

基于冗余寄存器分类的时序网络面积优化算法

郝亚男^{1,2}, 杨海钢¹

(1 中国科学院电子学研究所 可编程芯片与系统研究室, 北京 100190; 2 中国科学院研究生院, 北京 100039)

摘 要: 本文根据时序网络中存在冗余的特点, 提出将冗余寄存器分为三类: 在所有可达状态空间输出端逻辑值恒为常数的寄存器、输入端相同的寄存器和对原始输出端逻辑值无影响的寄存器; 并提出了一种基于 AIGs(And-Inverter Graphs)的移除冗余寄存器的面积优化算法. 将三值模拟、寄存器共享和 COI(Cone of Influence)化简三种方法结合实现冗余寄存器的消除, 达到减少寄存器数和节点数, 优化时序网络面积的目的. 实验结果表明, 本算法可以使寄存器规模平均下降 23%, 节点数平均减少 26%.

关键词: 冗余寄存器; AIGs; 三值模拟; 寄存器共享; COI; 面积优化

中图分类号: TP31

文献标识码: A

文章编号: 1000-7180(2012)07-0035-07

Area Optimization Algorithm in Sequential Logic Based on Classification of Redundant Registers

HAO Ya-nan^{1,2}, YANG Hai-gang¹

(1 System on Programmable Chip Research Department, Institute of Electronics, Chinese Academy of Sciences, Beijing 100190, China; 2 Graduate School, Chinese Academy of Sciences, Beijing 100139, China)

Abstract: This paper presents a classification of the redundant registers, and provides three kinds of redundancy removal sequential optimization techniques according the characteristic of each kind of redundant registers. We classify these redundant registers into three categories: the first one is the constant register whose output value in all the reachable state space is always constant value, and the second one is those registers have the same inputs, and the third one is those registers have no influence for all the Primary Output value in the circuit. Based on AIGs (And-Inverter Graphs), Three-Valued simulation, sharing registers and COI reduction are combined together to realize the removal of the three kinds of redundant registers. It can reduce the numbers of the registers and logic gates to optimize the area of the sequential network. Experiments show that the presented algorithm can get an average reduction of registers by 23% and logic gates by 26% respectively.

Key words: redundant registers; AIGs; three-valued simulation; sharing registers; COI; area optimization

1 引言

逻辑优化算法已经经过近 40 年的活跃研究, 与工艺映射^[1]、布局布线等 CAD 流程一样是至关重要的环节. 目前组合逻辑优化算法已经相对成熟^[2-4], 而时序逻辑由于存在存储元件, 问题更加复杂, 所以对时序网络的优化仍然是当前研究的重点和热点^[5-6].

时序网络优化算法一般有三个目的: 功耗优化, 延迟优化, 面积优化. 之前的时序网络优化算法主要分为两种: (1) 基于重定时和重综合^[6-8]. (2) 基于信号一致性^[9-10]. 基于重定时和重综合时序优化算法缺点是逻辑结构变动较大, 会给后续的验证造成一定的困难; 基于信号一致性的时序优化算法具有可扩展性, 易于验证等特点.

本文在基于 AIGs(And-Inverter Graphs)时序

图的基础上,对存在的冗余寄存器进行分类,并根据每种类别的特点,提出专门的算法来实现对每种冗余寄存器的检测和消除,从而实现对时序网络面积的优化,减少寄存器个数,进而实现功耗降低,电路规模缩小的目的。

2 基本定义

本文提出的对冗余寄存器的分类和移除算法都是基于 AIGs(And-Inverter Graphs)时序网络的。本文根据文献[11]中对组合 AIGs 的定义进行了扩展,得到时序 AIGs 的定义。

定义 1 AIGs 组合图是由两输入与门和反相器组成的有向布尔网络 $G = \langle V, E \rangle$, 图中节点 V 对应于逻辑门(即 2 输入与门), 有向边 E 则代表逻辑门之间的连接关系。

AIGs 时序图是由组合 AIGs(两输入与门和反相器组成)和存储元件(寄存器)构成的有向布尔网络。网络中没有输入的节点为原始输入(Primary Input, 简称为 PI), 没有输出的节点为原始输出(Primary Output, 简称为 PO)。原始输入和原始输出定义了电路与外部环境的连接关系。如图 1 所示为一个电路图及其对应的布尔函数 AIGs 图。图 1(b)中的圆圈代表 2 输入与门, 黑点代表反相器。图 1(a)与图 1(b)是两种等价的电路表示形式。

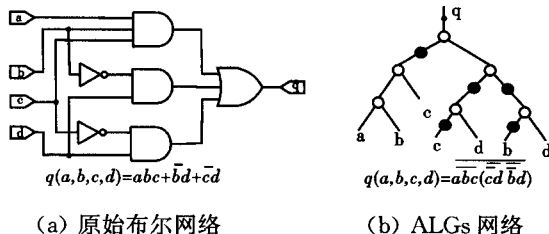


图 1 一个电路及其对应的 AIGs 图

定义 2 AIGs 的大小是指在 AIGs 图所包含的 2 输入与门的个数。AIGs 逻辑级数则是指从原始输入 PI 到原始输出 PO 的最长路径所经过的与门个数。

图 1(b)中的 AIGs 大小为 6, 逻辑级数为 3。AIGs 形式的布尔函数可以从其他形式的布尔函数转换得到。本文在实现中采用了文献[11]中一级结构化散列法实现 SOP(Sum of Product)转换为 AIGs 图。

3 算法实现

本文提出的算法目标是优化时序网络的面积。由于本文是在逻辑级实现的, 所以执行电路的面积

是与电路中的节点数 G 和寄存器数成正比的, 因此通过优化电路的节点数和寄存器数, 可以实现对电路面积的优化。

本文根据时序电路中冗余寄存器的特点, 将电路中的冗余寄存器分为三类:

- (1) 输出对所有 PO 无影响寄存器。
- (2) 具有相同输入端的寄存器。
- (3) 在可达状态空间输出恒为常数(0 或者 1)的寄存器。

对于第一种冗余可以通过 COI(Cone of Influence)化简算法和前向遍历搜索来实现检测和移除; 对于第二种冗余可以通过寄存器共享来实现减小寄存器个数的目的; 对于第三种输出为常数的冗余寄存器, 可以通过三值模拟在可达状态空间内实现对此类冗余寄存器的检测, 进而用常数节点代替这种输出恒为常数的冗余寄存器。通过这三种有效的移除方式的结合, 可以使电路中寄存器个数有效降低, 达到优化时序网络面积的目的。下面将对这三种冗余移除技术分别进行介绍。

3.1 COI 化简实现对 PO 输出无关的冗余寄存器的消除

COI 化简^[12]是一种易用有效的时序优化技术, 用于移除没有扇出到 PO 的寄存器。一个逻辑节点的影响锥 COI 是由该节点的输入节点(或寄存器)或者输入节点的输入构成的集合。此概念与节点的传递扇入 TFI(Transitive Fan In)类似。一个逻辑节点点的 COI 的计算算法如图 2 所示。

```

Search_Coi (Pobj, InfluenceSet)
if (Pobj) ∈ InfluenceSet
    return
else
    add Pobj to InfluenceSet;
    if (Pobj is an AND or PO) then
        for all inputs Fanins of Pobj
            Search_Coi(Fanins, InfluenceSet)
        end for
    else if (Pobj is a register) then
        Search_Coi(register_in, InfluenceSet)
    end if
end if
end Search_Coi

```

图 2 计算逻辑节点的 COI 的迭代算法

程序从一个逻辑门开始, 迭代遍历其所有输入端, 增加逻辑门到集合 InfluenceSet 中。直到没有新的逻辑能够增加到这个集合中, 则程序终止。假设

计中含有 n 个节点, m 个寄存器, 迭代计算一个节点的 COI 的最差情况就是将其他每个对象都访问一遍, 所以该算法复杂度为 $O(n+m)$ 。

一个逻辑门 g 的 COI 就是对 g 的输出有影响的节点集, 如果网络中的一个逻辑门 h 不属于 g 的 COI, 那么从网络中移除 h 将不会 g 的输出逻辑值产生影响。所以在时序网络中, 应用 COI 化简来消除对 PO 输出逻辑值没有影响的寄存器是一种有效的方法。首先计算出时序网络的每个原始输出 PO 的 COI, 并将它们的 COI 合并在一起。此时, 如果电路中的存在节点或者寄存器没有出现在这个合集中, 那么该节点或寄存器就可以被安全删除, 且完全不影响任意输出端的功能, 因此网络的规模在不影响任意输出的情况下, 可以得到显著降低。

3.2 寄存器共享的实现

由于在时序逻辑网络中, 经常存在具有多条扇出边的节点, 而且每条扇出边上可能含有不同个数的寄存器, 即时序电路中经常存在多个具有相同输入端的寄存器。鉴于此, 文中提出在具有多扇出边的节点或寄存器的输出端, 在保持路径延迟不变的情况下, 执行寄存器共享, 从而达到降低寄存器规模的目的。

一个节点的输出信号可能扇出到多个功能元件, 所以这一输出信号可以建立为具有一定权值的多条边。对于节点 u 的输出建立 k 路扇出的具有最大寄存器共享的扇出模型: 如果节点 u 的输出分别通过边 $u \xrightarrow{e_1} v_1, u \xrightarrow{e_2} v_2, \dots, u \xrightarrow{e_k} v_k$ 扇出到节点 $v_1, v_2, \dots, v_k$, 每条边 e_i (其中 $0 < i < k+1$) 经过了不同个数的寄存器, 因为节点 u 的多条输出边具有相同的信号, 那么实现电路功能最多需要的寄存器个数为经过寄存器数最多那条边的权值。

基于这一概念, 在一个节点 v 的输出边需要的寄存器个数为下面公式(1)所示, 其中的 $w(e_i)$ 代表 e_i 边经过的寄存器个数, 就是边 e_i 的权值。

$$\text{Req}_v = \max_a \{w(e_i); 1 \leq i \leq k\} \quad (1)$$

所以, 在每个含有多扇出边的节点 v , 可以减小的寄存器个数为 $\beta(v)$:

$$\beta(v) = \sum_{i=1}^{|FO(v)|} w(e_i) - \text{Req}_v \quad (2)$$

式中 $|FO(v)|$ 表示的是节点 v 的扇出数。对寄存器共享的实现过程, 首先是按照输入输出拓扑关系, 对每个寄存器进行遍历, 如果存在两个或若干个寄存器的具有相同的扇入地址, 那么说明这些寄存器具有相同的输入端, 则将这些具有相同扇入地址

的寄存器进行标志, 并将其中第一个访问到的寄存器作为代表寄存器, 将其余几个寄存器与各自的扇出断开, 并由该代表寄存器来驱动, 然后将余下的几个寄存器直接删除即可实现寄存器共享。

如图3所示, 圆圈代表2输入与门节点, 而灰色方框代表的是寄存器。节点 u 有4个扇出边, 分别驱动1个、3个、4个寄存器和一个与门节点 v_4 。

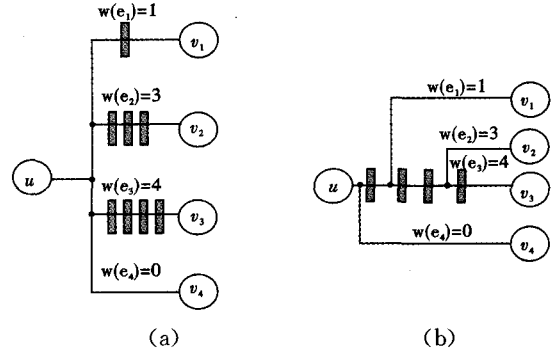


图3 在节点 u 的扇出端共享寄存器

图3(a)表示的是原始电路, 它使用 $1+3+4=8$ 个寄存器完成的执行设计; 图3(b)显示的是寄存器共享后的执行电路, 只用了 $\max(1, 3, 4)=4$ 个寄存器, 由于边 (u, v_4) 没有寄存器, 所以不能直接执行重定时操作, 但是可以执行寄存器共享。实现寄存器共享后, 时序网络中只需要4个寄存器, 在不影响时序性能的前提下, 比原来的执行设计(含有8个寄存器)减少了4个寄存器。

3.3 三值模拟对输出为常数的寄存器的检测

三值模拟(Three-Valued Simulation)用于逻辑模拟和冒险分析有一定的历史了, 三值模拟最早是由 Yoeli 和 Rinon 在 1964 年提出, 主要用于研究组合电路中的竞争冒险。到目前为止, 三值模拟已经被广泛用于检测冒险、错误诊断和形式验证等问题。

所谓三值逻辑模拟就是指利用三个逻辑值 0, 1, X(代表未知或不定态)进行模拟的过程。利用三值模拟可以实现对在所有可达状态空间内输出端为常数的寄存器的检测, 进而可以实现对此类冗余的移除。三值模拟首先在给定寄存器初始值的情况下, 将 X 值赋值给 PI 来进行电路仿真, 寄存器到达的三值状态集被收集, 当一个新的三值状态等于一个之前出现的三值状态时, 模拟停止。此时如果存在寄存器在所有可达的状态空间, 其值都是相同的常数(0 或 1), 那么该寄存器就是输出为常数的寄存器。下面是三值模拟算法的实现步骤, 它是从周期确认算法^[12]实现对类时钟信号的提取转化而来的。算法主要包含3个步骤:

(1) 进行初始化(将初始态存入已访问状态 V 中, S =初始态);

(2) 将 S 作为当前状态, X 作为所有 PI 的输入, 对电路进行模拟 $S' = \text{Simulation}(S)$, 得到下一状态集 S' ;

(3) 如果 S' 没有出现在已访问状态集 V 中, 则将 S' 并入已访问状态集中 $V = V \cup S'$, 并且赋值 $S = S'$ 返回执行步骤 2. 否则如果 S' 已经出现在已达状态集, 则停止模拟并对所有可达状态集按位取或, 状态值不变的寄存器即为所求冗余寄存器。

算法中使用的函数 $\text{Simulation}()$ 是从给定的状态开始, 把 X 值施加到每个 PI 输入端来计算电路中每个寄存器的下一状态. V 集合保存了当前已得到的所有寄存器的三值模拟状态值. 当新的状态 S_m 等于之前的某一状态 S_n 时, 那么一定会有 $S_{m+1} = S_{n+1}$, 因为 $S_{m+1} = \text{Simulation}(S_m) = \text{Simulation}(S_n) = S_{n+1}$.

表 1 所示为三值模拟实现的转换状态表, 通过该表来举例说明, 如何通过三值模拟算法来确认冗余寄存器的存在. 表 1 表示的电路中有 6 个寄存器 (分别为寄存器 $R_1, R_2, R_3, R_4, R_5, R_6$) 和 4 个原始输入端 (P_1, P_2, P_3, P_4), S_i 表示第 i 个寄存器的状态值, S_i' 表示第 i 个寄存器的下一状态值。

表 1 三值模拟实现的状态转换表

当前状态 $S_1, S_2, S_3, S_4, S_5, S_6$	原始输入 (P_1, P_2, P_3, P_4)	下一状态 $S_1', S_2', S_3', S_4', S_5', S_6'$
000000	XXXX	00X101
00X101	XXXX	011X0X
011X0X	XXXX	0X1101
0X1101	XXXX	0X010X
0X010X	XXXX	011X0X

因为所有寄存器的初始值为 0, 所以在第一次模拟后, 寄存器的初始值从输入端传到输出端, 所以此时状态集为全 0, 然后将 PI 赋值为 X , 进行模拟, 开始收集已访问的状态集, 当收集的新状态 (011X0X) 在已访问状态集中已经出现过时, 则停止模拟, 因为继续模拟得到的状态将会进入循环. 此时收集到所有的可达状态, 然后对所有状态按位取或, 如果或的结果状态的每一位就是每个寄存器的状态值的变化曲线. 如果存在某些寄存器的状态值一直保持不变, 则说明该寄存器的输出在可达状态空间内恒为常数. 从表 1 可以看出, 检测到的 S_1 和 S_5 的值不变, 则可以确认寄存器 R_1 和 R_5 为输出恒

为常数的冗余寄存器。

上述三值模拟检验冗余寄存器的算法一定会停止, 因为可能模拟次数的上界为 $3^{|\text{Regs}|}$, 其中 $|\text{Regs}|$ 为状态元件寄存器的个数. 尽管这个上界的值可能很大, 但是在实际应用中程序可以很快停止, 因为通过每一轮模拟中对所有的 PI 赋值为 X , 使得大部分的状态位尽快都变为 X . 若执行一定次数的模拟后, 每个寄存器的状态值为 X , 则说明电路中不存在输出为常数的寄存器. 当检测出所有的常数寄存器后, 将这些寄存器用常数节点 0 或 1 进行替换, 从而实现降低网络中寄存器规模的目的。

3.4 结合三种优化策略的算法实现

本文提出的对时序网络中冗余寄存器的确认和移除是通过将上述三种优化策略结合实现的. 结合三种优化策略对整体时序网络性能优化的算法伪代码如图 4 所示。

```

1. For( $i=0; i < n; i++$ )
2.   Search_Coi( $PO_i, \text{InfluenceSet}_i$ );
3.   Add influenceSeti to influence All;
4. End for
5. For( $i=0, i < m; i++$ )
6.   if(the register  $i$  is in the influenceAll)
7.     Return;
8.   Else
9.     Remove the register  $i$ ;
10. End for;
11. For( $j=0; j < k; j++$ )
12.   If(the node  $j$  is at least feeding two registers)
13.     Choose the representative register;
14. Remove other registers and use the representative
    to feed these outputs.
15. End if
16. End for
17. Visited=initial state,  $S$ =initial state
18.  $S' = \text{SimX}(S)$ 
19. If  $S' \in \text{Visited}$ 
20. then terminate and OR(state in Visited) and
    identify the constant register
21. else
22. then append  $S'$  to Visited, assign  $S'$  to  $S$ , and go to
    line 18

```

图 4 结合三种优化策略的算法实现

首先对电路中所有的原始输出 PO 进行遍历得到每个 PO 的 COI 集, 其中第 i 个 PO 的 COI 集合 InfluenceSet_i , 遍历结束后将这些 COI 集合进行合并, 得到所有 PO 的影响节点集合 influenceAll , 然后判断每个寄存器是否已经出现在 influenceAll 集

合中,如果没有则说明该寄存器是冗余的,可以进行直接删除. 然后对网络每个内部节点进行遍历,判断节点是否为多扇出节点且其扇出到至少两个寄存器,如果是的话,可以在此节点进行寄存器共享,按照输入输出拓扑关系,选择代表寄存器,然后将其他冗余寄存器与各自的扇出切断,并改由代表寄存器的输出来驱动. 最后进行三值模拟,将初始态分别并入已访问状态 Visited 和当前状态集 S 中,将 X 值赋值给 PI 来进行电路仿真,寄存器到达的所有三值状态被收集在 Visited 中,当一个新的三值状态等于一个之前出现的三值状态时,模拟停止. 此时如果存在寄存器的所有可达的状态值都是相同的常数,那么该寄存器就是输出为常数的冗余寄存器.

假设一个设计中含有 n 个 PO 节点, k 个 AIG 节点, m 个寄存器. 图 4 算法伪代码中第 1 行到第 10 行为得到对 PO 输出无影响的寄存器集合,那么 Search_Coi(POi, InfluenceSeti) 这个函数将执行 n 次,而 Search_Coi() 函数本身的时间复杂度为线性,所以该程序段的算法时间复杂度为 $O(k+m)$ 为线性的. 后面的第 11 行到第 16 行代码是实现寄存器共享是一种线性复杂度的执行算法,第 17 行到 22 行为三值模拟实现对常数寄存器的检测最坏情况是 $O(3^m)$,三值模拟是影响算法复杂度的主要因素,尽管是指数级的,但是由于 X 值的特殊性,在实际应用中程序通常可以很快停止.

4 实验结果

本文提出的基于 AIGs 的三种冗余寄存器移除

策略的结合来实现时序网络面积的优化,使用 C 语言实现. 在 IWLS2005^[13] 中的 ISCAS 中选取的 5 个较大规模电路 (s5378, s9234, s13207, s38584, s38417) 和 Altera 公司的公开测试集 Quip_9^[14] 中选取的 8 个经过 QuartusII 6.0 进行优化后的电路 (nut_000, nut_001, nut_002, nut_003, nut_004, oc_video_dct, radar12, oc_pci), 共计 13 个电路作为输入,进行优化性能测试.

表 2 所示为提出的算法对寄存器个数的优化性能结果. 表中“FFs”代表寄存器个数,“Level”代表电路的逻辑级数. 表 2 的第一列“电路名称”代表测试电路的名字,第二列为原始的未经优化电路的基本信息,第三列“Strashed AIGs”代表的是经过结构化散列后电路的基本信息,结构化散列将电路从 SOP 形式的逻辑网表转换为 AIGs 形式,逻辑级数和节点数都会发生变化,但是寄存器数不变,而且保证了没有任何两个与门的输入是相同的,这一步通过 UC berkeley 大学的逻辑综合验证工具 ABC^[15] 中的命令“strash”来实现的. 第四列为本文提出的优化算法三个方法,分别单独执行三个方法所得到的电路信息. 最后一列为通过三种算法结合得到的优化效果比例. 其中的(1)、(2)、(3)分别代表了使用第 3.1、3.2、3.3 节部分的冗余寄存器检测和移除算法得到的实验数据. 实验表明使用本文分别使用三个方法得到的寄存器个数与原始电路中的寄存器规模比例平均值分别为 97%、92%、82%.

选取的这 13 个测试电路中包含的寄存器个数从 164 到 3884 不等,使用三个优化策略的结合算法

表 2 本文算法对寄存器的优化

电路名称	原始电路		Strashed AIGs		本文提出的 3 个优化算法			(1)+(2)+(3) 优化效果
	FFs	Level	FFs	Level	(1)FFs	(2)FFs	(3)FFs	FFs
s5378	164	25	164	17	164	163	163	162
s9234	211	58	211	34	145	135	211	135
s13207	669	59	669	34	649	453	310	227
s38584	1452	56	1452	36	1451	1451	1396	1396
s38417	1636	47	1636	31	1564	1463	1636	1463
nut_000	328	63	328	90	328	320	313	307
nut_001	488	78	488	89	488	462	279	267
nut_002	203	17	203	32	203	188	111	103
nut_003	273	33	273	40	273	266	193	191
nut_004	186	24	186	23	185	178	142	140
oc_video_dct	3549	48	3549	51	3549	3537	3549	3537
radar12	3884	84	3884	92	3860	3846	3884	3846
oc_pci	1361	32	1361	46	1361	1353	981	975
平均比例	1				0.969	0.920	0.822	0.767

后得到的寄存器个数是原始电路的 76.7%，明显优于单独使用三种优化方法。

表 3 则是本文提出的优化算法对时序网络中节点数的优化能力。“nodes”代表网络中节点个数，“原始数据”是指原始电路中逻辑网络形式中 SOP 节点个数，实验数据表明本文提出的分别单独使用三个优化算法得到的网络中节点数是经过结构散列后网络中节点数的 96%，95% 和 78%。同理，使用三种

优化策略的结合算法得到的节点数是结构化散列后 AIGs 节点数的 74%。

需要特别说明的是，本文提出的优化算法不会对电路的时序特性造成影响，本文优化后电路的“Level”与表 2 中列“Strashed AIGs”中的“Level”是相等的，优化后电路的延迟与结构散列后的延迟保持一致。所以在表格中未将优化后电路的“Level”项列入表格中。

表 3 本文提出的优化算法对节点数的优化

电路名称	原始数据	Strashed AIGs	本文提出的 3 个优化算法			(1)+(2)+(3) 优化效果
	SOP nodes	nodes	(1)nodes	(2)nodes	(3)nodes	nodes
s5378	2779	1343	1343	1343	1342	1342
s9234	5597	1947	1058	1055	1947	1055
s13207	8027	2721	2616	2341	1104	1040
s38584	19407	12394	12394	12394	10456	10456
s38417	22397	9219	9021	8976	9219	8976
nut_000	1267	1195	1195	1195	1056	1055
nut_001	3620	4575	4575	4566	1602	1595
nut_002	685	806	806	805	438	438
nut_003	1964	2276	2276	2276	1808	1808
nut_004	816	506	503	503	270	270
oc_video_dct	49643	63420	63420	63408	63420	63408
radar12	18061	29910	29763	29754	29910	29754
oc_pci	5199	9409	9409	9409	7309	7309
平均比例		1	0.959	0.951	0.779	0.739

图 5 是对表 2 和表 3 中每个电路使用本文结合优化算法所能得到的寄存器个数和节点个数优化的比例。从图 5 中可以清晰的看出每个电路的优化比例。在不影响电路延迟的情况下，寄存器规模的优化范围从 0.3%~66%，平均优化了 23%，而节点数的优化范围从 0.1~65%，平均优化了 26%。

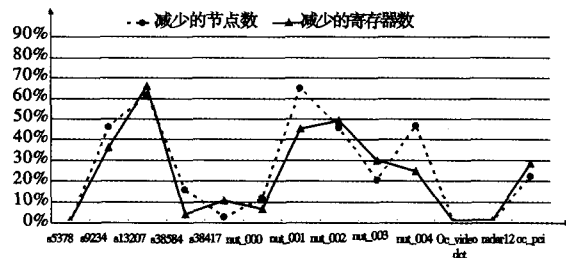


图 5 优化效果比例曲线图

5 结束语

本文提出了一种基于 AIGs 图的面积优化算法。首先使用 COI 算法检测对 PO 无影响的冗余寄存器，然后利用寄存器共享来减小多扇出节点输出

端驱动寄存器的个数，最后利用三值模拟检测出在可达状态空间恒为常数的寄存器。通过这三种技术的结合，可以使寄存器个数和时序网络规模明显减小。实验表明，在保持延迟特性不变的前提下，可以使寄存器规模平均降低 23%，节点数平均减少 26%，实现优化电路面积的目的。

参考文献:

- [1] Chen D, Cong J. DAOmap: A depth-optimal area optimization mapping algorithms for FPGA designs[C]// Proceedings of IEEE/ACM International conference on Computer-aided design. San Jose, USA, 2004: 752—759.
- [2] De Micheli G. Synthesis and optimization of digital circuits[M]. New York, McGraw-Hill Publishers, 1994.
- [3] Cong J, Ding Y. Combinational logic synthesis for LUT based field programmable gate arrays[J]. ACM Transactions on Design Automation of Electronic Systems, 1996, 2(1): 145-204.
- [4] Mishchenko A, Brayton R K, Jang S, et al. Delay opti-

- mization using SOP balancing[C]// Proceedings of International Workshop on Logic and Synthesis. San Diego, USA, 2011:75-82.
- [5] Hassoun S, Sasao T. Logic synthesis and verification [M]. New York: Kluwer Academic Publishers, 2002.
- [6] Das D, Wang J, Zhou H. iRetILP: an efficient incremental algorithm for min-period retiming under general delay model[C]// Proceedings of the 2010 Asia and South Pacific Design Automation Conference. Taipei, Taiwan, 2010:61-67.
- [7] Malik S, Sentovich E M, Brayton R K, et al. Retiming and resynthesis: optimization of sequential networks with combinational techniques[J]. IEEE transactions on Computer-Aided Design of Integrated Circuits and Systems, 1991, 1(10):74-84.
- [8] Jie-Hong R Jiang, Brayton R K. Retiming and resynthesis: a complexity perspective[J]. IEEE Transactions on Computer-Aided Design Of Integrated Circuits and Systems, 2006, 12(25): 2674-2686.
- [9] Mishchenko A, Case M L, Brayton R K, et al. Scalable and scalably-verifiable sequential synthesis[C]// Proceedings of IEEE/ACM International conference on Computer-aided design. San Jose, USA, 2008: 234-241.
- [10] Case M L. On invariants to characterize the state space for sequential logic synthesis and formal Verification [EB/OL](2011-02-14) <http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-46.html>.
- [11] Mishchenko A, Chatterjee S, Robert Brayton. FRAIGs: a unifying representation for logic synthesis and verification [R]. ERL Technical Report, EECS Dept, U.C. 2005.
- [12] Bjesse P, Kukula J. Automatic generalized phase abstraction for formal verification[C]// IEEE/ACM International conference on Computer-aided design Proceedings, San Jose, USA, 2006:1076-1082.
- [13] IWLS 2005 Benchmarks[EB/OL][2011-05-16]. <http://iwls.org/iwls2005/benchmarks.html>.
- [15] Berkeley Logic Synthesis and Verification Group. ABC: A System for Sequential Synthesis and Verification[EB/OL][2011-05-06]. <http://www.eecs.berkeley.edu/~alanmi/abc>.

作者简介:

郝亚男 女, (1983-), 博士研究生, 研究方向为规模集成电路计算机辅助设计。

杨海钢 男, (1960-), 研究员, 博士生导师. 研究方向为高性能可编程逻辑芯片设计技术、数模混合信号 SOC 设计技术。

(上接第 34 页)

5 结束语

本文提出了一个在函数调用点减少冗余内存访问的编译优化算法。该算法基于扩展的调用关系图, 统计每个函数的寄存器使用信息, 可以计算得出每个函数的最小寄存器保存集合, 从而达到减少冗余内存访问的目的。而且由于访存指令功耗较大, 该算法同时也可以减少内存占用空间和功耗。通过对 DSPstone 测试集进行评估, 该算法能够平均减少 40.36% 的访存操作, 并获得平均 2.99% 的性能提升。下一步计划将文献[4]中的算法移植到本算法中, 从而对数组密集访问应用的性能进行优化。另外, 本文还将使用更复杂的测试集对本方案进行评估。

参考文献:

- [1] Aho A V, Sethi R, Ullman J D. Compilers, principles, techniques and tools [M]. Boston: Addison Wesley, 1988.
- [2] Zhang Y, Lee H J, Register allocation by systematic

merge of register-reuse chains [J]. ELECTRONICS Letters, 2000, 36(1):16-17.

- [3] Benitez M, Davidson J. A retargetable integrated code improver[R]. Univ. of. Virginia, Dept of Computer Sc., Technical Report CS-93-64. 1993.
- [4] Davidson J, Jinturkar S, Memory access coalescing: a technique for eliminating redundant memory accesses [C]// proceedings of PLDI. USA: Flor: da, June 1994.
- [5] Barik R, Sarkar. V. Inter-procedural load elimination for dynamic optimization of parallel programs [C]// proceedings of PACT. Canda: Toronto, 2009: 41-52.
- [6] Frederick C. Minimizing register usage penalty at procedural calls [C]// Proceedings of PLDI USA: Georgia, 1988: 85-94.
- [7] DSPstone benchmark suits. [EB/OL][2011-10-12] <http://www.ert.rwth-aachen.de/Projekte/Tools/DSP-STONE/dspstone.html>.

作者简介:

时磊 男, (1983-), 博士. 研究方向为计算机系统于网络应用。