

基于不相交锐积运算的逻辑函数错误率计算

应秋红, 王伦耀*, 储著飞, 夏银水

(宁波大学 信息科学与工程学院, 浙江 宁波 315211)

摘要: 提出了一种通过比较逻辑覆盖差异的函数错误率计算方法, 该方法主要包括逻辑覆盖不相交锐积运算和双锐积运算, 分别用于实现2个逻辑覆盖之间不相交操作和公共部分删除操作, 进而实现TB逻辑函数之间或RM逻辑函数之间覆盖差异的提取. 通过对所提取覆盖包含的最小项数量统计, 实现函数错误率的计算. 所提出的方法用C语言实现, 并用MCNC测试电路进行测试. 实验结果表明, 该算法可以实现TB函数和RM函数的错误率计算, 且具有运算速度快, 适合处理大逻辑函数的特点.

关键词: 近似计算; 错误率计算; 逻辑优化

中图分类号: TN79+1

文献标志码: A

文章编号: 1001-5132 (2020) 02-0035-06

通常逻辑功能正确是电路设计首要条件, 为了保证电路输出正确, 会不惜牺牲电路的功耗、面积和速度等性能指标. 但是在实际应用中, 待处理数据的某些特性, 如容错特性, 在一定程度上的计算错误不影响实际的应用^[1]. 例如, 在多媒体处理中, 由于人类的感知能力有限, 偶尔发生丢弃特定的帧或小的图像质量损失通常很少影响用户的满意度. 另外, 随着半导体工艺尺寸进入到纳米级, 电路的供电电压更低, 集成密度更高, 电路输出对工艺参数的波动和故障更加敏感^[2]. 待处理数据自身具有的容错特性, 以及降低现有集成电路工艺的高成本, 使得人们重新思考逻辑功能“正确”的程度, 提出在设计中引入近似计算技术来实现电路优化, 提高电路良品率. 目前, 近似计算已经成为数字集成电路设计中一个新的、重要的策略. 近似计算技术已应用于从系统、架构、存储、逻辑综合到单元电路设计等不同电路层级的优化, 以及现场可编程逻辑门阵列(Field Programmable Gate Array, FPGA), 神经网络、通信、可逆逻辑的等诸多方面^[3-7].

在近似计算中, 近似度表示待优化函数与近似函数的偏差程度, 是一个关键指标. 基于近似计

算的电路面积、功耗、时延等方面的优化都要满足一定的近似度约束, 近似度通常用错误率(Error Rate, ER)、最大错误偏量(Worst Case Error, WCE)、平均绝对误差(Mean Absolute Error, MAE)、标准差(Standard Deviation, SD)等指标来衡量^[8-11]. 本文采用 ER 表示近似度.

目前, 基于近似计算的逻辑函数优化已经有一些研究. 文献[10]利用近似计算技术和启发式算法对布尔逻辑函数进行化简; 文献[12]通过试探性增加或删除乘积项来实现逻辑表达式简化. 这些研究都是对基于“与/或/非”运算的传统布尔(Traditional Boolean, TB)逻辑进行近似优化. 除TB逻辑外, 另一种常用的逻辑表达式是基于“与/异/或”的 Reed-Muller(RM)逻辑. 由于 RM 逻辑中存在 $1 \oplus 1 = 0$, 因此, 增加或删除偶数个乘积项不会给原函数引入错误. RM 逻辑的这种特性给错误率计算带来了挑战. 此外, 现有的错误率计算方法基本上是基于最小项展开的. 由于最小项的数量与输入变量数呈 2^n 增加, 当 n 比较大时导致最小项数量过于庞大, 以至于该方法无法用于大函数的错误率计算.

本文提出的逻辑函数错误率算法主要为基于

收稿日期: 2019-04-17.

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(61471211); 浙江省自然科学基金(LY19F040004); 宁波市自然科学基金(2019A610077).

第一作者: 应秋红(1994-), 女, 浙江台州人, 在读硕士研究生, 主要研究方向: 逻辑综合与优化. E-mail: mint_ying@foxmail.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

逻辑覆盖不相交锐积运算的错误计算方法. 该算法可实现对TB逻辑和RM逻辑函数的错误率计算, 同时运算速度对逻辑函数的输入变量个数不敏感, 可实现大函数的近似函数的错误率计算.

1 相关定义

对一个具有 n 个输入, m 个输出的逻辑函数 f , 其逻辑表达式可写成如下乘积项之和(SOP)形式:

$$f(x_1, x_2, \dots, x_n) = \sum_{i=0}^{z-1} p_i, \quad (1)$$

其中: p_i 为乘积项; z 为乘积项的个数.

乘积项 p_i 可表示为:

$$x_1, x_2, \dots, x_i, \dots, x_n | y_1, y_2, \dots, y_i, \dots, y_m, \quad (2)$$

其中: 竖线表示分隔符, 竖线左边表示输入变量, 右边表示输出变量. 输入变量可以取 $\{0, 1, -\}$, 1 表示原变量, 0 表示反变量, $-$ 表示因合并而消去的变量; 输出变量可以取 $\{0, 1, u\}$, u 表示该输出变量未被定义.

有时式(2)也称为一个立方体(Cube), 当输入变量 $x_i \in \{0, 1\}, 1 \leq i \leq n$, 此时式(2)称为最小项.

具有相同输入和输出的乘积项集合构成一个逻辑覆盖 C . 如果一个覆盖 C (或覆盖的集合) 包含的最小项和一个函数 f 包含的最小项一致, 那么 f 可以用这个覆盖或覆盖的集合来描述.

(1)乘积项的覆盖. 乘积项的覆盖就是该乘积项所包含的最小项的集合. 本文也将乘积项覆盖或乘积项集合的覆盖称为逻辑覆盖.

(2)乘积项的“并”运算. 乘积项覆盖的并运算用符号“ $\cup$ ”表示, 其运算规则是将构成多个乘积项包含的最小项合并在一个覆盖中.

(3)乘积项的“交”运算. 乘积项覆盖的“交”运算用符号“ $\cap$ ”表示. 假设 $X_1|Y_1$ 和 $X_2|Y_2$ 是 2 个乘积项, 则 $(X_1|Y_1) \cap (X_2|Y_2)$ 表示这 2 个乘积项公共顶点组成的乘积项. 如果 2 个乘积项不存在公共顶点, 即 $(X_1|Y_1) \cap (X_2|Y_2) = \emptyset$, 则说明这 2 个乘积项是不相交的.

(4)乘积项之间的不相交锐积运算. 2 个乘积项之间的锐积运算定义为:

$$(X_1|Y_1) \otimes (X_2|Y_2) = (X_1|Y_1) - (X_1|Y_1) \cap (X_2|Y_2), \quad (3)$$

其中: $\otimes$ 表示乘积项之间的不相交锐积运算, 即

在乘积项 $(X_1|Y_1)$ 包含的最小项中减去 $(X_1|Y_1)$ 和 $(X_2|Y_2)$ 的公共最小项后得到的乘积项集合.

假设 a, b 为单输出乘积项, 且 $a = (-1-), b = (1-0)$, 则

$$a \otimes b = a - a \cap b = (-1-) - (-1-) \cap (1-0) = \{(01-), (111)\}. \quad (4)$$

从式(4)可知, $a \otimes b$ 的结果为乘积项的集合, 且集合中的 2 个乘积项不相交.

2 个乘积项之间的不相交锐积运算的实现方法参见文献[13].

2 逻辑覆盖的不相交锐积及算法实现

假设 2 个逻辑覆盖 A 和 B 分别有 T 个和 S 个乘积项构成, 即

$$A = \{a_1, a_2, \dots, a_T\} = \bigcup_{i=1}^T a_i, B = \{b_1, b_2, \dots, b_S\} = \bigcup_{j=1}^S b_j,$$

则覆盖 A 与覆盖 B 之间的不相交锐积可表示为:

$$A \otimes B = (\bigcup_{i=1}^T a_i) \otimes (\bigcup_{j=1}^S b_j) = \bigcup_{i=1}^T (a_i \otimes (\bigcup_{j=1}^S b_j)) = \bigcup_{i=1}^T (a_i - a_i \cap (\bigcup_{j=1}^S b_j)) = A - A \cap B, \quad (5)$$

其中: 运算 $a_i \otimes (\bigcup_{j=1}^S b_j) = a_i - a_i \cap (\bigcup_{j=1}^S b_j)$ 在算法实现上可以通过乘积项 a_i 对覆盖 B 中的各个乘积项的连续不相交锐积实现, 即

$$a_i \otimes (\bigcup_{j=1}^S b_j) = (\dots((a_i \otimes b_1) \otimes b_2) \dots \otimes b_S). \quad (6)$$

现在以 2 个单输出覆盖 $A_1 = \{-01, 1--\}$, $B_1 = \{000-, 11--\}$ 不相交锐积为例, 先实现乘积项 (-01) 与 B_1 的不相交锐积, 再实现 $(1--)$ 与 B_1 的不相交锐积. 其中:

$$\begin{aligned} (-01) \otimes B_1 &= ((-01) \otimes (000-)) \otimes (11--)= \\ &= \{(1-01) \cup (0101)\} \otimes (11--)= \\ &= \{(1-01) \otimes (11--)\} \cup \{(0101) \otimes (11--)\}= \\ &= \{1001, 0101\}, \end{aligned}$$

同理可得 $(1--) \otimes B_1 = \{10-1\}$. 所以

$$A_1 \otimes B_1 = \{1001, 0101\} \cup \{10-1\} = \{1001, 0101, 10-1\}. \quad (7)$$

图 1 用卡诺图表示覆盖 A_1 与 B_1 不相交锐积过程, 其中图 1(a)的卡诺图对应覆盖 A_1 , 图 1(b)对应

覆盖 B_1 , 图 1(c) 为 $A_1 \otimes B_1$ 的结果. 在式(7)中由于乘积项 $\{1001\}$ 被 $\{10-1\}$ 包含, 所以在图 1(c) 中就没有单独圈出 $\{1001\}$. 从图 1 可知, 2 个逻辑覆盖 A_1 与 B_1 不相交锐积结果等于从覆盖 A_1 中除去与 B_1 相交的部分, 即 $A_1 \otimes B_1 = A_1 - A_1 \cap B_1$.

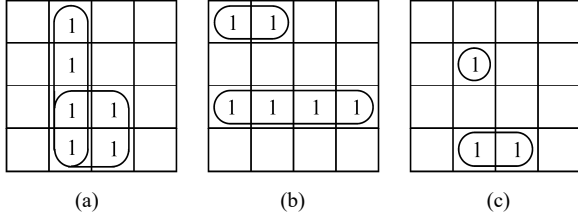


图 1 2 个覆盖的不相交锐积运算示例

算法 1 为逻辑覆盖的不相交锐积的算法实现, P_w 和 P_v 分别为逻辑覆盖 C_1 和 C_2 中的乘积项, 最终覆盖 C_1 , 即为 $C_1 \otimes C_2$ 的运算结果.

算法 1 Two_covers_shp_dsj (C_1, C_2)

Input: The cubes sets of C_1 and C_2 ;

Output: The cubes in $C_1 = C_1 \otimes C_2$;

For any cube P_w in C_1 , and any cube P_v in C_2 ,

while ($P_w \cap P_v \neq \emptyset$) {

$C_1 = (C_1 - \{P_w\}) \cup (P_w \otimes P_v)$;

3 基于不相交锐积的逻辑函数错误率计算原理

错误率的定义是导致错误的输入组合占总的输入组合的比例. 对于一个输入个数为 n 的逻辑函数 f , 如果 C_{err} 表示引起输出错误的乘积项集合, $M[C_{err}]$ 表示 C_{err} 中的顶点个数, 则逻辑函数的错误率 E_{ER} 可以表示为:

$$E_{ER} = (M[C_{err}] / 2^n) \times 100\%. \quad (8)$$

在利用近似计算技术实现逻辑函数优化时, 可在错误约束下引入一定的错误项, 从而实现逻辑表达式的简化. 逻辑函数乘积项的集合构成了 f 的逻辑覆盖, 用 C_f 表示. 逻辑函数 f 的近似函数定义为 f_a , 将 f_a 对应的乘积项集合定义为 C_a , 则 C_f 与 C_a 的差异部分就构成了错误乘积项集合 C_{err} . 因此 C_{err} 包含两部分: 一部分属于 C_f , 但不属于 C_a ; 另一部分属于 C_a , 但不属于 C_f . 所以 C_{err} 可表示为:

$$C_{err} = (C_f - C_f \cap C_a) \cup (C_a - C_a \cap C_f) = (C_f \otimes C_a) \cup (C_a \otimes C_f). \quad (9)$$

在 TB 逻辑中存在 $1+x=1$ 的运算关系, 而 RM 逻辑中存在 $1 \oplus 1=0$, 使得在 RM 函数中增加或删除

去偶数个相同的乘积项不会改变 RM 逻辑的正确性, 这导致 RM 逻辑在错误率计算方面不同于 TB 逻辑.

以图 2 卡诺图为例, 取值为 “1” 的表示原始的逻辑函数包含的最小项的组合.

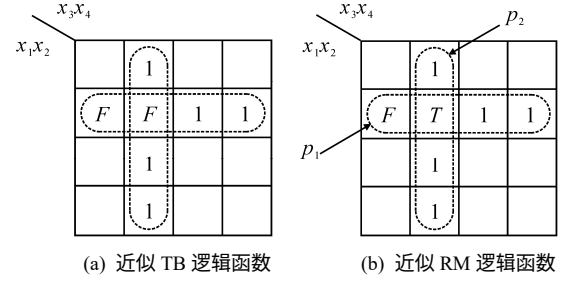


图 2 近似逻辑函数对应的卡诺图示例

图 2(a) 是采用 TB 逻辑表示近似函数, 得 $f_{TB} = \overline{x_1}x_2 + \overline{x_3}x_4$; 图 2(b) 采用 RM 逻辑表示近似函数, 得 $f_{RM} = \overline{x_1}x_2 \oplus \overline{x_3}x_4$. 虽然 f_{TB} 和 f_{RM} 包含的乘积项一样, 但两者引起错误输出的最小项不一样. 图 2(a) 中有 2 个, 分别标有 “F”; 图 2(b) 中只有 1 个最小项标有 “F”, 而标有 “T” 的最小项被覆盖了 2 次, 考虑到 $1 \oplus 1=0$, 因此标有 “T” 的最小项在 RM 逻辑中可以简化逻辑表达式, 而不会引入增加错误率. 所以如果直接利用式(8)计算图 2(b) 所示近似 RM 函数的错误率, 而不对近似 RM 函数包含的乘积项进行预处理, 就是将那些被偶数次包含的最小项也统计到错误率计算中, 从而导致错误率计算不准确. 因此, 在计算近似 RM 逻辑错误率时必须去除 RM 函数中被偶数次包含的最小项. 为此, 本文在乘积项不相交锐积基础上引入乘积项双不相交锐积操作, 实现 2 个相交乘积项之间相交部分的去除. 对于 2 个乘积项 p 和 q , 其双不相交锐积定义为^[14]:

$$p \diamond q = (p - p \cap q) \cup (q - q \cap p) = (p \otimes q) \cup (q \otimes p). \quad (10)$$

以图 2(b) 为例, 共有 2 个乘积项, $p_1 = (01--)$, $p_2 = (--01)$, 则

$$p_1 \diamond p_2 = \{(01--) \otimes (--01)\} \cup \{(--01) \otimes (01--)\} = \{011-, 0100\} \cup \{1-01, 0001\} = \{011-, 0100, 1-01, 0001\}. \quad (11)$$

对比不难发现, 式(11)中得到的 4 个乘积项对应最小项实现了对图 2(b) 中去掉标有 “T” 的最小项的去除. 算法 2 为实现 RM 逻辑函数中去除被偶

数次包含的最小项.

算法 2 Double_shp_dsj_RM(C_{RM})
Input: The products of RM function;
Output: C_{RM} after double sharp disjointed;
For any two products and q in C_{RM} ;
while ($p \cap q \neq \emptyset$) {
 $C_{RM} = (C_{RM} - \{p, q\}) \cup (p \oplus q)$ };

4 基于不相交锐积运算的错误率计算算法

错误率计算的关键是找出原函数 f 与近似函数 f_a 之间的差异. 这个差异可以通过比较 f 与 f_a 之间的逻辑覆盖差异来求得, 也可以通过比较 f 与 f_a 包含的最小项的不同来求得. 考虑到逻辑函数包含的最小项数量与输入变量数成 2 的指数次方关系, 因此随着输入变量数增加, 最小项的数量将急剧增加, 导致基于最小项比较的方法的运算速度受到严重影响, 所以错误率处理方法往往算法处理速度慢, 且不能处理大电路. 本文提出的算法是基于乘积项的不相交锐积展开的, 逻辑函数中的乘积项的个数远小于最小项展开的个数, 因此本文方法更适合处理大电路.

算法 3 为基于不相交运算的逻辑函数错误率算法实现, 其中 C_f 为原函数的乘积项集合, C_{ap} 为近似函数的乘积项集合, C_{err} 为引起错误输出的乘积项集合.

算法 3 Get_error_cover(C_f, C_{ap}, C_{err})
Input: C_f, C_{ap} ;
Output: C_{err} ;
Copy C_f to C_{tmp} ;
If C_{ap} is in RM form, Then
Double_shp_dsj_RM(C_{ap}); //step 1
Two_covers_shp_dsj(C_f, C_{ap}); //step 2
Two_covers_shp_dsj(C_{ap}, C_{tmp}); //step 3
 $C_{err} = C_f \cup C_{ap}$; //step 4
For any two products p, q in C_{err} ,
while ($p \cap q \neq \emptyset$) {
if ($p \subseteq q$), $C_{err} = (C_{err} - p)$;
else $C_{err} = (C_{err} - p) \cup (p \otimes q)$; //step 5

在 step 1 中先判断近似逻辑函数是否采用 RM 形式, 如果为 RM 形式, 则利用双锐积运算实现被偶数次覆盖部分的去除; step 2 到 step 4 是利用式

(9)实现 C_f 和 C_{ap} 覆盖差异比较, 得到错误覆盖; 在 step 5 中, 由于多个覆盖锐积后可能会存在相交的乘积项, 因此本文利用乘积项之间的不相交锐积将 C_{err} 中的乘积项转化为不相交乘积项, 方便 C_{err} 包含的最小项数量的计算.

假设错误覆盖 C_{err} 中有 w 个不相交乘积项, 则 C_{err} 包含的最小项的计算方法为:

$$N = \left(\sum_{i=0}^{w-1} 2^v \mid_{P_i \in C_{err}} \right), \quad (12)$$

式中: $2^v \mid_{P_i \in C_{err}}$ 表示属于集合 C_{err} 的不相交乘积项 P_i 有 v 个变量没有出现.

以图 2 所示函数为例, 图 2 中的卡诺图取值为“1”的覆盖对应的逻辑函数可表示为: $f = x_1 \bar{x}_3 x_4 + \bar{x}_1 x_2 x_3 + \bar{x}_1 x_2 x_3 x_4$, 因此 $C_f = \{1-01, 011-, 0001\}$. 如果用 TB 逻辑来进行近似逻辑优化, 得到的函数可由图 2(a)所示的卡诺图表示, 则 $f_{ap, TB} = \bar{x}_3 x_4 + \bar{x}_1 x_2$, 因此 $C_{ap} = C_{ap, TB} = \{-01, 01--\}$. 由于 C_{ap} 是 TB 逻辑的乘积项集合, 因此不执行算法 3 中 step 1. 执行 step 2 时, 由于 $C_f \subset C_{ap}$, 因此执行完 step 2 时 $C_f = \emptyset$. 执行 step 3 时, $C_{ap} = \{-01, 01--\}$, $C_{tmp} = \{1-01, 011-, 0001\}$, 因此 $C_{ap} \otimes C_{tmp} = \{0101, 010-\}$.

由于乘积项 (0101) 与 (010-) 相交, 需要将其转化为不相交乘积项. 令 $p = (0101)$, $q = (010-)$, 由于 $p \subseteq q$, 所以 $C_{err} = (C_{err} - p) = (010-)$.

利用式(12)得到 C_{err} 对应的最小项为 $2^1=2$. 考虑到图 2(a)为 4 变量逻辑函数, 因此图 2(a)对应的错误率为 $2/(2^4)=1/8$.

同理, 如近似逻辑函数为图 2(b)所示的 RM 逻辑, 则 $f_{ap, RM} = \bar{x}_3 x_4 \oplus \bar{x}_1 x_2$, $C_{ap} = \{-01, 01--\}$. 执行算法 3 中 step 1 后得 $C_{ap} = \{1-01, 0001, 011-, 0100\}$.

执行 step 2 后, 得到 $C_f = \emptyset$, 原因是 $C_f \subset C_{ap}$. 执行 step 3 后, 得到 $C_{ap} \otimes C_{tmp} = \{0100\}$, 因此 $C_{err} = (0100)$. 考虑到 C_{err} 只有一个乘积项, 所以可以直接计算错误率为 $1/(2^4)=1/16$. 对照图 2(a)和图 2(b), 上述 2 个结论显然是正确的.

5 实验结果分析

为了测试所提出的逻辑函数错误率算法的性能, 将该算法分别应用于基于近似计算的 TB 逻辑电路和 RM 逻辑电路面积优化中. 测试算法用 C 语言实现, 并在操作系统为 Windows 10, CPU 时钟频

率为 3.0 GHz, 内存为 8 GB, i5 处理器的 PC 机上实现. 测试电路为 MCNC 标准电路, 文件格式为 PLA.

对于输入变量数一定的 TB 逻辑函数, 构成该函数的某个乘积项的字母数越多, 该乘积项包含的最小项就越少, 因此, 删除该乘积项所引起的错误率越少, 面积的优化效果越明显. 本文通过如下方法得到近似函数: (1)先用 espresso 综合工具实现对原始 TB 逻辑函数优化; (2)将优化后得到的乘积项按照包含的字母数的多少排序; (3)删除包含字母数多的前 10%乘积项, 并用所提出的错误率计算方法得到近似 TB 逻辑的错误率. 实验结果见表 1. 在 RM 逻辑测试方面, 先将原始函数转换为零极性下的 FPRM 函数, 再随机删除其中 10%的乘积项, 然后用所提出的错误率计算方法计算近似 RM 逻辑的错误率, 实验结果见表 2.

表 1 基于不相交锐积运算的 TB 函数错误率计算结果

电路名	<i>i/o</i>	N_{Lts_esp}	N_{Lts_app}	$D_{imp}/\%$	$E_{ER}/\%$	t/ms
f51m	8/8	319	277	13.17	0.73	117
amd	14/24	444	383	13.74	0.39	60
ldd	9/19	72	57	20.83	0.37	9
ibm	48/17	882	736	16.55	0.13	190
ti	47/72	1 835	1 588	13.46	0.04	211
in1	16/17	972	846	12.96	<0.01	90
sao2	10/4	421	371	11.88	0.12	50
x6dn	39/5	641	556	13.26	0.10	70
cht	47/36	253	221	12.65	1.38	90
ryy6	16/1	624	552	11.54	0.34	145

表 2 基于不相交锐积运算的 RM 函数错误率计算结果

电路名	<i>i/o</i>	N_{Lts_FPRM}	N_{Lts_app}	$D_{imp}/\%$	$E_{ER}/\%$	t/ms
f51m	8/8	264	220	16.67	4.68	137
amd	14/24	210	177	15.71	0.10	29
ldd	9/19	6 227	2 321	62.73	6.22	118
ibm	48/17	20 145	15 856	21.29	11.10	31 519
ti	47/72	91 528	68 626	25.02	0.19	1 769
in1	16/17	49 041	35 665	27.28	0.51	351
sao2	10/4	13 013	5 005	61.54	5.39	2 410
x6dn	39/5	108 085	71 006	34.31	2.16	1 350
cht	47/36	594	518	12.79	3.21	129
ryy6	16/1	624	471	24.52	2.09	387

由于在基于近似计算的电路面积优化中, 电路面积一般用逻辑表达式包含的字母数表示^[15].

因此, 表 1 和表 2 也统计了逻辑函数近似优化前后的字母数. 表中 *i/o* 表示测试电路的输入和输出变量数, N_{Lts_esp} 表示 espresso 综合工具优化后函数电路包含的字母数, N_{Lts_FPRM} 表示零极性下 FPRM 函数包含的字母数, N_{Lts_app} 表示近似函数电路包含的字母数. E_{ER} 为错误率, 用式(8)求得. D_{imp} 表示通过近似计算优化后逻辑电路的面积优化程度, 用式(13)求得. 式(13)中 N_{Lts_org} 表示没有引入错误时逻辑函数包含的字母数, 其在表 1 中对应的变量为 N_{Lts_esp} , 在表 2 中对应的变量为 N_{Lts_FPRM} , 算法运行单位为 ms.

$$D_{imp} = \frac{N_{Lts_org} - N_{Lts_app}}{N_{Lts_esp}}. \quad (13)$$

从表 1 和表 2 不难发现, 所提出的错误率方法可以计算近似 TB 逻辑函数和近似 RM 逻辑函数的错误率. 算法运行时间与待处理的逻辑函数的变量个数无明显的关联, 算法可以实现输入变量数大于 40 的逻辑函数. 算法的耗时与不相交锐积运算执行的次数有关, 当函数包含的乘积项数较多时, 耗时较长. 在计算 RM 逻辑函数的错误率时, 需要用双锐积运算实现被偶数次覆盖部分的去除, 再用不相交锐积实现错误率计算. 因此, 同一个电路在转化成 RM 后, 计算错误率的时间更长.

6 结语

本文提出了一种基于逻辑覆盖不相交运算的逻辑函数错误率的计算方法, 分别实现 TB 逻辑函数和 RM 逻辑函数的错误率计算. 该方法的核心是利用逻辑覆盖不相交算法实现近似函数和原始函数逻辑覆盖的求解, 进而实现函数错误率的计算. 由于在 RM 逻辑中删除或增加偶数次相同逻辑覆盖不改变逻辑准确性, 因此在 RM 函数错误率计算中提出了利用双锐积运算实现 RM 逻辑覆盖中偶数次覆盖消除方法, 进而实现 RM 逻辑错误率的计算. 从实验结果看, 提出的算法可以处理输入变量数超过 40 的大函数, 并且算法的运算速度与待处理的函数的输入变量数没有明显的关联.

参考文献:

- [1] Venkataramani S, Chakradhar S T, Roy K, et al. Computing approximately, and efficiently[EB/OL]. [2019-01-20]. <https://ieeexplore.ieee.org/document/7092486>.

- [2] Han J. Introduction to approximate computing[EB/OL]. [2019-01-20]. <https://ieeexplore.ieee.org/document/7477305/metrics#metrics>.
- [3] Jung M, Mathew D M, Weis C, et al. Approximate computing with partially unreliable dynamic random access memory-approximate DRAM[EB/OL]. [2019-01-20]. <https://dl.acm.org/citation.cfm?id=2905002>.
- [4] Wu Y, Shen C, Jia Y, et al. Approximate logic synthesis for FPGA by wire removal and local function change[EB/OL]. [2019-01-20]. <https://ieeexplore.ieee.org/document/7858314>.
- [5] Sen S, Raghunathan A. Approximate computing for long short term memory (LSTM) neural networks[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2018, 37(11):2266-2276.
- [6] Nguyen D T, Kim H, Lee H. An approximate memory architecture for a reduction of refresh power consumption in deep learning applications[EB/OL]. [2019-01-20]. https://www.researchgate.net/publication/324956616_An_Approximate_Memory_Architecture_for_a_Reduction_of_Refresh_Power_Consumption_in_Deep_Learning_Applications.
- [7] Muhammad S, Rehan H, Semeen R, et al. Cross-layer approximate computing: From logic to architectures[EB/OL]. [2019-01-20]. https://www.researchgate.net/publication/303542746_Invited_Cross-layer_approximate_Computing_from_logic_to_architectures.
- [8] Liang J H, Han J, Lombardi F. New metrics for the reliability of approximate and probabilistic adders[J]. IEEE Transactions on Computers, 2013, 62(9):1760-1771.
- [9] Han J, Orshansky M. Approximate computing: An emerging paradigm for energy-efficient design[EB/OL]. [2019-01-20]. <https://ieeexplore.ieee.org/document/6569370>.
- [10] Miao J, Gerstlauer A, Orshansky M. Approximate logic synthesis under general error magnitude and frequency constraints[EB/OL]. [2019-01-20]. <https://ieeexplore.ieee.org/document/6691202>.
- [11] Su S, Zou C, Kong W, et al. A novel heuristic search method for two-level approximate logic synthesis[EB/OL]. [2019-01-20]. <https://ieeexplore.ieee.org/document/8599060>.
- [12] Ichihara H, Inaoka T, Iwagaki T, et al. Logic simplification by minterm complement for error tolerant application[EB/OL]. [2019-01-20]. <https://ieeexplore.ieee.org/document/7357089>.
- [13] 薛宏熙, 边计年. 数字系统设计自动化[M]. 北京: 清华大学出版社, 1996:214-216.
- [14] 王伦耀, 夏银水, 陈偕雄. 基于多数覆盖的二级MPRM函数逻辑优化[J]. 电子与信息学报, 2012, 34(4): 986-991.
- [15] Shin D, Gupta S K. Approximate logic synthesis for error tolerant applications[EB/OL]. [2019-01-20]. <https://ieeexplore.ieee.org/document/5456913>.

Error rate calculation of logic functions using disjoint sharp product operation

YING Qiuhong, WANG Lunyao*, CHU Zhufei, XIA Yinshui

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

Abstract: In this paper an error rate calculation algorithm for two logic functions is proposed by comparing the differences between their logical covers. The algorithm mainly includes the disjoint sharp product operation and the double sharp product operation of logical covers, which are employed for the two covers disjointed operation and the common part deletion respectively. Using the proposed algorithm, the difference of the logical covers between two TB logic functions, or RM logic functions, are extracted and the error rate can be obtained by calculating the number of minterms in the extracted cover. The proposed algorithm is implemented in C and tested under MCNC benchmarks. The experimental results show that the proposed algorithm can carry out the error rate calculation for both TB functions and RM functions. Furthermore, it runs faster and is more efficient for large functions.

Key words: approximate computing; error rate calculating; logic optimization

(责任编辑 史小丽)