DOI:10.19304/j.cnki.issn1000-7180.2012.12.001

面向算术单元的 FPGA 工艺映射算法

路宝珠[1,2]，杨海钢[1]，郝亚男[1,2]，张茉莉[1,2]，崔秀海[1]
(1 中国科学院 电子学研究所，北京 100190；2 中国科学院研究生院，北京 100049)

摘 要：本文提出了一种针对算术单元的 FPGA 工艺映射算法 ArithM. 实验结果表明，与公认 ABC 中的黑盒子映射算法相比，本文算法能平均减少逻辑单元面积 7%，减少电路关键路径延时 5%. ArithM 采用了单元共享、平衡算术链以及吸收邻近节点三种方法来优化算术资源.

关键词：FPGA；工艺映射；算术单元；单元共享
中图分类号：TN402 **文献标识码**：A **文章编号**：1000－7180(2012)12－0001－06

Arithmetic Operation Oriented FPGA Technology Mapping Algorithm

LU Bao-zhu[1,2], YANG Hai-gang[1], HAO Ya-nan[1,2], ZHANG Mo-li[1,2], CUI Xiu-hai[1]
(1 Institute of Electronics, Chinese Academy of Sciences, Beijing 100190, China;
2 Graduate University of the Chinese Academy of Sciences, Beijing 100049, China)

Abstract: This paper proposes arithmetic mapping algorithm, ArithM. Experiments show that, compared with accepted ABC black box mapping algorithm, this algorithm can reduce area of logic blocks by 7%, and decrease circuit delay by 5%. AithM adopts three optimizations—components sharing, balancing arithmetic list and absorbing abutting resource, to reduce the mapping cost of area and delay.

Key words: FPGA; technology mapping; arithmetic components; components sharing

1 引言

FPGA 具有高灵活性、资源丰富等特性，但在性能和代价花费上，和 ASIC 相比还存在差距[1－2]. 通过在 FPGA 中采用专用硬核模块是现今消除这种差距的有效方法之一. 在实现特定专用功能时，通过采用专用硬核模块(如加法器、嵌入式存储器、数字信号处理器等)，减少了电路的面积开销，提高了电路的性能.

大型设计中的很大一部分是由加法器、减法器、乘法器等算术单元单组成[3]. FPGA 如果采用专用加法器和乘法器实现设计，就会减少电路资源和功耗，提高电路的运行速度. 但传统的结构工艺映射算法[2]，一般先把输入网表分解为门级网表再进行映射. 这单元的规则性来实现算术单元支持. 样就会打破模块固有的层次规则性，丢失算术单元的特殊属性，如进位链的处理等，给后续映射实现造成困难[4]. 前人大都基于文献[4－5]提出在高抽象级根据规则知识提取功能模块，来实现对算术单元的支持；文献[3]采用参数模块生成器来实现对算术单元的支持；文献[6]提出从门级网表中提取出专用功能单元，从而实现对算术单元的支持. 上述方法虽然支持对算术单元的映射，但映射后的资源开销一般较大. 为此本文从三个方面来优化算术单元的映射结果：单元共享、平衡算术链以及利用算术单元的空余资源吸收邻近门节点，从而优化了映射结果.

2 术语说明

为了方便对算术单元进行优化操作，本文定义

收稿日期：2012－04－20；修回日期：2012－05－22
基金项目：国家“九七三”重点基础研究发展计划项目(2011CB933202)

算术链的概念. 一个只含算术单元节点的连通图称为算术链. 只含有相同算术单元类型 F 的算术链称为 F 类型算术链，本文对算术单元优化的操作都是基于 F 类型算术链，也可称为同类型算术链.

图 G(V,E)表示有向无环图，V 表示网表中节点集合，E 表示节点之间的连线. 其中节点和线可以为单比特位和多比特位. 基本输入节点 PI 表示图中无扇入节点的集合，基本输出节点 PO 表示图中无扇出节点的集合.

3 映射

FPGA 中逻辑单元具有很好的规则性，所以很容易把带层次规则性的多位算术单元，用 FPGA 单元库中的算术单元来表示. 这种映射过程和逻辑分解相似. 比如：如果 FPGA 库单元中加法器为 2×2 全加法器单元，乘法器为 4×4 的乘法器单元. 映射就是实现把 8×8 加法器用 4 个 2×2 全加器单元来实现. 8×8 乘法器用 4 个 4×4 个乘法器以及 8 个 2×2 全加器单元来实现. 虽然这种简单的分解方法可以实现映射，由于没采用优化方法去除冗余逻辑，一般会增加资源的开销. 前人通过采用常数传递[7]、结合律[8]等方法实现对算术映射资源的优化. 随着用户设计层次越来越高，自动化软件实现映射时需要解决的问题就会变多，如等价功能点、单元输入位宽不同以及算术单元空余资源都影响着映射的结果.

3.1 算术单元共享

一些大型设计中，经常会存在功能等价点. 如果合并这些功能等价点，就可以减少映射结果的资源开销. 常用的方法，就是对全部基本输入节点的所有值，检测任何两个节点的所有输出是否相同，如果相同，则为功能等价点. 但这种搜索方法，为指数复杂度. 逻辑综合中一般利用 SAT 算法和仿真来寻找等价功能点[9]. 对于同类型的操作单元，如果保证输入相同，就可以保证输出相同. 为此本文采用了基于算术单元规则性的检测方法，即检测同一时间帧内所有的算术 F 链之间，或者算术 F 链内部，单元的输入对是否相同. 如果相同，则为功能等价点.

为了保证等价功能点合并后不影响设计的时序功能，需要在检测功能等价点之前，对网表进行时间帧划分. 时间帧保证需要交换的输入必须在设计的同一时间帧内. 如果输入网表是 RTL 级，则以寄存器为时间帧的界点，如果输入网表是行为级，则以时序状态为界点. 划分后，就对时间帧进行编号 1...i...n，相同时间帧赋为相同的编号. 图 1 所示 RTL 级网表时间帧的划分结果，其中虚线表示帧分界线，寄存器之间帧值由驱动寄存其中前驱帧值最大的值决定.

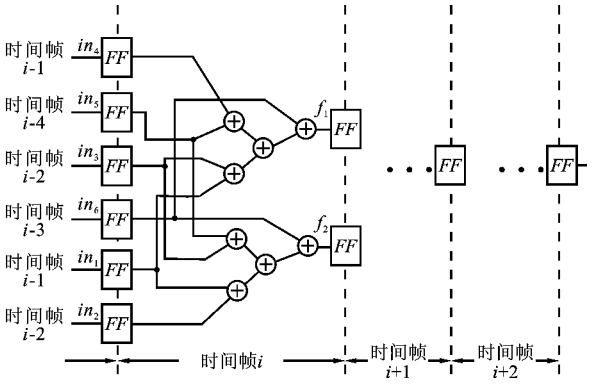


图 1 时间帧的划分

时间帧划分完后，就需要从网表中搜索算术 F 链，其中 F 标明算术操作的类型. 本文在宽度搜索的基础上采用种子生长的方法，来搜索同类型算术链. 如下边伪代码所示，首先对网表采用宽度方法来搜索算术单元，一旦找到算术单元 F，接着以此单元为种子寻找对 F 算术链，搜索时对候选种子进行前生长，直到所有的前驱节点不为 F 类型算术单元，或者节点输入输出位数不匹配为止. 种子生长完成后就会得到其中的一个算术链，然后继续进行宽度搜索，直到候选集合为空为止.

```
FindArithLink (V, ArithLink ){
    V=∅;Push(V,PO);k=-1;
    while (V!=∅)
        v_i=Pop(V);
        if(v_i.lable) continue; endif
        Label(v_i);
        if(isArithm(v_i))
            k++;F= v_i.f; ArithLink
            [k].frame= v_i.frame;
            add(ArithLink [k], v_i);
            SeedGrowth ( v_i, F, Arith-
            Link[k]);
        endif
    endwhile
}
SeedGrowth(v,F,ArithLink){
    U=∅;Push(U,input(v));Push(U,out-
    put(v));
    while (U!=∅)
```

```
        u_i = Pop(U);
        if(u_i.frame != ArithLink.
        frame) continue; endif
        if(u_i.f == F&&tEqual)
                add(ArithLink, u_i); Label
                (u_i); Push(U, input(u_i));
        endif
    endwhile
}
```

经过对网表进行宽度搜索和种子生长，可以得到网表中所有各类型的算术链，接着就可以对同一时间帧内的同类型的算术链进行功能点合并，从而减少资源开销. 功能点合并，类似于逻辑优化中面积优化，但两者还有很大的不同，如算术链子图中每个基本输入位宽为多位，而逻辑优化操作为一位；算术链中的算术操作有累积效应，不像逻辑中的与非操作没有累积效应，加法操作 $a+a=2a$，而逻辑操作 $a+a=a$.

假设时序帧 t 上的算术 F 链共有 m 个，分别为 $f_1, f_2, \cdots f_i \cdots f_m$，基本输入个数为 k，分别为 $in_1, in_2, \cdots, in_j, \cdots in_k$，其中所有输入的位宽相同. 本文用无向图 G(V,E) 来表示时间帧上的所有算术链，其中节点代表单元操作，线表示单元之间的数据连接. $Pair_l(in_j, in_p)$ 表示两个基本输入对，所有输入对的数目为 $L=C_k^2$. f_i 为第 i 个基本输出，$Fout$ 为输出集合，它包含了图的全部或者部分基本输出元素，a_{ij} 为 in_j 对 f_i 的驱动数目，b_{li} 为输入对 in_j 和 in_p 对 f_i 驱动数目，$Drc_l(f)$ 为输输入对 in_j 和 in_p 对 $Fout$ 的驱动数目，$NonDrc_l(f)$ 为输入对 in_j 和 in_p 对 $Fout$ 的无驱动数目.

$$f_i=\sum_{j=1}^{k} a_{ji} in_j \qquad a_{ji}=0,1,2\ldots n \tag{1}$$

$$b_{li}=\begin{cases}0 & \text{if } a_{ji}\geq a_{pi}\\ 1 & \text{others}\end{cases} \tag{2}$$

$$c_{li}=\begin{cases}0 & \text{if } b_{li}\geq 1\\ 1 & \text{if } b_{li}=0\end{cases} \tag{3}$$

$$Drc_l(f)=\sum_{f_i\in Fout} b_{li} \tag{4}$$

$$NonDrc_l(f)=\sum_{f_i\in Fout} c_{li} \tag{5}$$

由于算术链功能点合并和逻辑优化中节点优化一样，为 NP 问题. 所以本文采用了以驱动为优先的启发式算法，即对输出有效驱动数最大的两个输入优先安排算术操作. 具体实现如下：

(1)首先建立同一时间帧内算术链对应基本输入的驱动矩阵和无驱动矩阵，输出集 $Fout$ 为算术链的基本输出集合，即 $f=\bigcup_{i=0}^{k} f_i$. 建立新图 GNew，用来表示合成后的算术链. 候选点集合 V 为空.

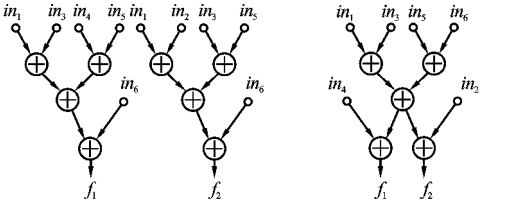
(2)得到对应输出集 $Fout$ 和基本输入的驱动矩阵 a. 接着根据公式(2)得到无效驱动矩阵 c，然后根据公式(3)和(4)可以得到基本输入对输出集 f 的驱动数目和无驱动数目. 如果不存在输入对，则转入(4). 如果存在输入对，则按照下列原则选择输入对：优先 $NonDrc$ 数目最小的输入对，再选择 Drc 数目最大的输入对. 如果选择的 $NonDrc$ 数目为不为 0，则需要把图 G 分割为输入对驱动有效的图 G_d 和驱动无效的图 G_n. 如果 $NonDrc$ 数目为为 0，不需要对图 G 进行分割，图 G 就为输入对驱动有效的图 G_d. 把图 G_d 和 G_n 放入集合 V 中. 转入(3).

(3)从 V 中取出一个待处理的图 G_s，如果 G_s 处为驱动有效图，则转入(4)，否则转入(2).

(4)从图 G_d 中取出一对输入，放入图 GNew 中. 如果图 G 然后把新生成的输出 in_{new} 连入图 G 中. 转入步骤(2).

(5)如果 V 为空，则合并结束，否则从 V 中取出一个候选点继续进行处理，转入(2).

通过上述步骤，就可以得到实现算术单元的共享. 图 2 所示时间帧 t 内有两个独立的加法链，不进行等价功能合并点，则需要 8 个 n 位加法器. 经过等价功能点合并后，则仅需要 5 个 n 位加法器，可以减少 3 个 n 位加法器资源开销.



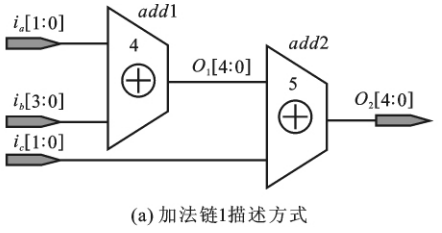
(a) 等价功能点合并前　　(b) 等价功能点合并后

图 2　单元共享

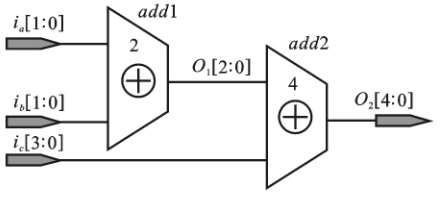
3.2　平衡算术链

随着用户描述语言越来越抽象，算术单元输入位数不相等的情况就会增多. 目前对位宽不相等的输入，都采用高位补零的方法. 这种方法虽然保留了算术单元的层次规则性，但浪费了资源. 例如，三个位宽不同的 i_a、i_b 和 i_c 三个数相加，如图 3(a)和 3(b)加法链 1 和加法链 2 最后结果相同，如采用图 3(a)方式相加，则需要图 3(c)所示 9 个加法单元和 1 个 LE(logic element)来实现，如采用图 3(b)的顺序相加，则仅需要图 3(d)所示 6 个加法器单元和和 2

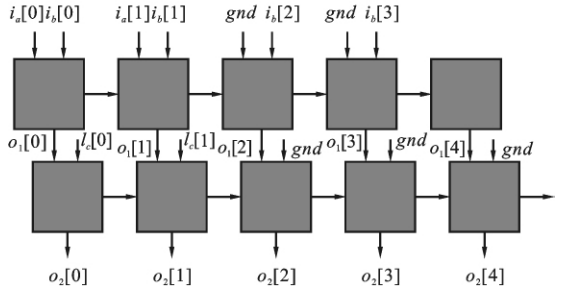
个 LE(logic element)来实现. 由于 FPGA 中加法进位链为专用连线,把它送到 FPGA 逻辑单元的输入端(不是进位输入端),需要耗费一个 LE 的资源.



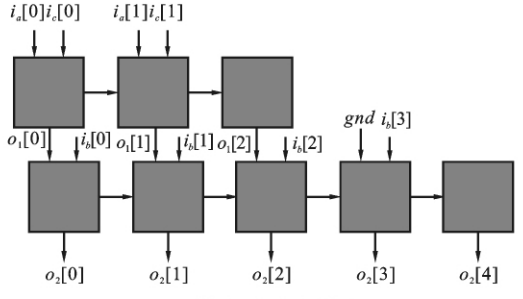
(a) 加法链1描述方式



(b) 加法链2描述方式



(c) 描述1的映射结果



(d) 描述2的映射结果

图 3 不等位宽加法链映射资源比较

从上述分析可知,可以通过改变输入的操作顺序,优化映射资源. 可采用启发式算法解决该问题. 假设该算术链输入为 $in_1, in_2, \cdots, in_j, \cdots in_k$,算术链之间不存在共享算术单元. 本文用位宽映射代价函数 $ArithCost(in_i, in_j)$ 表示两输入形成的算术单元需要的映射资源,由于按照规则性映射,映射代价很容易就可以计算得到. 首先从 k 个输入当中选择位宽映射代价最小的一对输入组成一个算术操作,接着把该算术操作的输出作为一个新的输入添加到剩余的输入集合中,然后再从 $k-1$ 个输入当中选择位宽映射代价最小的一对输入组成一个算术操作,直到输入集合中为空为止.

3.3 算术链的延迟优化

考虑到现代电路对时延要求越高,还需要对算术链中进行延迟优化,在具体时延优化实现中忽略节点之间连线延迟的影响. 本文用 $l(in_i, v)$ 表示输入 in_i 到节点 v 经过算术单元的数目,$ArithDelay(list, v)$ 表示从算术链基本输入到该节点 v 经过算术单元数目最大值,即 $ArithDelay(list, v) = \max_{in_i \in PI(list)} (l(in_i, v). Delay(v)$ 表示节点 v 的内部延迟. 那么算术链基本输入 PI 到节点 v 输出的关键延迟为 $D(list, v) = ArithDelay(list, v) + Delay(v)$. 如图 4(a)和 4(b)所示算术链的延迟 $D(list, v) = 1 + 5 = 6$. $d(list)$ 算术链的关键延迟. 一般来说算术链最后一个节点内部延迟为固定值,它由最终的输出数据位宽决定.

$$d(list) = \min_{v \in PO}(D(list, v)) = \min_{v \in PO}(ArithDelay(list, v) + Delay(v)) = \min_{v \in PO}(ArithDelay(list, v))$$

所以只要保证算术链基本输入到输出节点经过算术单元数目最大值最小,就可以保证该算术链关键路径延迟为最优,从而把算术链延迟优化简化为可以利用标号法进行求解的问题[8].

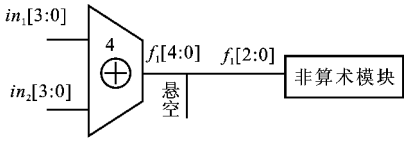
3.4 算术链延迟和资源优化

在实际的算术链中,单元共享和不同位宽可能同时存在. 本文在实现映射时,以延迟优化为先,面积优化为次的目标来优化输入设计中的算术链.

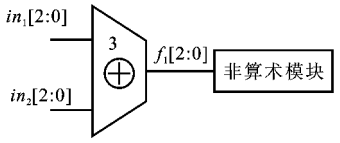
通过分析 3.1~3.3 节中实现算法流程可知,它们采用的启发式算法,都是根据相应优化目标的代价函数寻找最优的输入对. 本文在实现算术链时延和资源优化时,采用类似于 3.1 中单元共享中所采用的算法流程,不过在选择输入对时,不是利用驱动数和无驱动数来选择,而是优先选择经过延迟最小的的输入对,再选择无驱动数最小、驱动数最大以及位宽映射代价最小的输入对,从而实现一个优化的映射结果.

3.5 算术链 IO 的优化

在具体电路实现时,算术单元的输出可能只有部分比特位驱动其他逻辑,而另一部分比特位可能悬空. 如果对具有输出部分比特位的算术单元进行裁减优化,就可以减少映射的资源开销,如图所示,图 4(a)需要 4 位全加器,映射后需要 4 个一位全加器,而经过裁减优化,如图 4(b)所示,需要 3 位全加器,映射后仅需要 3 个一位加法器,从而节省一个加法器的资源开销.



(a) IO优化处理前结果



(b) 优化处理后的结果

图4　算术链IO优化

3.6　空余资源利用

算术单元在实现特定算术功能时，还可能存在一些剩余资源，这些剩余资源可以吸收邻近网表节点，从而增加电路的利用率. 如图5所示，当LUT实现的加法器为常数输入时，就可以利用LUT的查找功能实现常数加法功能，从而使加法器空出空余资源A端，那么就可以吸收后继点f_2，使逻辑单元数目由原来的2个变为1个，逻辑单元级数也从原来的2个变成1个.

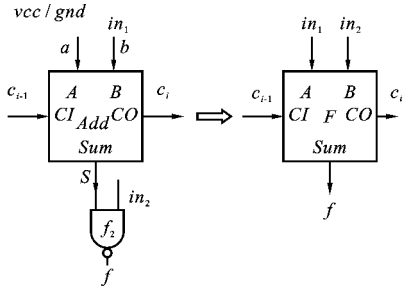


图5　算术单元空余资源的利用

3.7　算法复杂度分析

算术单元共享、平衡算术链以及实现延迟优化的算术链，它们都是基于对输入对的处理，即从候选的输入当中，选择代价函数最大的输入对. 假如阶段i有n个输入，则从n个输入中选择代价函数最大的输入对，就会减少2个输入，但会增加一个新输入(选择的输入对组成单元后的输出)，那么阶段$i+1$还有$n-1$个输入，以此类推，直到处理完为止. 因此算术单元共享、平衡算术链以及实现延迟优化的算术链算法复杂度为$\Theta(n)$.

去除悬浮逻辑的算术链IO优化是对网表中的所有节点进行扫描，对有驱动的信号和节点进行标记，因此算术链IO优化的复杂度为$\Theta(n)$，其中n为节点连线和节点数目的和. 针对空余资源的利用，本质是对每个算术单元进行动态操作，算法复杂度为$\Theta(n)$. 对剩余组合逻辑网表采用基于割的工艺映射算法，复杂度为$\Theta(n)$[3]，因此本文算法复杂度和ABC黑盒子实现工艺映射算法复杂度一样，都为$\Theta(n)$.

4　实验结果

实验平台为Intel E4600 2.4GHz、内存2GB，编程语言为C语言. 为了评估本文算法ArithM(Arithmetic Mapping)的映射性能，实验选取了ODIN中5个信号处理电路[7]. 这些测试集都采用RTL级描述的Verilog，以便于RRMapArithM利用规则性从RTL级来提取出算术单元. ArithM首先从设计网表中提取出算术单元，接着按照3.4算法实现算术单元的延迟和资源优化，其次进行算术链IO优化和分解映射，分析每个算术单元的空余资源，并利用空余资源吸收相邻的门节点，最后再采用基于割的结构映射方法实现网表中其他逻辑单元的映射. 本实验实现了对加法器单元的映射，但该方法还适用于其他算术单元的映射. 库单元中的加法器单元采用Cyclone加法结构[6]，即利用一个LUT和专用进位逻辑实现一位全加器. 由于加法器的专用进位逻辑为硬模块，使用它不需要额外的LUT资源，本实验中加法器面积等于一个LUT的面积.

为了评估ArithM算法的性能，本文选取了ABC工艺映射算法做对比，其中ABC先对算术单元采用黑盒子方式进行处理，然后再采用优先割(Priority Cuts)算法[3]实现映射. 本文以黑盒子算法表示该映射方法.

表1为ArithM和ABC算法的比较结果，其中列1为测试用例，列2和3为ABC工艺映射的面积和延迟结果，列4和5为ArithM算法映射的面积和延迟结果，其中逻辑单元数目为LUT数目，路径延迟为LUT级联级数. 由于黑盒子算法在映射时没有考虑算术单元的延迟，为了求得实际的映射面积和延迟值，需要对ABC映射后的网表进行面积和延迟值进行重新计算. 比较列2和列4可知，通过对算术单元进行优化处理，就可以减少7%的面积开销. 比较列3和列5，可以通过对算术单元进行优化处理，能减少关键路径延迟5%. 对于第3和第5个电路，延迟没有改变，这是因为这些电路中关键路径上没有算术链或者算术链没有进一步的优化空间.

表 1 ArithM 和 ABC 黑盒子算法比较

测试电路	ABC 黑盒子算法		ArithM	
	逻辑单元数目（等效面积）	关键路径延迟	逻辑单元数目（等效面积）	关键路径延迟
cordic_8_8_8	1172	10	924	9
cordic_18_18_18	5653	18	5054	16
fft_256_8	10349	18	10130	18
fir_3_8_8	716	20	714	19
fft_1024_16	54420	47	53020	47
比率	1	1	0.93	0.95

5 结束语

本文提出一种面向算术单元的 FPGA 工艺映射算法 ArithM. 通过利用无驱动数和驱动数代价函数、位宽映射代价函数以及延迟代价函数优化设计网表中的算术链结构，然后分析分解后专用算术单元的空余资源，并利用这些空余资源吸收邻近的门节点，从而减少映射结果的延迟和面积.

参考文献：

[1] Hadi P A, Paolo L. Measuring and reducing the performance gap between embedded and soft multipliers on FPGAS[C]//The 21stInternational conference on Field Programmable Logic and Application. Crete, Greece, 2011: 225-231.

[2] Ye A, Rose J, Lewis D. Synthesizing datapath circuits for FPGAs with emphasis on areaminimization[C]// 2002 IEEE International Conference on Field-Programmable Technolo-gy. Los Alamitos: IEEE Computer Society Press, 2002: 219-226.

[3] Mishchenko A, Cho S M, Chatterjee S, et al. Combinational and sequential mapping with priority cuts[C]// 2007 IEEE/ACM International Conference on Computer-Aided Design. Piscataway: IEEE Press, 2007: 354-361.

[4] Jamieson P, Rose J. A verilog RTL synthesis tool for heterogeneous FPGAs[C]//International conference on Field Programmable Logic and Applications. Los Alamitos: IEEE Computer Society Press, 2005: 305-310.

[5] Jamieson P, Kent K B, Gharibian F, et al. Odin II-an open-source verilog HDL synthesis tool for CAD research[C]//18th IEEE Annual International Symposium on Field-programmable Custom Computing Machines. Washington: IEEE Computer Society Press, 2010: 149-156.

[6] Alastair M S, George A C, Peter Y K, et al. An automated flow for arithmetic component Generation in Field-Programmable Gate Arrays[J]. ACM Transactions on Reconfigurable Technology and Systems, 2008: 1-20.

[7] Chen D M, Cong J, Pan P C. FPGA design automation: a survey[J]. Foundations and Trends Electronic Design Automation, 2006, 1(3): 139-169.

[8] Chen K C, Cong J, Ding Y, et al. Dag-map: graph-based FPGA technology mapping for delayoptimization [J]. IEEE Design & Test of computers, 1992, 9(3): 7-20.

作者简介：

路宝珠 男，博士研究生. 研究方向为大规模集成电路设计自动化技术.

杨海钢 男，博士，研究员，博士生导师. 研究方向为高速可编程逻辑芯片设计技术、数模混合信号 SoC 设计技术.

郝亚男 女，博士研究生. 研究方向为大规模集成电路设计自动化技术.

张茉莉 女，博士研究生. 研究方向为大规模集成电路设计自动化技术.

崔秀海 男，博士，副研究员. 研究方向为大规模集成电路设计自动化技术.