

DOI: 10.20098/j.cnki.1001-5132.2023.0305

基于树形 Mux 的逻辑电路优化

于宗源, 廖春柳, 胡 张, 王伦耀*

(宁波大学 信息科学与工程学院, 浙江 宁波 315211)

摘要: 为实现用 case 语句描述的逻辑电路的面积和延迟优化, 提出了一种基于树形 Mux 的逻辑电路优化方法. 该方法先将 case 语句转换为树形 Mux, 通过合并 case 语句实现 Mux 树中 Mux 门的个数和层级减少, 并通过化简地址逻辑实现地址再编码电路的精简, 进而实现映射后电路面积与延迟的优化. 提出的算法使用 C++ 语言实现, 电路面积和延迟优化结果由常用学术开源 EDA 工具 abc, 结合国内 EDA 公司提供的映射库得到. 实验结果表明, 相比于 abc 工具, 使用该方法得到的面积和延迟优化分别提升了 26% 和 21%.

关键词: Mux 树; case 语句综合; 逻辑优化; Verilog HDL

中图分类号: TP391.41

文献标志码: A

文章编号: 1001-5132 (2023) 05-0069-07

从电路的功能角度, 数字电路内部可以分为数据通路和控制逻辑两大部分. 数据选择器 (Multiplexer, Mux) 可以同时作为构建数据通路和控制逻辑的模块, 且可以实现任何逻辑功能^[1], 可以用在加法器^[2]、乘法器^[3]、物理不可克隆函数 (Physical Unclonable Function, PUF)^[4]等领域.

Mux 在现场可编程门阵列 (Field-Programmable Gate Array, FPGA) 中应用尤其广泛, 在传统的岛式 FPGA^[5] 中, FPGA 的设计指标很大程度上取决于其架构. 文献[6]提出了一种二级 Mux 结构来进行面积和延迟的优化. 在对基于 Altera 公司 FPGA 的 120 个实际的客户设计分析中发现, FPGA 中 Mux 的面积往往占了 25% 以上^[7]. 文献[4]通过 Mux 链组建 PUF 结构来保障硬件安全. 另外, 在逻辑电路的行为级描述中, 使用频率很高的 if、else 与 case 语句也可以映射成 Mux 电路. 所以基于 Mux 的数字逻辑电路优化是逻辑综合领域中一个重要的研究方向.

在对 Mux 的研究中, Mux 的优化方法主要有两种, 一种是使用结构变换^[7-9], 即使用不同规格的 Mux 或其他门来代替原电路的 Mux, 另一种是使

用二元决策图 (Binary Decision Diagram, BDD)^[10]. 文献[7]使用逻辑共享来优化 Mux 树, 是针对于 4-LUT 且依赖于 4 选 1 的 Mux 实现优化. 文献[8]指出 n 选 1 的 Mux 可设计方案的数量是一个 NP (Non-deterministic Polynomial) 完全问题, 随着 n 的增加, n 选 1 的 Mux 设计方案呈指数增加. 文献[9]在此基础上提出了一种树分解方法, 从所有可能的分解中选择最佳方案, 但该方法的效率受限于电路规模. 文献[10]提出了一种基于 BDD 的优化算法, 其优化效果会受变量输入顺序影响. 而本文提出的方法具有较好的普适性, 可以有效处理规模较大的电路.

基于 Mux 的逻辑电路的结构大致由两部分组成, 即树形的 Mux 阵列和控制每个 Mux 选通状态的地址电路. 因此, 针对 Mux 电路结构的优化可以通过优化 Mux 阵列和地址电路来实现. 通过减少 Mux 阵列中的 Mux 使用数量和地址电路的逻辑门数量实现电路面积的优化, 也可以通过压缩 Mux 阵列和地址电路的层级数实现电路延时的优化.

基于 Mux 的逻辑电路的结构优化主要利用 Mux 逻辑门的如下特性, 当一个 Mux 门的数据输

收稿日期: 2023-03-03.

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 (61871242); 浙江省自然科学基金 (LY19F040004).

第一作者: 于宗源 (1998-), 男, 山东淄博人, 在读硕士研究生, 主要研究方向: 逻辑综合与优化. E-mail: 1546313887@qq.com

*通信作者: 王伦耀 (1972-), 男, 浙江宁波人, 教授, 主要研究方向: 逻辑综合与优化. E-mail: wanglunyao@nbu.edu.cn

入都相同时,该 Mux 的输出与输入之间可以用导线代替而不改变逻辑功能,进而实现电路的简化.因此,实现基于 Mux 的逻辑电路优化的一个有效手段就是确定某个 Mux 门,让它的数据输入都一样,达到省略该 Mux 门的目的.

本文主要研究用硬件描述语言 Verilog 中 if-else 和 case 语句描述的逻辑电路的优化.此类描述对应的逻辑功能一般可以用 Mux 电路实现.考虑到 if-else 与 case 之间可以相互转化,为叙述方便,同时不失一般性,本文以 case 语句为例来说明优化过程.通过将 case 语句描述转化为树形 Mux 结构,然后简化 Mux 树,实现对 case 语句描述的逻辑电路的优化.为减少第三方 EDA 工具对实验结果的影响,本文先将优化后的 Mux 树转化为基于 assign 语句描述的准网表结构,然后用学术上通用的 EDA 工具 abc 进行逻辑综合,得到优化后电路的性能参数,并将本文结果与 abc 结果进行比较.

1 基于 Mux 的逻辑电路结构

Verilog 中一个具有 W 个分支的 case 语句常被映射为具有 W 选 1 功能的 Mux 树结构^[10].通常 Mux 树可由具有 2 选 1 或 4 选 1 功能的 Mux 门实现^[11],本文采用前者,并用 Mux_2 表示具有 2 选 1 功能的 Mux 门.

图 1 为基于 Mux 门的逻辑电路结构,其中包含了两部分,即 Mux 树和地址再编码电路.

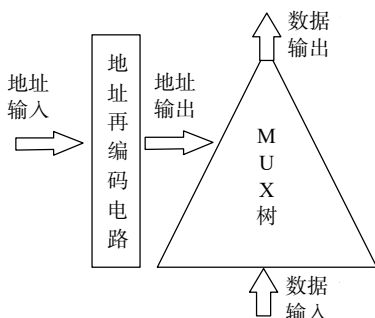


图 1 基于 Mux 的逻辑电路结构

以 Verilog 中的一个 case 语句为例来说明图 1 中 Mux 树和地址再编码电路的作用.

图 2(a)中 case 语句描写的电路功能可以理解为一个具有 8 个数据输入端的数据选择器,其地址信号为 4 位宽的 *sel*. 输入的 8 个数据中部分是相同

的. 与图 2(a)中 case 语句对应的 Mux 树结构如图 2(b)所示. 图 2(b)中每个梯形表示一个 Mux_2 门,梯形下部的 2 个连线为数据输入,上部的连线为输出,腰部的连线为地址输入. 梯形中的“1”和“0”分别表示当地址输入为 1 或 0 时,与“1”端或“0”端相连的数据输入将被选中作为 Mux_2 的输出.

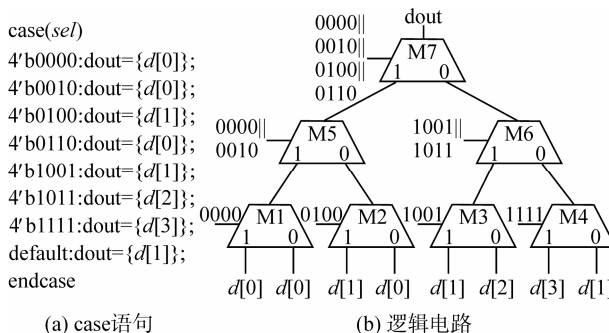


图 2 Verilog 的 case 语句及其用 Mux_2 实现的逻辑电路

在图 2(a)中的 *sel* 为地址输入, *sel* 中“0”表示输入为反变量,“1”为原变量;同时本文约定,若某一位变量不出现(即无关项),则用“-1”表示. 构成地址表达式的变量之间为逻辑“与”关系. 如图 2(b)中 M1 的地址表达式为“0000”,则该地址表达式为 4 个反变量构成的乘积项.

图 2(b)中各 Mux_2 门的地址逻辑用如下方法得到: 属于 Mux 树的任意一个 Mux_2 门的地址逻辑可以表示为与该 Mux_2 门“1”端相连的所有可能输入对应的地址逻辑表达式的“或”,因此,最底层 Mux_2 门的地址逻辑表达式等于选中该 Mux_2 门“1”端输入的地址逻辑表达式. 以 M5 为例,若从 M5 的“1”端分析,与 M5 的“1”输入端相连的是 M1,因此, M5 可能的输出就是 M1 的两个数据输入之一. 它们的地址逻辑表达式分别为“0000”和“0010”. 因此, M5 的地址逻辑表达式为“0000||0010”.

另外,在图 2(b)中, M1 的两个数据输入均为 $d[0]$,因此,不管与 M1 相连的地址如何变化,其输出恒为 $d[0]$,所以 M1 可以省略,同时将 M5 与 M1 相连的输入改成 $d[0]$ 即可,从而实现电路的简化.

2 树状 Mux 的逻辑电路创建

将 case 语句转化为树状 Mux 前,需将 case 语句转化为两个矩阵,分别为地址矩阵和数据矩阵.

因多输出电路可以等效为多个单输出的组合, 而单输出电路可以用一个 Mux 树表示, 因此本文以单输出 case 语句为例来说明 Mux 树形电路的创建与优化.

与图 2(a) case 语句对应的地址矩阵和数据矩阵如图 3 所示. 地址矩阵中的“0”和“1”分别表示 case 语句中地址信号 *sel* 的某个位取值 0 或 1, “-1”表示该位取值任意. 数据矩阵中分别用 2,4,...,(2*n*+2)表示原变量输入 *d*[0],*d*[1],...,*d*[*n*], 用 3,5,...,(2*n*+3)分别表示 *d*[0],*d*[1],...,*d*[*n*]的反变量输入.

[0,0,0,0],	[2],
[0,0,1,0],	[2],
[0,1,0,0],	[4],
[0,1,1,0],	[2],
[1,0,0,1],	[4],
[1,0,1,1],	[6],
[1,1,1,1],	[8],
[-1,-1,-1,-1]	[4]
地址矩阵	数据矩阵

图 3 case 语句与地址矩阵和数据矩阵对应关系

由于每个 Mux 树只有 1 位输出, 因此要表示 *n* 位输出的逻辑函数就需要建立 *n* 个 Mux 树.

算法 1 给出了利用数据矩阵和地址矩阵将 case 语句转化为对应的树形 Mux 逻辑电路的过程.

算法 1

Create_MuxTree(case_list[n][m],sel_list[m][p])

Input: case_list[n][m], sel_list[m][p]

Output: *M_T*[*n*];

for(*i*=0;*i*<*n*,*i*++1) {

Num_in=eva(case_list[*i*],sel_list); //step1

M_T[*i*]=gen_T(case_list[*i*],*Num_in*); //step2

T_addr(*M_T*[*i*], sel_list); //step3

 cmpat(*M_T*[*i*]); } //step4

算法 1 中 *case_list* 是二维数据矩阵, *n* 代表输出的位宽, *m* 表示 case 语句中列出的地址表达式的数量, 亦即 case 包含的分支数量; 输入 *sel_list* 是二维地址矩阵, *m* 含义同上, *p* 表示某个地址的第 *p* 位.

生成的 Mux 树用 *M_T* 表示, 其中 *M_T*[*i*]存储着第 *i* 个 Mux 树的根节点地址.

step1 中, 子程序 *eva*(case_list[*i*],sel_list)用来检测第 *i* 个输出对应的输入是否存在任意取值情况,

如果有, 就删除 case_list[*i*], sel_list 为任意取值所在的行; 并计算出第 *i* 个 Mux 树实际的数据输入个数 *Num_in*.

step2 中用二叉树来建立树形 Mux. 其中二叉树的每个节点可存储对应 Mux_2 门的地址表达式和 2 个数据输入端数据. 子程序 *gen_T*(case_list[*i*], *Num_in*)用于生成 Mux 树, 过程如下: 由根节点自顶向下生成满二叉树, 直到最底层的二叉树节点的 2 个输入数据储存单元可以容纳全部的 case_list[*i*]中数据为止. 生成二叉树后, 在最底层的二叉树节点上的输入数据储存单元中自左向右依次存入 case_list[*i*]中的数据. 若最底层的二叉树节点上的数据输入存储单元数超过 case_list[*i*]中存储的数据, 则用 case_list[*i*]中最后一位的数据填满剩余的数据输入存储单元.

step3 中子程序 *T_addr*(*M_T*[*i*], sel_list)用来实现将各 Mux_2 的地址表达式填写到 Mux 树对应的节点中. 方法如下: 找到 *M_T*[*i*]最底层二叉树节点, 将 sel_list 中与某节点的两输入端数据对应的地址填入到该节点的地址寄存单元中, 该存储单元被称为 *sel_mat*. *sel_mat* 包含两部分, 即 *sel_mat1* 和 *sel_mat0*, 分别用于存储构成 Mux_2 门中“1”输入端和“0”输入端的对应地址表达式的乘积项.

step4 中子程序 *cmpat*(*M_T*[*i*])用来删除具有相同两输入数据的节点, 以实现对 *M_T*[*i*]压缩.

以图 2(a)中的 case 为例, 对应的地址和数据矩阵如图 3 所示, 其数据矩阵一共有 8 个数据输入, 即 *Num_in*=8. 执行子程序 *gen_T* 时会生成一个满二叉树, 由于输入的数据为 8 个, 因此底层为 4 个节点时, 已满足将 8 个输入存储到最底层二叉树的要求, 因此二叉树生成完毕; 然后自左向右依次将 8 个数据输入填到 4 个节点的“1”输入端和“0”输入端.

在执行子程序 *T_addr* 时, 先填入底层 Mux_2 门的地址表达式, 这些表达式等于这些门的输出为“1”端数据时的地址表达式, 其地址表达式可通过查找地址矩阵得到. 与根节点 M7 的“1”相连的数据输入来自 M1 和 M2 的 4 个输入, 它们的地址表达式分别为“0000”“0010”“0100”和“0110”. 因此, M7 的地址逻辑表达式为上述 4 个乘积项的“或”.

子程序 *cmpat* 用来检查 Mux 树中是否存在输入相同的节点, 如有, 则删除该节点. 在图 2(b)中, M1 满足此条件, 可删除. 图 4 为删除 M1 后的 Mux 树结构.

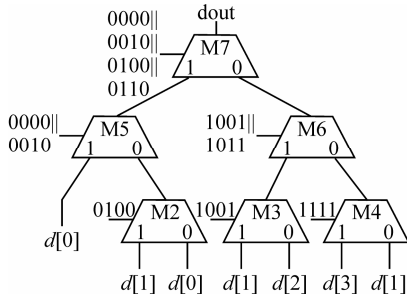


图 4 执行子程序 *cmpat* 后的逻辑电路

3 树形 Mux 的逻辑电路优化

3.1 case 语句的合并

通常情况下, case 语句中存在一定数量的相同赋值操作, 即不同地址输入下, 输出是相同的. 但在创建 Mux 树时依然会按照不同地址创建不同的 Mux 门. 考虑到 Mux 门的所有数据输入相同时, 该 Mux 门是可以删除的, 因此, 若能将相同的赋值操作移动到同一个 Mux 门之下, 就可以将对应的 Mux 门删除, 从而达到电路简化的目的.

图 5 是图 3 地址矩阵和数据矩阵合并简化后的结果, 其中左侧为合并后的地址矩阵, 右侧是数据矩阵. 图 5 中, 对应相同数据的地址被合并在一起, 使得数据矩阵只包含 4 个元素, 因此只需 3 个 Mux_2 门就可以了.

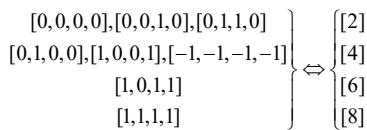


图 5 化简后地址矩阵和数据矩阵的对应关系

图 6 是化简后的 case 语句建立的 Mux 树结构. 化简之后 Mux 树中 Mux 门的数量和 Mux 树的层级都得到了减少.

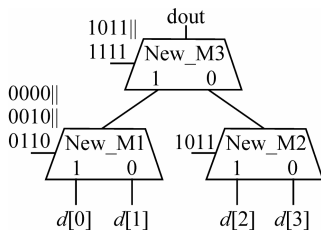


图 6 化简后的 Mux 树结构

算法 2 为 case 语句的合并算法伪代码. 其中, 输入 *case_list[n][m]* 表示 *n* 个输出的数据矩阵, 输入 *sel_list[m][p]* 是地址矩阵. 输出 *hash_list[n][data]* 用于存储具有 *n* 个输出的数据矩阵和地址矩阵合并后的结果.

算法 2

Case_hash(case_list[n][m], sel_list[m][p])

Input: *case_list[n][m]*, *sel_list[m][p]*;

Output: *hash_list[n][data]*;

for(*i*=0; *i*<*n*; *i*++) {

for(*j*=0; *j*<*m*; *j*++)

Merg_case_list(case_list[i][j], sel_list[j], hash_list[i][data]); }

子程序 *Merg_case_list(case_list[i][j], sel_list[j], hash_list[i][data])* 实现将第 *i* 个输出对应的 case 语句进行合并. 方法如下: 检查 *case_list[i]* 中所有输出取值情况, 然后按取值情况进行分类, 将相同输出对应的地址放在一起, 实现 case 语句的合并.

3.2 地址逻辑表达式的简化

从图 1 可知, 树形 Mux 电路包含 Mux 树和地址再编码电路, 因此, 树形 Mux 电路的优化除了要简化 Mux 树外, 还要简化地址再编码电路.

以图 6 中的 M2 为例, 由图 2(a) 的 case 语句可得, 当地址信号 *sel*==4'b1011 时, M2 的输出为 *d*[2], 当地址信号 *sel*==4'b1111 时, M2 的输出为 *d*[3]. 用 assign 语句可以表示为

assign M1_out = (sel == 4'b1011) ? d[2] : d[3]; (1)

或者

assign M1_out = (sel == 4'b1111) ? d[3] : d[2]. (2)

式(1)和(2)中都用到了 *sel* 的 4 位地址进行判断, 但分析上面 2 个语句的判断条件不难发现, 只要判断 *sel*[2] 就可以区分出选择输出是 *d*[2] 还是 *d*[3]. 因此可以将式(1)和式(2)改为

assign M1_out = (sel[2] == 0) ? d[2] : d[3]; (3)

assign M1_out = (sel[2] == 1) ? d[3] : d[2]. (4)

相比于式(1)和(2)用“与”门实现地址逻辑, 式(3)和(4)直接用 *sel*[2] 作为地址逻辑, 没有用到任何逻辑门.

虽然式(3)和(4)的判断只用到了 4 位地址中的 1 位, 并没有考虑另外 3 位的变化对数据选择的影响, 但与 M1 输出相连的 M3 的地址逻辑已表明有

且只有另外 3 位地址输入均为“1”时, 输出才能是 $d[2]$ 或 $d[3]$, 因此 M1 地址逻辑的简化不会影响电路的逻辑功能.

对于 M_T 中任何一个 Mux_2 地址表达式中的乘积项阵列 sel_mat1 和 sel_mat0 都执行算法 3 的简化操作.

算法 3

Reduce_sel(M_T)

Input: M_T ;

Output: *simplified M_T*;

{ for any Mux_2 in M_T , named M_i ;

$sel_mat1 \in M_i$ $sel_mat0 \in M_i$;

$col_set1 = same_col(sel_mat1)$;

$col_set0 = same_col(sel_mat0)$; //step1

if(($col_set1 \neq \emptyset$ && $col_set0 \neq \emptyset$) &&

$col_set1 \cap col_set0 \neq \emptyset$) {

$Y = find_dif(sel_mat1, sel_mat0, col_set1,$

$col_set0)$; //step2 }

if($Y \neq -1$) $sel_mat_Rdu(Y, sel_mat1, sel_mat0)$;

//step3

$exchang_in(M_i, sel_mat1, sel_mat0)$; //step4

step1 中子程序 *same_col* 分别检查 sel_mat1 和 sel_mat0 有无存在取值一样的列, 如果有, 则将列的编号分别存入集合 col_set1 和 col_set0 中. 如果 col_set1 和 col_set0 均非空, 且 col_set1 和 col_set0 包含公共的列的编号, 则在 step2 时检查属于 sel_mat1 和 sel_mat0 且编号属于 $col_set1 \cap col_set0$ 的列存储数值是否一致, 如果不一致, 则返回对应的列编号, 否则返回-1.

在 step3 中, 若 Y 不为-1, 则将原来的 sel_mat1 和 sel_mat0 分别用 $sel_mat1[Y]$ 和 $sel_mat0[Y]$ 代替, 即用一位变量来代替一个地址表达式, 进而实现地址表达式的简化. 若 $Y = -1$, 表示 sel_mat1 和 sel_mat0 无法简化, 保持不变.

在 step4 中, 检查 sel_mat1 和 sel_mat0 阵列的行数, 然后用数量少的地址存储阵列控制 Mux_2 的“1”输入端, 相应地, 与地址对应的输入端也要交换, 以保证逻辑功能正确.

以图 2(b) 的 M5 为例, 其 sel_mat1 存储的数据为 0000 和 0010, sel_mat0 存储的数据为 0100 和 0110. 假设最右边为第 0 列, 经过 step1 后得到

$col_set1 = \{0, 2, 3\}$, $col_set2 = \{0, 2, 3\}$, 即在 sel_mat1 和 sel_mat0 中, 第 0、2、3 列数据相同. 经 step2 后, 可以得到 sel_mat1 和 sel_mat0 中第 2 列数据不一样, 因此返回值 $Y = 2$. 经 step3 后, sel_mat1 和 sel_mat0 中分别简化为 $sel_mat1[2]$ 和 $sel_mat0[2]$. 由于 sel_mat1 和 sel_mat0 行数一样, 因此不执行 1 和 0 的输入及地址存储的交换.

算法 4 为 case 语句对应的树形 Mux 逻辑电路的优化算法伪代码. 其中待优化的电路用 Verilog 语言的 case 语句描述, 用 F_case 表示, 优化后的逻辑电路功能用 Verilog 的 assign 语句描述, 用 F_assign 表示.

算法 4

Opt_case(F_case, F_assign);

Input F_case ;

Output F_assign ;

{ *Case2Array(F_case, case_list, sel_list)*;

Case_hash(case_list, sel_list);

do *Reduce_sel(sel_mat1, sel_mat0)* for any
 Mux_2 in M_T ;

Create_Muxtree(M_T);

$F_assign = Gen_asg(M_T)$;}

算法 4 中, 子程序 *Case2Array* 用于将 case 语句转化为数据矩阵和地址矩阵; *Case_hash* 用于实现数据和地址矩阵的压缩, 使得与相同输出对应的地址表达式都放在一起, 从而方便逻辑简化; *Reduce_sel* 用于实现地址逻辑的简化. 在简化后的数据和地址矩阵基础上用 *Create_Muxtree* 实现树形 Mux 逻辑电路的生成, 并用 M_T 表示. 然后用 *Gen_asg* 将 M_T 转化为用 assign 语句构成的 Verilog 文件 F_assign . 将 F_assign 导入到第三方逻辑综合工具即可得到优化后逻辑电路的性能参数. 之所以要将 case 语句转化为 assign 语句, 是因为电路逻辑功能的 case 描述属于高层级的电路行为描述, 其优化结果受第三方综合工具影响很大. 用 assign 语句来代替 case 语句可以将 case 语句的行为化描述转化为 assign 语句的准网表描述, 进而减少第三方综合工具对优化后结果的影响.

子程序 *Gen_asg* 用于实现 M_T 向 assign 语句的转化, 方法如下: 由于每个 Mux_2 门对应一个 assign 语句, 对于 Mux 树中第 i 个 Mux_2 门, 其对

应的 assign 语句描述为

$$\text{assign } out_i = (adrs_i == 1) ? on_i : off_i, \quad (5)$$

其中 out_i 、 $adrs_i$ 、 on_i 和 off_i 分别表示第 i 个 Mux₂ 门的输出、地址逻辑表达式、“1”端数据输入和“0”端数据输入。

以图 6 的 M3 为例, 其对应 assign 描述为

$$\text{assign } M3_out = ((sel == 4'b1011) \parallel (sel == 4'b1111)) ? M1_out : M2_out. \quad (6)$$

由式(6)可知, M3 的 2 个数据输入与 M1 和 M2 的输出相连. 因此将 Mux 树中每个 Mux₂ 门转化为 assign 语句后, 自然得到构成整个 Mux 树的每个 Mux₂ 门之间的连接关系, 即由 Mux₂ 门构成的网表.

4 实验结果与分析

本文的测试电路以及映射库来自于由多家 EDA 公司举办的 2022 年 EDA 精英挑战赛^[12]. 由于提供的映射库中只包含 3 种逻辑门, 即二输入与门、Mux₂ 和反相器, 因此最终得到的电路实际上是由上述 3 种逻辑门构成的.

实验中用到的第三方 EDA 工具有开源软件 yosys^[13-14]和学术上常用的开源 EDA 工具 abc^[15]. 其中 yosys 是用于将用 Verilog 描述的电路功能文件转化为 abc 可以接受的文件格式, 然后由 abc 执行电路的综合与优化, 并完成指定映射库下的映射, 进而得到电路的性能指标值. 在用 abc 实现逻辑优化时, 本文采取的是面积优先优化命令, 因此优化后部分电路可能出现延迟小幅度上升的情况.

表 1 为本文方法与 abc 工具在各测试电路上的面积和延迟优化结果. 从表 1 可得, 在绝大多数测试电路上, 本文方法的面积或延迟优化效果要好于 abc. 有些测试电路的优化效果尤为突出, 如 large_same 面积相对优化达到了 96%. 其原因是描述该电路的 case 语句中的地址信号位宽为 n , 而 case 语句数量正好为 2^n , 即 full case 描述. full case 对应的 Mux 树, 其两两相邻的子节点间的地址均可以进行布尔表达式化简, 从而实现面积的优化. 而对于那些 default 中包含许多种地址信号, 或者说 case 中列出的语句数量与 2^n 差距越大, 本算法效果越欠佳. 测试电路 large_mix 和 large_mix2 等就属于这种情况.

表 1 abc 优化并映射后的面积与延迟

测试电路	I/Os	优化后面积		面积	优化后	延迟	相对
		abc	本文	相对	abc	本文	
				率/%	率/%	率/%	率/%
multi_bits	35/4	85.5	55.5	35	6.6	3.4	48
normal	11/1	27	14	48	5.4	3	44
simple_case	9/3	33.5	23	31	5.6	4	29
large_mix4	20 018/16	12 829.5	11 342	12	19.2	19.4	-1
large_bits	53/10	352.5	302.5	14	12.2	10.4	15
samechoosen	11/1	13	6	54	4.6	2	57
large_casez2	63/10	475.5	430.5	9	13.4	13	3
add	5/4	16	17	-6	5	6	-20
large_mix	20 024/16	130 458	147 533.5	-13	19.8	22.8	-15
constant	11/1	22.5	12.5	44	5.8	3.4	41
large_casez3	63/12	570	501	12	13.2	13	2
not	11/1	29	16	45	5.6	3.2	43
casez	16/1	40	19.5	51	8	5	38
large_casez	52/10	266	240	10	11	8.4	24
mix	11/1	16.5	8.5	48	6	3.2	47
large_mix3	20 024/16	23 349.5	25 157	-8	18.8	18.8	0
large_casez4	64/12	1 034.5	905.5	12	14.4	14.4	0
large_mix2	20 024/16	38 722	42 658.5	-10	19.8	20.6	-4
large_same	2 060/8	92 497.5	4 080	96	18.6	8	57
平均值		-		26	-		21

另一类电路, 如 add, 其 case 语句中的赋值全为常数, 对于 Mux₂ 门而言, 其逻辑表达式为 $F = SA + \bar{S}B$. 当数据输入 A 、 B 为常数 0 或 1 时, 全部 Mux₂ 门都可化简为 and 或 or 门, 导致用本文提出的树形 Mux 电路来描述 add 功能效果不理想.

总体而言, 本文算法效果好于 abc, 表 1 显示面积和延迟相对优化率平均值分别为 26% 和 21%.

5 总结

本文提出了一种基于树形 Mux 结构的逻辑电路优化算法, 主要用于 FPGA 和 Verilog 中逻辑功能用 case 语句描写的电路的优化. 提出的算法主要包括 case 语句的合并和地址逻辑的化简. 该算法通过缩减 Mux 树中门的个数和层级, 以及减少地址再编码电路的复杂性来达到电路面积和延迟减小的目的. 从实验结果来看, 本文算法与开源工具 abc 相比, 对读取的测试电路进行相同的优化后,

延迟和面积优化效果分别提升了 26% 和 21%, 其中对 full case 描述的电路的优化效果尤为突出。

参考文献:

- [1] Rai S, Nath P, Rupani A, et al. A survey of FPGA logic cell designs in the light of emerging technologies[J]. IEEE Access, 2021, 9:91564-91574.
- [2] Singh N K, Sharma P K. 2T 2:1 MUX based 1 bit full adder design[C]//2014 International Conference on Communication and Signal Processing, IEEE, 2014: 1491-1493.
- [3] Murugeswari S, Mohideen S K. Design of area efficient and low power multipliers using multiplexer based full adder[C]//Second International Conference on Current Trends In Engineering and Technology-ICCTET, IEEE, 2014:388-392.
- [4] Hatti K, Paramasivam C. The MUX-based PUF architecture for hardware security[C]//2021 International Conference on Circuits, Controls and Communications (CCUBE), Bangalore, India, 2021:1-7.
- [5] 杨海钢, 孙嘉斌, 王慰. FPGA 器件设计技术发展综述[J]. 电子与信息学报, 2010, 32(3):714-727.
- [6] Shen Y, Qian J, Shi K, et al. Two-level MUX design and exploration in FPGA routing architecture[C]//2021 31st International Conference on Field-Programmable Logic and Applications (FPL), IEEE, 2021:234-241.
- [7] Metzgen P, Nancekievill D. Multiplexer restructuring for FPGA implementation cost reduction[C]//Proceedings of 42nd Design Automation Conference, IEEE, 2005:421-426.
- [8] Mitra S, Avya L J, McCluskey E J. Efficient multiplexer synthesis techniques[J]. IEEE Design & Test of Computers, 2000, 17(4):90-97.
- [9] Basiri M M A, Noor M S. High speed multiplexer design using tree based decomposition algorithm[J]. Microelectronics Journal, 2016, 51:99-111.
- [10] Kohutka L, Pistek P. Faster synthesis of combinational logic based on multiplexer trees and binary decision diagrams[C]//2014 IEEE 12th International Conference on Emerging eLearning Technologies and Applications (ICETA), Stry Smokovec, Slovakia, 2015:239-244.
- [11] Yang P H, Chen J M, Lin K S. A high-performance 128-to-1 CMOS multiplexer tree[C]//2012 International Symposium on Intelligent Signal Processing and Communications Systems, Tamsui, Taiwan, China, 2013: 806-809.
- [12] 中国电子学会电子设计自动化专委会. 集成电路 EDA 设计精英挑战赛[EB/OL]. [2023-02-11]. <https://eda.icisc.cn/file/cacheFile/43ea22a2ef174074ae8dbea183b31763.pdf>.
- [13] Wolf C, Glaser J, Kepler J. Yosys - A free verilog synthesis suite[C]//Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip), 2013:97.
- [14] Shah D, Hung E, Wolf C, et al. Yosys nextpnr: An open source framework from verilog to bitstream for commercial FPGAs[C]//2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), IEEE, 2019:1-4.
- [15] Mishchenkoet A. ABC: A system for sequential synthesis and verification. (Release 20200217)[DB/OL]. [2023-01-08]. <https://github.com/berkeley-abc/abc>.

Mux tree-based logic circuit optimization

YU Zongyuan, LIAO Chunliu, HU Zhang, WANG Lunyao*

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

Abstract: In order to optimize the area and delay of logic circuits described using case statements, an optimization method based on Mux tree is proposed. The method first converts case statements into a Mux tree, followed by reducing the number of Muxes and levels of Mux tree with merging case statements. It also simplifies the address logic to achieve a simpler address encoding circuit, hence optimizing the area and delay of the circuits after being mapped. The proposed algorithm is implemented in C++, and the optimization results are obtained using academic EDA tool named abc with the mapping library provided by domestic EDA companies. The experimental results show that compared with abc tool, the performance of area and delay are improved by 26% and 21%, respectively.

Key words: Mux tree; case statement synthesis; logic optimization; Verilog HDL

(责任编辑 韩 超)