrix which is implemented to recursively generate MAIG with a smaller area and smaller delay. Compared to ABC, the proposed algorithm achieves an average optimization by 72% in the number of logic gates and 52% in logic depth before technology mapping came into being. It also achieves an average improvement by 67% in circuit area and 33% in delay reduction after technology mapping became available.

Key words: Verilog case statement; multiplexer; binary tree; MAIG

(责任编辑 韩 超)