快速设计高性能有符号乘法器电路的编程语言研究

焦继业[1],穆 荣[2],郝 跃[1]
(1.西安电子科技大学宽禁带半导体材料与器件教育部重点实验室,陕西西安 710071;2.西安科技大学网络中心,陕西西安 710054)

摘 要: 提出了一种有符号乘法器电路的编程语言,其核心思想是采用指令表示乘法器的编码器、加法器树、快速加法器等三个部分,然后经由指令描述互联关系形成乘法器.通过 Lex/Yacc 构成编译器,解析程序得到乘法器的 Verilog 代码.采用该设计语言生成的七种典型结构的 32 位有符号单周期乘法器,在 200MHz 工作频率设定下,使用 GRACE 0.18μm 1P6M 工艺,进行逻辑综合、布局布线、静态时序和功耗分析.实验结果表明,这七种乘法器速度都优于 Synopsys Design Ware 产生的乘法器,其中由改进型 Booth Radix4 编码、冗余二进制加法器树和跳跃进位加法器构成的乘法器综合性能超出 Synopsys Design Ware 产生的乘法器达 35%,因此该设计语言可应用于高性能乘法器电路快速设计应用中.

关键词: 乘法器; 编程语言; 编码; 加法器树; 快速加法器
中图分类号: TP342.22 **文献标识码:** A **文章编号:** 0372-2112 (2013) 11-2256-06
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2013.11.023

A Programming Language for Rapid Design of High Performance Signed Multiplier Circuits

JIAO Ji-ye[1], MU Rong[2], HAO Yue[1]
(1. *Key Laboratory of Ministry of Education for Wide Band Gap Semiconductor Materials and Devices, Xidian University, Xi'an, Shaanxi* 710071, *China*; 2. *Network Center, Xi'an University of Science and Technology, Xi'an, Shaanxi* 710054, *China*)

Abstract: This paper presents a programming language for designing signed multiplier circuit. The key idea is using instruction to express the encoding units, addition tree units and fast adder units of multiplier, and using the connection of instruction description to obtain a multiplier. The multiplier of program through Lex and Yacc translate source code containing connection into Verilog code. Seven typical structures of 32 bits signed multipliers are obtained by the instruction description. Under 200MHz synthesis condition and in GRACE 0.18μm process, these multipliers are run for logic synthesis, placed and routed, static timing analysis, and power analysis. The experiment results suggest that the speeds of all the seven multipliers show advantage over that produced by Synopsys design ware , and the multiplier performance composed of modified Booth Radix4 encoding, redundant binary addition tree and carry skip adder exceeds that produced by Synopsys design ware by 35%. Therefore, this language can be used to the application of high performance multiplier design.

Key words: multiplier; programming language; encoding; addition tree; fast adder

1 引言

对于嵌入式图形顶点处理器,需要使用高速乘法器完成图形顶点的各种运算.有符号乘法器主要分为阵列乘法器[1]和布斯(Booth)[2]乘法器两种,而这两种实现方法下又细分多种实现方式[1~3],如编码规则、加法器树结构是否适合版图实现,加法器求和速度等.在众多的乘法器结构中,如何找出哪种结构更适合设计需求是一件繁琐复杂的工作.因此需要一种能快速描述各种结构乘法器的设计方法,用于评估各种结构乘法器的面积、运行速度和功耗等关键参数.

本文提出了一种设计高性能有符号乘法器的编程

收稿日期:2012-11-30;修回日期:2013-04-08;责任编辑:李勇锋
基金项目:陕西省自然科学基金 (No.2009JM8004)

语言,其核心思想是将乘法器的编码器、加法器树、快速加法器等三个部分的基本单元分离出来,采用指令表示基本单元功能和互联关系.由于乘法器各部分模块的规则性,可以采用多条指令描述基本单元的互联关系,以构成不同结构的乘法器.采用 Lex/Yacc 设计编译器,解析所描述的高速有符号乘法器程序得到 Verilog 代码.为了验证生成的乘法器性能,设计了七种常用结构的 32 位有符号乘法器,并与 Synopsys Design Ware 所产生乘法器进行面积、速度和功耗的性能比较.

2　乘法器编程指令设计

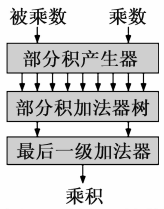


图1　乘法器结构

二进制有符号乘法器的设计实现归结为三个步骤,如图 1 所示,首先将二进制的被乘数与乘数的每一位相乘得到部分积;然后扩展符号位,将部分积按权值相加,进行部分积压缩;最后将最终压缩得到的两个部分积求和,处理符号位,得到乘法器运算结果.目前提出的乘法器编程语言在编码器阶段支持改进型 Baugh-Wooley 编码和改进型 Booth Radix4 的编码的指令;然后在加法器树阶段,支持 4 – 2 加法器(4 – 2 Compressor Addition),7 – 3 加法器(7 – 3 Counter Addition),3 – 2 加法器和冗余二进制加法器(Redundant Binary Addition)的指令编码;在最后的快速加法器求和阶段,支持超前进位加法器(CLA)和进位跳跃加法器(CSA)的快速加法器指令编码.

2.1　编码器指令

2.1.1　改进型 Baugh-Wooley 编码器指令

Baugh-Wooley 算法[1]是一种二进制补码形式的带有正负符号数的阵列乘法器算法,其部分积编码输出规律,允许所有的部分积带符号位操作,因此可以提高乘法器编码运算速度.图 2 给出了 8x8 改进型有符号 Baugh-Wooley 乘法器编码结构,其结构整齐,采用基本的与门单元形成部分积的输出,延迟较少,并且不用考虑符号位扩展运算.研究图 2,可以采用三种指令表示 Baugh-Wooley 的编码规则,指令如表 1 所示,

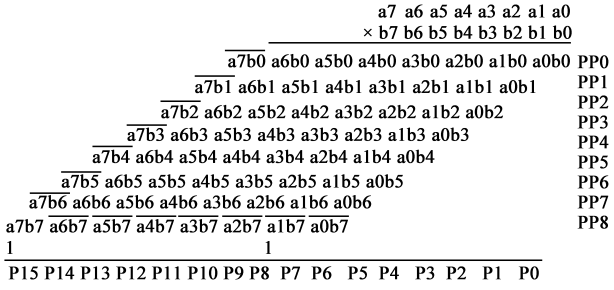


图2　8×8改进型Baugh-Wooley乘法器

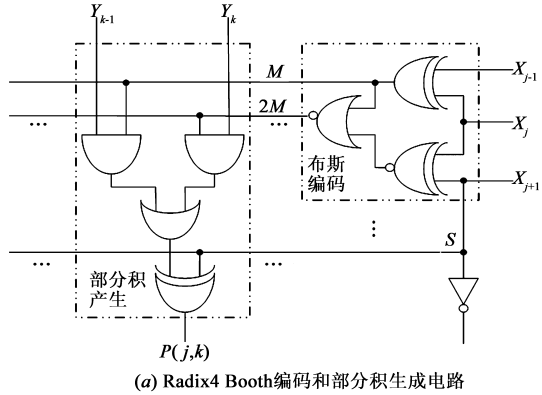
图 2 中的 PP0 至 PP6 可以采用 bw_pp0 进行描述,PP7 采用 bw_pp1 描述,PP8 采用 bw_pp2 描述.

表 1　Baugh-Wooley 编码指令

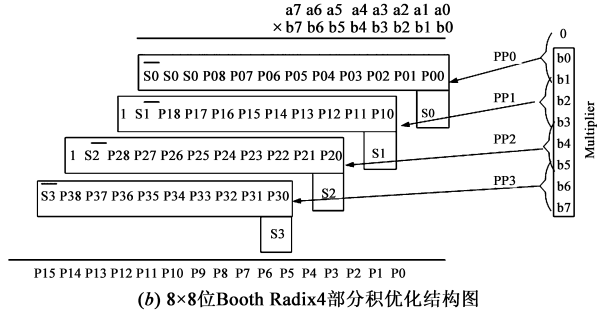
指令助记符	指令格式	备注
bw_pp0	bw_pp0 pp0,x,y.0	pp0 表示编码输出结果;x 表示
bw_pp1	bw_pp1 pp7,x,y.7	被乘数;y 表示乘数;y.0 表示采用
bw_pp2	bw_pp2 pp8,x,y.8	第 0 位编码,同时 0 表示右移位数

2.1.2 改进型 Booth Radix4 的编码器指令

Booth 算法[2,3]是乘法器中一种有效编码方法,有符号数和无符号数可采用统一的编码方式,只须做相应的符号位扩展,并且经编码生成的部分积数目可以大大缩减,提高了乘法器的加法器树运算速度.乘数 X 采用 Booth Radix4 编码可生成的部分积为 $-2M$, $-1M$, 0, $+1M$, $+2M$(M 表示被乘数),其编码和部分积生成电路如图 3(a)所示.根据文献[4]中所提供算法,图 3(b)给出了 8×8 的有符号最优结构的 Booth Radix4 编码结构图,其符号位计算得到了最优处理.相对于 Baugh-Wooley 编码器,使用 Radix4 Booth 编码输出的分积个数几乎可以减少一半,但是编码路径延迟要长的多.研究图 3,可以得到如表 2 所示的四种 Booth Radix4 编码器指令.图 3(b)中的 PP0 可以采用 booth_pp0 进行描述,PP1 和 PP2 采用 booth_pp1 描述,PP3 采用 booth_pp2 描述.



(a) Radix4 Booth编码和部分积生成电路



(b) 8×8位Booth Radix4部分积优化结构图

图3　改进型Booth Radix 4乘法器

表 2 Booth Radix4 编码指令

指令助记符	指令格式	备注
	booth_encode	将乘数 y 进行 Booth Radix4 编码,
booth_encode	booth_code,y	存储为 booth_code
	booth_pp0	将被乘数 x 与 booth 编码的 0 进行
booth_pp0	pp0,x,booth_code.0	编码
	booth_pp1	将被乘数 x 与 booth 编码的 1 进行
booth_pp1	pp1,x,booth_code.1	编码
	booth_pp2 pp4,pp3,x	将被乘数 x 与 booth 编码的 3 进行
booth_pp2	booth_code.3	编码

2.2 加法器压缩树指令

加法器压缩树指令目前支持 4 – 2 加法器(4 – 2 Compressor Addition),7 – 3 计数器加法器(7 – 3 Counter Addition),3 – 2 加法器和冗余二进制加法器(Redundant Binary Addition)等四种加法器树指令,如表 3 所示.

表 3 加法器压缩树指令

指令助记符	指令格式	备注
wallace	wallace wc,ws, pp0,pp1,pp2	Wallace 3 – 2 加法器压缩规则
comp4_2	comp4_2 wc,ws, pp0,pp1,pp2,pp3	4 – 2 加法器压缩规则
counter7_3	counter7 – 3 w2,w1, w0,pp0,pp1,pp2, pp3,pp4,pp5,pp6	7 – 3 计数器压缩规则
rba_encode	rba_encode rb,pp0	将 pp0 转换成冗余二进制数 rb
rba_decode	rba_decode s0,rb	将冗余二进制数 rb 转换成为 s0
rba4_2	rba4_2 wc,ws,pp0, pp1,pp2,pp3	Rba 加法器压缩规则

2.2.1 华莱士树(Wallace Tree)指令

华莱士树又称为 3 – 2 压缩树,是通过提高电路的并行度来提高运算速度的一种结构.采用 3 – 2 的进位保留加法器构成华莱士压缩树时,其乘法器延迟时间约正比于 $\log_{3/2}(n/2)$(n 为乘法器输入位数)[5],所以其电路延迟相对于简单阵列 n 位乘法器是具有优势的.指令定义如表 3 所示,当编译该条指令时,会采用 3 – 2 进位保留加法器描述加法器的构成.

2.2.2 4 – 2 压缩器指令

4 – 2 压缩器的出现改变了以往基于全加器的部分积压缩标准方式,引入了水平数据通路,减少了压缩树的级数.由于 4 – 2 压缩器规整的互连,使它极易构造规整的、低复杂度的树型结构.图 4 给出了一种基于多路选择器的 4 – 2 压缩器电路结构,该结构的关键路径延时为 3XOR,从不同数据输入到输出的延时比较均衡,确保能够同时产生 Sum 和 Carry 信号.指令定义如表 3 所示.

2.2.3 7 – 3 计数器指令

7 – 3 计数器树可以很好压缩输入位数,图 5 给出了由四个 3 – 2 加法器构成的 7 – 3 计数器加法器.指令定义如表 3 所示.

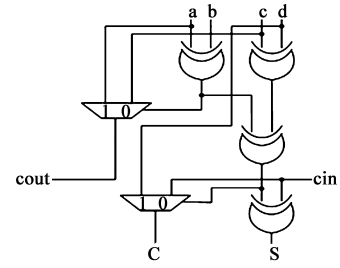


图4 基于多路选择器的4-2压缩器电路结构

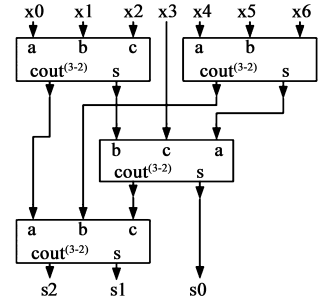


图5 基于3-2压缩器构成的7-3压缩器

2.2.4 冗余二进制加法器指令

冗余二进制加法器(RBA)[6,7]可完成两组冗余数值加法运算,如图 6 所示,冗余二进制加法器也是一种 4 – 2 压缩器,但其拥有无进位传播的优势,且运算架构具有规则性,非常适合电路布线,可以提高乘法器速度和面积综合性能.冗余二进制加法器输入输出数据是冗余二进制,所以输入输出结果需要转成二进制,用于下一级运算.指令定义如表 3 所示.

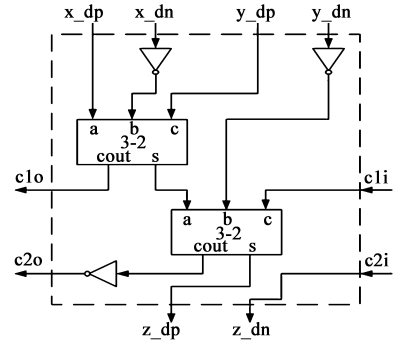


图6 由全加器构成的冗余二进制加法器

2.3 求和加法器指令

经过加法器树压缩输出两组数据,此时需要快速加法器进行求和运算,得到最终乘积结果.快速求和加法器指令目前支持超前进位加法器(CLA)和进位跳跃加法器(CSA)的快速加法器指令,如表 4 所示.

表 4　加法器树编码指令

助记符	指令格式	备注
cla	cla out,in1,in2	超前进位加法器
csa	csa out,in1,in2	进位跳跃加法器

2.3.1 超前进位加法器(Carry Look-ahead Adder)指令

超前进位加法器(CLA)是将所有的进位并行产生,而不用等待加法器进位输出传播,其运算原理如式(1)所示,Gi 为进位产生位,Pi 为进位传播位,通过递归,可以并行得到加法器的进位运算,因此加快了加法器的求和速度.指令定义如表 4 所示.

$$c_{i+1} = x_i y_i + (x_i \oplus y_i) c_i,\ G_i = x_i y_i,\ P_i = x_i \oplus y_i,$$
$$c_{i+1} = G_i + P_i c_i \qquad (1)$$

2.3.2　进位跳跃加法器(Carry Skip Adder)指令

进位跳跃加法器(CSA)利用进位传播关系 $p = a + b$,可以将加法器组的输入进位直接跳跃到加法器输出进位上,加快了进位的传播时间.图 7 是由 4 个全加器组成的进位跳跃加法器.指令定义如表 4 所示.

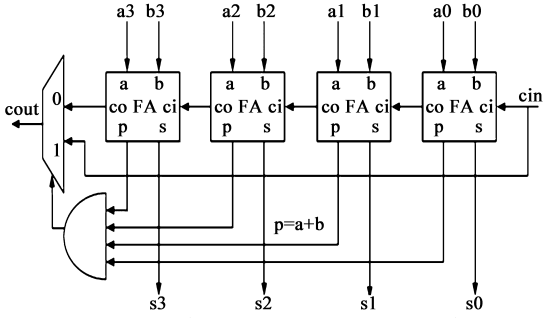


图7　由4个全加器构成的进位跳跃加法器

3　编译器设计

3.1　由 Lex/Yacc 构成编译器

编译工具 Lex 和 Yacc[8,9]分别是词法分析程序和语法分析程序的自动生成软件.他们由贝尔实验室开发,并广泛应用于编写编译程序.按照 Lex 和 Yacc 规范,以及前面定义的指令,编写词法规则和语法规则,生成乘法器设计语言的解释器,并增加词法、语法错误定位功能.程序代码分为 data 段和 code 段,data 段用于定义乘法器的输入宽度,code 段为乘法器结构描述.

3.2　乘法器编程语言与 Synopsys Design Ware 乘法器比较

图 8 给出了采用乘法器编程语言设计的 16 位有符号乘法器,采用架构为 Booth Radix4 编码、华莱士压缩树和超前进位加法器.图 9 是采用 Synopsys Design Ware 规则编写的 32 位有符号乘法器,采用布斯编码和华莱士树架构.Synopsys Design Ware 产生的乘法器,虽然可以设定部分参数,但是无法控制编码规则、加法器树的多种选择和快速加法器的类型.

```
.data
;=====input define====================
multiplier   x,16
multiplicand  y,16

.code
;=====booth encoding==================
booth_decode  booth , y       ; booth coding, m0-m7

;====booth patial encoding==============
booth_pp0     pp0,x,booth.m0   ;pp0[19:0]
booth_pp1     pp1,x,booth.m1   ;pp1[20:0]
booth_pp1     pp2,x,booth.m2   ;pp2[22:2]
booth_pp1     pp3,x,booth.m3   ;pp3[24:4]
booth_pp1     pp4,x,booth.m4   ;pp4[26:8]
booth_pp1     pp5,x,booth.m5   ;pp5[28:10]
booth_pp1     pp6,x,booth.m6   ;pp6[30:12]
booth_pp2     pp8,pp7,x,booth.m7 ;pp7[31:14]

;====adder tree=========================
;stage0
wallace       w0_0, w0_1, pp0  , pp1  , pp2
wallace       w0_2, w0_3, pp3  , pp4  , pp5
wallace       w0_4, w0_5, pp6  , pp7  , pp8

;stage1
wallace       w1_0, w1_1, w0_0, w0_1, w0_2
wallace       w1_2, w1_3, w0_3, w0_4, w0_5

;stage2
comp4_2       w2_0, w2_1, wc1_0, w1_1, w1_2, w1_3

;====final fast adder==============
cla           out,w2_0,w2_1
```

图8　16位有符号乘法器语言描述

```
`timescale 1ns/1ns

module syn_mul( in1, in2, out);
input  signed [31:0] in1;
input  signed [31:0] in2;
output signed [63:0] out;

reg signed [63:0] out;

always @ (*)
begin : signed_mul
  /* synopsys resource r0:
     map_to_module = "DW02_mult",
     implementation = "Wall",
     ops = "a1"; */
  out = in1 * in2;
end
endmodule
```

图9　Synopsys Design Ware 32位有符号乘法器

3.3　Perl/Tk 构成的乘法器生成用户界面和乘法器验证

根据乘法器设计语言的指令定义和编译规则,设计了一个 Perl/Tk 图形界面乘法器生成程序,如图 10 所示,可选择乘法器输入位宽、编码、压缩器和快速加法器规则.采用 Perl 生成乘法器程序,经由编译器自动生成不同输入位宽的乘法器,并输出 Verilog 代码.使用图形界面操作,可以方便用户快速生成不同结构的乘法器.搭建统一的 VVM 验证平台验证所有产生的乘法器 IP,验证结果表明产生的乘法器 IP 逻辑正确.

4　性能评估

为了验证由乘法器编程语言生成的乘法器性能,由图 10 所示的图形界面生成七种典型结构的 32 位有符号单周期乘法器,与图 9 所示 Synopsys DesignWare 的描述的乘法器构成八种乘法器.具体结构见表 5 乘法器结构栏,为了方便描述,采用 Arch1 至 Arch8 将乘法器统一编号.

八种乘法器统一采用 Grace 0.18μm 数字工艺库,使用 Synopsys Design Compiler (2010.03.sp5 版本)进行逻辑综合,综合条件设定为 200MHz,输入输出延迟为 0.1ns,综合结果如表 5 所示.从综合结果可以得到 Synopsys DesignWare 产生的 32 位有符号乘法器的速度仅为

12.98nS,远远低于 200MHz 的设计要求.由于综合工具产生的面积、功耗和最长路径延迟都采用的是预估策略,其结果有不确定性,因此需要进行实际的布局布线后以统计各种参数.

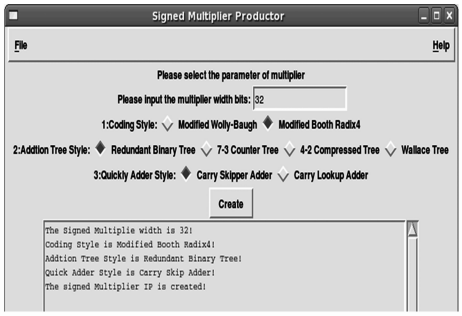
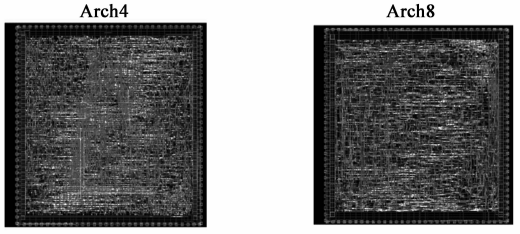


图10 由Perl/Tk构成的乘法器IP产生工具

采用布线工具为 ICC(2010.03.sp5 版本),选择 Grace 0.18μm 1P6M 工艺,将八种乘法器逻辑综合后的网表进行布局布线.布线规则为电源环采用 10μm 宽度,芯片利用率设定为 93%,Floorplan 形状为方形.将八种乘法器进行布局布线,这里给出其中两种乘法器布线结果,如图 11 所示.可以看出图 Arch4 比 Arch8 布线更密,连线更复杂.八种乘法器布线后的面积统计结果见表 6 中面积栏.对比 Arch1、Arch2、Arch3 和 Arch5 的布线面积,可以得出:冗余二进制加法器压缩树因其结构优势,相对于华莱士树、4 – 2 压缩树和 7 – 3 计数器树有更小的面积;而快速加法器 CSA 面积比 CLA 面积更小.



(*a*) Booth Radix4,RBA Tree,CSA (*b*) Synopsys Design Ware

图11 Arch4和Arch8布线结果

由于需要进行功耗对比,为了保证乘法器的输入数据是一致的,所以通过验证平台自动随机生成一万组随机 32 位有符号数据,将这些输入数据加载到八个布局布线后的乘法器网表,经仿真产生 VCD 波形,用于功耗分析.采用 PrimeTime PX 2010.06.sp3 进行功耗分析,分析结果如表 6 中功耗一栏所示.对比 Arch3、Arch5、Arch6 和 Arch7,可以得出采用改进型 Baugh-Wooley 编码比改进型 Booth Radix4 编码所消耗的功耗更少;对比 Arch3 和 Arch4,CSA 加法器比 CLA 加法器所消耗的功耗更少.采用 PrimeTime (2010.06.sp3 版本)进行乘法器的静态时序分析,得到最差(Max)和最好(Min)两种工作环境下的最长路径分析结果,如表 6 所示.由实验结果可以得出 4 – 2 压缩树比华莱士树、7 – 3 计数器树和冗余二进制加法器树有更快的压缩速度;CLA 速度快于 CSA 加法器.

表 5 八种乘法器在 Grace 0.18μm 工艺 200MHz 综合结果

乘法器结构	面积 (μm^2)	功耗 (mW)	最长路径 (nS)
Arch1:Booth Radix4, 4 – 2 Compress Tree,CLA	268314.08	65.92	5.25
Arch2:Booth Radix4, 7 – 3 Counter Tree,CLA	233636.36	57.47	5.38
Arch3:Booth Radix4, RBA Tree,CLA	230163.59	59.72	5.92
Arch4:Booth Radix4, RBA Tree,CSA	194617.69	45.68	6.66
Arch5:Booth Radix4, Wallace Tree,CLA	271424.26	63.59	5.32
Arch6:Baugh-Wooley, RBA Tree,CLA	296718.21	60.55	5.92
Arch7:Baugh-Wooley, Wallace Tree,CLA	336864.53	62.40	5.42
Arch8:Synopsys Design Ware	139762.05	23.56	12.98

表 6 八种乘法器采用 Grace 0.18μm 1P6M 工艺布局布线后的统计结果

乘法器结构	面积 (μm^2)	功耗 (mW)	最长路径 (nS)(Max)	最长路径 (nS)(Min)
Arch1	536.58 * 534.24	31.4	5.1295	4.9836
Arch2	500.94 * 498.96	28.7	5.1691	4.9791
Arch3	496.98 * 493.92	34.8	5.6427	5.5016
Arch4	457.38 * 453.6	25.0	6.0658	5.8852
Arch5	539.88 * 539.28	31.7	5.2362	5.0635
Arch6	564.3 * 559.44	31.6	5.8057	5.4811
Arch7	601.26 * 599.76	28.7	5.2915	5.0692
Arch8	387.42 * 383.04	29.6	12.7156	12.5249

为了综合比较这些乘法器性能,定义一个新的参数 *Perf*,如式(2)所示,表示面积、功耗和速度的乘积,当 *Perf* 为最小时,表示该乘法器性能在该条件下最优.为了方便表示所有数据,将七种乘法器相应数据除以 Synopsys Design Ware 的乘法器相应数据(仅统计最差工作环境下的最长路径),得到如图 12 所示的归一化性能参数图.从图中可以看出,由 Booth Radix4 编码、冗余二进制加法器树和跳跃进位加法器构成的 Arch4 乘法器,

由于其面积和功耗优于其他乘法器,其综合性能超出 Synopsys Design Ware 乘法器 Arch8 的综合性能达 35%,性能达到了最优.

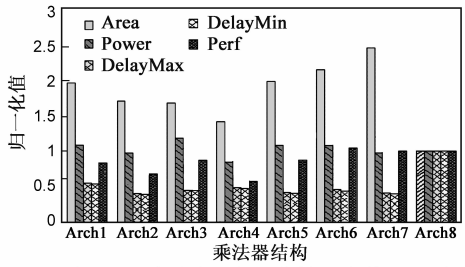


图12　八种乘法器归一化性能参数对比图

综上所述,采用乘法器编程语言实现的 32 位高性能有符号乘法器相对于 Synopsys Design Ware 的乘法器具有优越的性能,乘法器实现快速简单.

$$Perf = 面积 \times 功耗 \times 最长路径 \qquad (2)$$

5　结论

本文提出的乘法器设计语言,可以快速描述乘法器架构,通过编译器可以直接输出乘法器 Verilog 代码.采用乘法器设计语言可以设计出高性能的有符号乘法器,具有开发周期短,乘法器性能优越,并可快速评估不同结构乘法器性能.通过乘法器设计语言产生的七种 32 位有符号乘法器,其速度都优于 Synopsys Design Ware 产生的乘法器,其中采用 Booth Radix4 编码、冗余二进制加法器树和跳跃加法器的乘法器架构性能最优,其综合性能超出 Synopsys Design Ware 产生乘法器的 35%.但目前加法器树和快速加法器描述仅限于论文中的几类结构,后期工作将会添加多种加法器和快速加法器结构,用于生成更多样的乘法器电路.

参考文献

[1] Chen Yuan-Ho. A high-accuracy adaptive conditional-probability estimator for fixed-width booth multipliers[J]. IEEE Transactions on Circuits and Systems, 2012(3): 594 – 603.

[2] Swee K L S. Performance comparison review of 32-bit multiplier designs[A]. 2012 4th International Conference on Intelligent and Advanced Systems[C]. Kuala Lumpur: IEEE, 2012. 836 – 841.

[3] Bansal M, Nakhate S, Somkuwar A. High performance pipelined signed 64x64-bit multiplier using radix-32 modified booth algorithm and wallace structure[A]. Proceedings of Computational Intelligence and Communication Networks[C]. Gwalior: IEEE, 2011. 411 – 415.

[4] N Ravi, T Subba Rao, T Jayachandra Prasad. Pipelined C2Mos register high speed modified booth multiplier[J]. International Journal of Advanced Networking and Applications, 2011(V3 – 01): 1031 – 1034.

[5] K Lal Kishore. V S V Prabhakar. VLSI Design[M]. New Delhi: I K International Pvt Ltd, 2009. 228.

[6] Sahoo S K, Chakraborty S. A high speed, high radix 32-bit redundant parallel multiplier [A]. Proceedings of Emerging Trends in Electrical and Computer Technology [C]. Tamil, Nadu: IEEE, 2011. 917 – 921.

[7] Sahoo S K, Reddy K S. A high speed FIR filter architecture based on novel higher radix algorithm[A]. Proceedings of VLSI Design[C]. Hyderabad: IEEE, 2012. 68 – 73.

[8] Upadhyaya M. Simple calculator compiler using Lex and YACC [A]. 2011 3rd International Conference on Electronics Computer Technology[C]. Kanyakumari : IEEE, 2011. 182 – 187.

[9] Beevi S N, Prasad D C, Chandra S S V. Enhancing flexibility and portability of execution preserving language transformation using meta programming[A]. 2012 International Conference on Power, Signals, Controls and Computation[C]. Thrissur: IEEE, 2012. 1 – 4.

作者简介

焦继业　男,1977 年 9 月生于新疆乌鲁木齐.西安电子科技大学微电子学院博士研究生,主要从事低功耗高性能计算单元设计、移动三维图形处理器设计和应用、数模混合电路设计等方面的研究工作.

E-mail: jiaojiye@126.com

穆　荣　女,1979 年 10 月生于陕西西安.现为西安科技大学网络中心工程师.主要从事移动三维图形处理器设计和应用.嵌入式系统设计等方面的研究工作.