双游程编码的无关位填充算法

方 昊, 姚 博, 宋晓笛, 程 旭
(北京大学理科 1 号楼 1815 室, 北京 100871)

摘 要: 双游程编码是集成电路测试数据压缩的一种重要方法, 可分为无关位填充和游程编码压缩两个步骤. 现有文献大都着重在第二步, 提出了各种不同的编码压缩算法, 但是对于第一步的无关位填充算法都不够重视, 损失了一定的潜在压缩率. 本文首先分析了无关位填充对于测试数据压缩率的重要性, 并提出了一种新颖的双游程编码的无关位填充算法, 可以适用于不同的编码方法, 从而得到更高的测试数据压缩率. 该算法可以与多种双游程编码算法结合使用, 对解码器的硬件结构和芯片实现流程没有任何的影响. 在 ISCAS89 的基准电路的实验表明, 对于主流的双游程编码算法, 结合该无关位填充算法后能提高了 6%~9% 的测试数据压缩率.

关键词: 集成电路测试; 测试数据压缩; 游程编码; 无关位填充
中图分类号: TP391.76 文献标识码: A 文章编号: 0372-2112 (2009) 01-0001-06

The Algorithm of Filling X Bits in Dual-Run-Length Coding

FANG Hao, YAO Bo, SONG Xiao-di, CHENG Xu
(*Room* 1815, *Science Building* 1, *Peking University*, *Beijing* 100871, *China*)

Abstract: The dual-run length codes are the important technique for test data compression. Test compression has two steps: first, the don't care bits in the test data are filled with 0 or 1s and the test data are divided into run sequences; second, every run in the sequences is converted to the compression code according to the given encoding algorithm. However, all the former existing papers focus on the second step, ignoring the importance of the first step thus to lose a certain potential compression ratio. In this paper, we address the importance of don't care bits filling to test data compression ratio and propose a novel algorithm, which fills don't care bits according to the selecting codes to achieve the higher compression ratio. This algorithm can be used with many dual-run length codes without impacting on the decoder structure or the chip implementation flow. For the mainstream dual-run length codes, the compression ratio is improved by 6%~9%.

Key words: IC test; test data compression; run length codes; fill don't care bits

1 引言

产品测试成本的迅速增长已经成为集成电路设计中一个不可忽视的问题. 随着特征尺寸的减小, 一方面, 集成电路包含逻辑门数量的增长使得测试向量增加, 另一方面, 新的故障模型(如延迟故障、桥接故障等)也会增加大量的测试向量. 这两点都使得测试数据容量迅速膨胀, 难以全部存储在自动测试机台有限的内存之中[1].

为了解决测试数据容量巨大的问题, 业界提出了一系列测试数据的压缩算法, 其中一个重要的系列基于具有很高的压缩率和很小的解码器硬件开销的游程编码. 游程编码的测试压缩分为两个步骤: 首先将测试数据中的无关位(X 位)填充为 0 或者 1 并将测试数据分离成游程序列, 然后根据给定的编码算法将序列中的每个游程替换成相应的压缩码. 现有研究大都着重在第二步, 提出了各种不同的编码算法来提高测试压缩率. 文献[2]提出了变长-定长的基本游长编码算法. 随后, 文献[3]提出了变长-变长的 Golomb 编码算法和具有更高压缩率的 FDR 编码算法[4,5]. 接着, 文献[6]提出了 VIHC 编码算法, 对不同长度的游程使用了 Huffman 编码, 达到了更好的压缩效果. 而文献[7]又提出了改进版本的 VIHC 编码——SVIHC 编码. 国内也有学者开展了研究工作, 如文献[8]在 2002 年提出了 variable-tail 编码. 以上所有的游程编码算法都是基于 0 游程的. 除此之外, 文献[9]又提出了基于双游程编码的 EFDR 算法. 该算法同时对 0、1 游程进行编码, 使得游程的数量大大减少, 从而进一步提升了压缩率. 文献[10]提出了同样对 0、1 游程进行编码的交替式编码 AFDR 算法. 国内也于最近提出了各自的算法[11,12]. 另外, 一些基于游程编码

收稿日期: 2008-03-05; 修回日期: 2008-10-28
基金项目: 国家 863 高技术研究发展计划(No. 2006AA010202)

来重新配置扫描链的算法在布线资源允许的情况下进一步提高了压缩率[13,14].

针对测试数据的编码方法的压缩率很大程度取决于待编码数据的特征. 通常一个测试向量中包含确定位和无关位(*X* 位),无关位的值并不影响故障覆盖,因此在测试压缩中可以被利用,赋对压缩有利的数值,提高测试压缩率. 最近的一些针对商用处理器的统计数据表明,大规模芯片的测试数据中 *X* 位比例很高[15,16],可以达到 95% 以上. 因此如何对 *X* 位进行填充赋值,对测试数据压缩率影响很大. 本文致力于研究无关位填充方法,分析无关位填充对于测试数据压缩率的重要性,并提出一种新颖的双游程编码的无关位填充算法,可以根据不同的编码方法来使用不同的无关位填充策略,从而得到更高的测试数据压缩率. 该算法可以与多种双游程编码算法结合使用,对解码器的硬件结构和芯片实现流程没有任何的影响. 在 ISCAS89 的基准电路的实验表明,对于主流的双游程编码算法,结合该无关位填充算法后能提高了 6%~9% 的测试数据压缩率.

2 游程编码技术介绍

传统的双游程编码过程往往由两个步骤组成. 第一步,把测试数据中的无关位 *X* 填充为 0 或者 1,此时测试数据已经被分离成游程序列;第二步,对于得到的游程序列,依次对每个游程进行编码. 基于双游程的编码技术需要将测试数据分离并识别为 0 游程和 1 游程. 所谓 0(1)游程,是指一个仅包含 0 和 1 的序列中,除了最后一位是 1(0)之外,其它位都是 0(1),其中 0(1)的个数称为该游程的长度. 如 0001 是长度为 3 的 0 游程,110 为长度为 2 的 1 游程. 传统的游长编码(Runr length codes)、Golomb 编码、FDR 编码、VIHC 编码等等都是仅仅基于 0 游程的. 而 EFDR 编码和 AFDR 编码则是基于双游程的. 由于同时对 0、1 游程进行编码,EFDR 编码和 AFDR 编码的游程总数要小于仅仅基于 0 游程的编码,从而大大提高了数据压缩率. EFDR 的部分编码表在表 1 给出. 由于 EFDR 编码可以作为双游程编码的代表性编码,其所得的结论也适用于其它双游程编码,因此,本文的理论部分仅以 EFDR 编码为例.

表 1 EFDR 编码表

组	游程长度	编码组成		完整编码	
		组前缀	后缀	0 游程编码	1 游程编码
1	1	0	0	000	100
	2		1	001	101
2	3	10	00	01000	11000
	4		01	01001	11001
	5		10	01010	11010
	6		11	01011	11011

测试向量中存在着大量的无关位(*X* 位),这是能够取得高压缩率的重要前提. 在压缩编码过程中,无关位的值并不影响故障覆盖,因此在测试压缩中可以被利用,赋对压缩有利的数值,尽可能的提高测试压缩率. 对于仅有 0 游程的编码,*X* 位只需简单的全部填充成 0 就可以了,因为这样可以保证游程总数最少,从而往往使得压缩率变高. 在 EFDR 的编码中,*X* 位是按照所处的 *X* 串(连续的 *X* 位)两边的确定位的值来填充的. 仅当 *X* 串的左右边界确定位都为 1 时,该 *X* 串才全被填充为 1,其余情况则都填充为 0. 表 2 给出了 EFDR 在 *X* 串各种左右边界值情况下的填充方式,其核心思想都在于使得游程总数尽量少. 整个 *X* 串都被填充成相同的值,并且保持至少和一个边界确定值相等,使得整个 *X* 串能够和左边界的确定位或者右边界的确定位处于同一个游程序列中,从而尽量不增加新的游程. 这个填充思想和 0 游程的 *X* 填充思想完全一致,都是为了保证游程总数尽量的少. 然而,从下一章可以看到,这种填充方法并不是最优的,它们都没有充分考虑到对后续编码的影响,从而损失了一定的潜在压缩率.

表 2 EFDR 的 X 填充方式和填充结果

左边界	右边界	填充方式	举例	填充结果
0	0	全为 0	0*XXX*0	00000
1	1	全为 1	1*XXX*1	11111
0	1	全为 0	0*XXX*1	00001
1	0	全为 0	1*XXX*0	10000

3 双游程编码的无关位填充算法

3.1 动机

EFDR 并没有对 *X* 的填充引起足够的重视,所给的 *X* 填充方法在很多情况下都不能够取得最优的压缩效果. 例如,测试数据 T_{D1}= 111*XXXX*00001,其中包含长度为 4 的 *X* 串. 如果按照 EFDR 编码,如表 3 的第二行所示,根据 EFDR 给出的填充方法,由于该 *X* 串两边的边界分别为 1 和 0,因此该 *X* 串中的 4 个 *X* 都将被填充为 0,测试数据将变成 1110 00000001,形成 1 个长度为 3 的 1 游程和 1 个长度为 7 的 0 游程,对照 EFDR 编码表表 1 可知,编码后长度为 12. 而如果按表 3 的第三行所示,把这个 *X* 串填充成 1110,则测试数据将变成 111110 000001,形成 1 个长度为 6 的 1 游程和 1 个长度为 4 的 0 游程,编码后长度为 10,比 EFDR 的填充方式节省了 2 位. 从这个例子中可以看出:

表 3 对测试数据 T_{D1}= 111*XXXX*00001 进行 EFDR 填充后压缩结果比较

填充方式	填充结果	EFDR 编码	压缩后位数
EFDR 填充	1110 00000001	11000 01 10000	12
最优填充	111110 00001	11011 01001	10

(1)即使对于相同的测试数据, 并使用相同的编码方案, X 位的不同填充算法仍将会导致不同的游程序列, 从而得到不同的压缩率.

(2) EFDR 的 X 填充算法并不能获得最优的压缩率. 以上两点发现正是本文的动机所在, 显然我们可以使用其它的 X 填充方法来形成不同的游程序列, 从而获得更高的压缩率.

3.2　带余量的游程识别

对于左右边界值相同的 X 串(0、0 和 1、1 的情况), EFDR 的填充方式毫无争议(表 2 第 3、4 行), 因为这样往往使得整个 X 串和左右的边界值处于同一个游程中, 从而不会增加总的游程数量. 而对于左右边界值相异的 X 串(01 和 10 的情况), EFDR 的填充方式也有一定的道理, 它使得整个 X 串和至少一个边界值处于同一个游程中, 从而也不会增加总的游程数量. 可见, 在填充 X 串的时候, 不增加总的游程数量是一个基本的原则. 这点并不难理解, 因为游程总数越少, 则意味着游程平均长度越大, 从编码表表 1 可以发现, 游程长度越大, 则压缩率越高. 因此, 尽量不增加总的游程数量, 则可以保证压缩率不会降低太多. 然而, 在不增加总的游程数量的填充方法中, 还存在着比 EFDR 更优秀的填充方案. 仔细观察表 3 的例子, 可以发现, 对 X 的不同填充方式形成了不同的压缩结果. EFDR 填充和最优填充这两种 X 的填充方式都形成了两个游程序列, 前者的两个游程序列长度为 3 和 7, 后者的两个游程序列长度为 6 和 4. 虽然两者的游程序列的总长度都是 10, 但是由于游程长度 ——编码长度函数为离散递增函数, 对于长度为 3~ 6 的游程, 编码后长度都为 5; 而对于长度为 7 的游程, 编码后长度则为 7. 虽然在第一个游程中, EFDR 填充的游程长度 3 比最优填充的游程长度 6 小 3, 但是编码后的长度是一样的. 而对于第二个游程, EFDR 填充的游程长度 7 比最优填充的游程长度 4 大 3, 但编码后的长度却大了 2 位, 从而整个测试数据在编码后的长度就大了 2 位.

从这个分析, 我们可以看出, 对于左右边界值相异(0、1 或 1、0) 的 X 串, 该 X 串的填充方式可能会直接影响到前后的两个游程的长度, 进而影响到整体的压缩率. 为了使得整体的压缩率达到最优, 必须合理的调整此类 X 串前后游程的长度.

之所以说"可能" 会影响, 而不是一定会影响, 是因为在不增加不必要的总游程数量的前提下, 有些左右边界值相异的 X 串的填充方式是固定的(这点我们会在稍后讲述). 因此, 我们先识别出会影响前后游程长度的 X 串, 对于这种 X 串, 保留前后游程的长度变化范围, 而每个游程的长度的具体确定, 由随后选择的编码算法来决定, 这样就能使得总编码长度最短.

我们使用了余量的概念来描述每个游程的长度可变范围. 每两个邻接的游程之间都存在着一个余量值, 定义如下:

余量: 对于一个 X 串, 当此 X 串能够在不增加额外的游程总数的前提下, 有多种填充方式, 使得其邻接的两个游程的长度可以在一定范围内变化. 游程长度的可变化范围定义为余量. 对于一个给定的游程序列 $R = \{r_1, r_2, \cdots, r_n\}$, 余量 m_i 定义为游程 r_i 末尾处和 r_{i+1} 起始处需要额外增加的长度总和. 余量对应的 X 串称为余量串, X 串成为余量串当且仅当意味着该 X 串的填充会影响前后两个游程的长度.

如测试数据 T_{D2}= 11*XXXXXX* 0000*XXXXX* 1111100001, 可以看成 4 个游程 r_1, r_2, r_3, r_4 组成的游程序列, 我们用 $r = (a, b)$ 来表示类型为 a 且最小游程长度为 b 的游程 r. 则 $r_1 = (1, 2)$, $r_2 = (0, 3)$, $r_3 = (1, 4)$, $r_4 = (0, 3)$ 表示有 4 个类型分别为 1, 0, 1, 0 的游程, 它们的游程长度至少为 2, 3, 4, 3. 我们用余量 m_i 表示游程 r_i 末尾处和 r_{i+1} 起始处还需要额外增加的长度, 如在 T_{D2} 中 $m_1 = 6$, 表示游程 r_1 和 r_2 还需要从第一个 X 串共增加 6 的长度. 同理, $m_2 = 5$. 一般而言, 余量值正好为 X 串的长度. 特别的, 当相邻的游程中不含有 X 串的时候, 我们可以认为它们之间的余量为 0, 如 T_{D2} 中 $m_3 = 0$, 表示游程 r_3 与 r_4 之间不存在游程. 为了描述方便, 最后一个 m_i 永远定义为 0, 如在 T_{D2} 中 $m_4 = 0$. 我们用游程类型序列 $T = \{t_i\}$, 游程最小长度序列 $S = \{s_i\}$, 余量序列 $M = \{m_i\}$ 来描述上述的测试数据, 对于 T_{D2}, $T = \{1, 0, 1, 0\}$, $S = \{2, 3, 4, 3\}$, $M = \{6, 5, 0, 0\}$.

经过观察, X 串能够成为余量串必须满足以下条件:

(1) X 串两边的边界值不同;

(2) 左边界值不为上一个游程的末尾;

(3) 右边界值不为当前游程的末尾;

(4) 相邻的游程类型不同.

根据这些条件, 我们就可以写出识别游程余量的算法, 由于算法比较简单, 此处不再赘述.

3.3　余量分配算法

在使用游程余量识别算法后, 测试数据中所有不在余量串中的 X 位已经直接被填充. 对于剩下处于余量串中的 X 位, 我们将使用余量分配算法来填充. 余量的分配方案决定了余量串中 X 位的填充方式. 从前面的例子可以看出, 每个游程都可能受到前后两个 X 串的影响, 如测试数据 11*XXXXXX* 0000*XXXXXX* 111110, 填充后可以被切分为三个游程. 如 11 1111000000 001 111 111110. 可以看到, 中间的 0 游程的长度同时受到前后两个 X 的填充策略的影响. 在最复杂的情况下, 整个测

试向量中, 除了头尾两个游程之外, 每个游程的长度都会受到前后两个游程的影响, 我们需要全局考虑如何填充 X 来分配余量, 使得整体测试数据的压缩率最高. 如前所述, 在使用余量识别算法得到所有游程的类型序列 T、基本长度序列 S 和相应的游程余量序列 M 后, 我们希望把每一个游程余量, 合理的分配给前后游程来增加它们的长度, 从而得到选择最佳数据压缩率. 该问题的数学描述如下:

余量分配问题: 给定 3 个长度为 n 的非负整数序列 $T=\{t_i\}$, $S=\{s_i\}$, $M=\{m_i\}$, $(1\leqslant i\leqslant n)$, 其中 $m_n=0$. 对于长度为 n 非负整数数列 $A=\{a_i\}$, 其中 $0\leqslant a_i\leqslant m_i$, 令 $b_i=m_i-a_i$, $b_{-1}=0$, $w_i=s_i+b_{i-1}+a_i$, 给定函数 $f(i)=f(t_i,w_i)$, 总代价函数为:

$$Y(A)=\sum_{i=1}^{n}f(i)=\sum_{i=1}^{n}f(t_i,w_i)$$

问: 求非负整数序列 A, 使得总代价函数 $Y(A)$ 最小?

在该问题中, $T=\{t_i\}$ 为游程类型序列, $S=\{s_i\}$ 为游程最小长度序列, $M=\{m_i\}$ 为余量序列. 需要对每个 m_i 进行切分 $m_i=a_i+b_i$. 若从余量分配的角度看, 余量 m_i 分配长度增量 a_i 给第 i 个游程, 分配长度增量 b_i 给第 $i+1$ 个游程; 若从游程的角度看, 第 i 个游程将从余量 m_i 得到长度增量 a_i, 从余量 m_{i-1} 得到长度增量 $b_{i-1}=m_{i-1}-a_{i-1}$. w_i 表示第 i 个游程的最终游程长度, $f(t_i,w_i)$ 为相应的编码长度函数, 它表示类型为 t_i 长度为 w_i 的游程的编码长度. $Y(A)$ 表示了所有的游程编码后的总长度. 在 $T_{D2}=11\underline{XXXXXX}0000\underline{XXXXX}11111100001$ 的例子中, $T=\{1,0,1,0\}$, $S=\{2,3,4,3\}$, $M=\{6,5,0,0\}$. 采用 EFDR 编码的 $f(t_i,w_i)=1+2\lceil\log_2(w_i+1)\rceil$, 其中 $\lceil x\rceil$ 表示 x 的上取整, 即不小于 x 的最小整数. 若将 T_{D2} 填充为 $11\underline{11}1000000000\underline{1111}11111100001$, 无关位填充方案对应的 $\{a_i\}=\{3,1,0,0\}$, $\{b_i\}=\{3,4,0,0\}$ 填充后的游程长度序列为 $S'=\{2+0+3,3+3+1,4+4+0,3+0+0\}=\{5,7,8,3\}$

由于每个游程的长度均可能会受前后两个余量分配的影响, 仅仅单独考虑一个余量的分配是无法得到最优解的. 本节给出一种通用的的余量分配算法, 对于编码长度仅仅与游程类型和长度相关的编码函数 $f(t_i,w_i)$, 可以得到最佳的压缩效果. 当编码函数满足该约束条件时, 余量分配过程是一个多步决策过程, 且满足无后效性原则, 我们可以使用动态规划算法来得到余量分配的最优解.

对于某特定的余量分配问题, 给定问题中的各种参数, n, $\{m_i\}$, $\{s_i\}$, $\{t_i\}$, 我们可以这样定义该问题的子问题 $Q(h,j)$: 在余量分配问题中, 序列 $\{m_i'\}$, $\{s_i'\}$, $\{t_i'\}$ 都截取为长度为 h 的序列, 其中序列中的元素除了 $s_h'=s_h+j$ 和 $m_h'=0$, 均取原序列 $\{m_i\}$, $\{s_i\}$, $\{t_i\}$ 的前 h 项. 特别的, 原问题也可以看作一个特殊的子问题: $Q(n,0)$. 我们再用 $y(h,j)$ 表示子问题 $Q(h,j)$ 对应的最优解, 即总最短编码长度. 若 $Q(h,j)$ 存在最优解 $y(h,j)$, 且该最优解从 $Q(h-1,k)$ 的某个解得到, 则 $Q(h-1,k)$ 所对应的那个解必然也是 $Q(h-1,k)$ 中最优的, 即 $y(h-1,k)$. 为了得到获得最优解所对应的切分点 k, 我们需要依次检测每个可能的 k 值来得到最优解, 即 $y(i,j)\leftarrow\min_{0\leqslant k\leqslant m_{i-1}}\{y(i-1,k)+f(t_i,s_i+j+m_{i-1}-k)\}$. 最终 $Y(A)$ 的最优解将由 $Q(n,0)$ 对于的最优解 $y(n,0)$ 给出. 下面给出了相应的动态规划伪代码.

输入: 游程类型序列 $T=\{t_i\}$, 游程最小长度序列 $S=\{s_i\}$, 余量序列 $M=\{m_i\}$

输出: 最短编码长度 Y 和相应的余量分配序列 $A=\{a_i\}$

```
1.  function MarginAssignAlgorithm(T_D)
2.  for i←0 to m_1      //设定初始边界条件
3.    y(1, i)←f(t_1, s_1+ i)
4.  for i←2 to n // 对于后续的 n- 1 步决策
5.      forj ←0to m_i//对于当前游程的每一个后续选择
6.        y(i,j) ←min_{0≤k≤m_{i-1}} {y(i- 1, k)+ f(t_i, s_i+ j
          + m_{i-1}- k)}
7.        p(i,j) ←min k // 记录最佳划分点
8.  Y←y(n, 0)
9.  //计算最佳切分点
10.   a_n ←0
11.   for i ←n downto 2
12.     a_{i-1} ←p(i, a_i)
13  return Y, {a_i}
```

我们来分析一下该余量分配算法的时间复杂度, 核心部分第 4-7 行有三重循环, 时间复杂度为:

$$Z=\sum_{i=2}^{n}(m_{i-1}+1)(m_i+1)$$

$$\leqslant\sum_{i=2}^{n}(m_{i-1}+1)(m_{\max}+1)\leqslant N(m_{\max}+1)\qquad(1)$$

$T_D=0XXX11XX000XXX11110$
游程最短长度序列 $\{s_i\}=(1,1,2,3)$
游程余量序列 $\{m_i\}=(3,2,3,0)$
计算中间过程: $y(i,j)/p(i,j)$ (最短编码长度/切分点)

i/j	0	1	2	3
1	3/-	3/-	5/-	-/-
2	8/0	8/0	8/0	-/-
3	11/2	13/0	13/0	13/1
4	16/0	-/-	-/-	-/-

计算结果
$Y=16$ $A=\{a_i\}=(0,2,0,0)$
填充后测试数据 $T_D'=01\underline{11}11\underline{1}000\underline{1}\underline{11}11110$
编码 $T_E=000110110011011011$

图 1 余量分配的动态规划算法演示示例

其中 $m_{max}=\max\{m_i\}$, 而 $\sum_{i=2}^{n} m_{i-1}$ 为 T_D 中余量串的 X 数量, 由于每个余量串都处于确定位之间, 因此 $\sum_{i=2}^{n}(m_{i-1}+1)\leqslant N$, N 为 T_D 的大小. 设 C 为测试电路中的扫描单元数量, 也是每一条测试向量的长度, P 为测试向量数量, 则 $N=CP$. 由于每一条测试向量至少要测出一个故障, 每一条测试向量中不可能全部为 X, 换言之, 每一条测试向量中的 X 串长度至多为 $C-1$. 考虑到 X 串可能跨越测试向量, 则在 T_D 中 X 串的长度至多 $2C-2$. 根据式(1), 可知最坏时间复杂度为 $O(N(2C-2))=O(NC)$. 实际上, 并非所有的 X 串都会成为余量串, 且 X 串的分布随着长度的增加迅速减少, 因此实际的算法运行速度非常快. 图 1 给出了一个实际计算的例子.

4　实验结果和分析

我们用 Java 语言实现了上述算法, 并且在 ISCAS-89 的基准电路上评测了性能. 实验环境为 2.8G 的奔腾 4 处理器和 512M 内存. 实验电路选取了 ISCAS-89 中 7 个最大的基准电路. 它们相应的测试向量由 MinTest 生成[15].

表 4　基准电路的测试向量

基准电路	$N=\|T_D\|$	C	P	X 比例(%)	EFDR[9]		余量选择+ EFDR			AFDR[10]		余量选择+ AFDR		
					T_{E1}	R_1	T_{E2}	R_2	改进(%)	T_{E1}	R_1	T_{E2}	R_2	改进(%)
s5378f	23,754	214	111	72.62	11,419	51.93	10,806	54.51	5.37	11,988	49.53	11,068	53.41	7.67
s9234f	39,273	247	159	73.01	21,250	45.89	19,420	50.55	8.61	21,704	44.74	19,332	50.78	10.93
S13207f	165,200	700	236	93.15	29,995	81.84	27,924	83.10	6.90	30,530	81.52	27,934	83.09	8.50
S15850f	76,986	611	126	83.56	24,643	67.99	23,345	69.68	5.27	25,766	66.53	23,464	69.52	8.93
S35932f	28,208	1,763	16	35.30	5,554	80.31	5,347	81.04	3.73	5,610	80.11	5,324	81.13	5.10
S38417f	164,736	1,664	99	68.08	64,962	60.57	60,108	63.51	7.47	65,988	59.94	59,914	63.63	9.20
S38584f	199,104	1,464	136	82.28	73,853	62.91	68,559	65.57	7.17	75,710	61.97	68,564	65.56	9.44
平均值	–	–	–	–	–	–	–	–	6.36	–	–	–	–	8.54

基准电路的测试向量特征如表 4 所示. 其中第一列给出了基准电路的名称; 第 2-4 列分别给出了压缩前的测试数据大小, 扫描单元的数量 C, 测试向量数量 P 和测试向量中 X 的比例. 后几列比较了原始的 EFDR 和 AFDR 编码压缩率和使用了无关位填充算法后的 EFDR 和 AFDR 编码压缩率. 所有测试电路的运行时间均在 1 秒以内. 第 6、7 列和 11、12 列分别列举了用 EFDR 填充后的和 AFDR 填充后的编码大小 T_{E1}、压缩率 R_1(相对于原始数据大小的压缩百分比); 第 8-10 列, 13-15 列分别给出了和使用本文的余量选择算法后的 EFDR 编码和 AFDR 编码大小 T_{E2}、压缩率 R_2 和相对的改进百分比 I, 即改变填充方式后编码后的大小的改进比例. 平均而言, EFDR 改进 6.36%, AFDR 改进 8.54%.

5　总结

本文提出了一种基于 0、1 游程的识别算法, 通过针对当前选择的编码方法来填充无关位, 来调整相邻游程的长度, 从而提高了数据压缩率. 该算法可以无缝的嵌入到多种基于 0、1 游程编码算法中结合使用. 由于编码方法没有改变, 因此相应的片上解码器, 芯片实现流程都不需要做出任何的变化. 实验表明, 在仅仅通过调整软件压缩算法, 加入了少量的程序代码后, 就可以提高 6%-9% 的测试数据压缩率.

参考文献:

[1] K M Butler. Estimating the economic benefits of DFT[J]. Design & Test of Computers, IEEE, 1999, 16(1): 71-79.

[2] A Jas, N A Touba. Test vector decompression via cyclical scan chains and its application to testing core-based designs[A]. International Test Conference[C]. Washington, DC, USA: IEEE Computer Society, 1998. 458-464.

[3] A Chandra, K Chakrabarty. System-on-a-chip test-data compression and decompression architectures based on Golomb codes[J]. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 2001, 20(3): 355-368.

[4] A Chandra, K Chakrabarty. Frequency-directed run-length (FDR) codes with application to system-on-a-chip test data compression[A]. VLSI Test Symposium[C]. Washington, DC, USA: IEEE Computer Society, 2001. 42-47.

[5] A Chandra, K Chakrabarty. Test data compression and test resource partitioning for system-on-a-chip using frequency-directed run-length (FDR) codes[J]. Computers, IEEE Transactions on, 2003, 52(8): 1076-1088.

[6] P T Gonciari, B M Al-Hashimi, N Nicolici. Improving compression ratio, area overhead, and test application time for system-on-a-chip test data compression/decompression[A]. Design, Automation and Test in Europe Conference and Exhibition[C]. Washington, DC, USA: IEEE Computer Society, 2002. 604-611.

[7] C Giri, B M Rao, S Chattopadhyay. Test data compression by Split-VIHC (SVIHC)[A]. International Conference on Computing: Theory and Applications[C]. Washington, DC, USA: IEEE Computer Society, 2007. 146-150.

[8] 韩银和, 李晓维, 徐勇军, 李华伟. 应用 Variable-Tail 编码

压缩的测试资源划分方法[J]. 电子学报, 2004, 32(8): 1346- 1350.
Han Yinhe, Li Xiaowei, Xu Yongjun, Li Huawei. Test resource partitioning using variable-tail code[J]. Acta Electronica Sinica, 2004, 32(8): 1346-1350(in Chinese).

[9] A H El-Maleh, R H Al-Abaji. Extended frequency-directed runlength code with improved application to system-on-a-chip test data compression[A]. International Conference on Electronics, Circuits and Systems[C]. Washington, DC, USA: IEEE Computer Society, 2002. 449- 452.

[10] A Chandra, K Chakrabarty. Reduction of SOC test data volume, scan power and testing time using alternating runlength codes[A]. Design Automation Conference[C]. Washington, DC, USA: IEEE Computer Society, 2002. 673- 678.

[11] 梁华国, 蒋翠云. 基于交替与连续长度码的有效测试数据压缩和解压[J]. 计算机学报, 2004, 27(4): 548- 553.
Liang Huaguo, Jiang Cuiyun. Efficient test data compression and decompression based on alternation and run length codes[J]. Chinese Journal of Computers, 2004, 27(4): 548- 553(in Chinese).

[12] 彭喜元, 俞洋. 基于变游程编码的测试数据压缩算法[J]. 电子学报, 2007, 35(2): 197- 201.
PENG Xiyuan, YU Yang. A test set compression algorithm based on variable-runlength code[J]. Acta Electronica Sinica, 2007, 35(2): 197-201(in Chinese).

[13] Y Shi, et al. Alternative runlength coding through scan chain reconfiguration for joint minimization of test data volume and power consumption in scan test[A]. Asian Test Symposium[C]. Washington, DC, USA: IEEE Computer Society, 2004. 432- 437.

[14] Fang Hao, Tong Cherguang, Cheng Xu. Run based reordering: a novel approach for test data compression and scan power[A]. Asia and South Pacific Conference on Design Automation[C]. Piscataway, NJ, USA: IEEE Press, 2007. 732- 737.

[15] I Hamzaoglu, J H Patel. Test set compaction algorithms for combinational circuits[J]. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 2000, 19(8): 957- 963.

[16] Han Yinhe, Hu Yu, Li Xiaowei, et al. Embedded test decompressor to reduce the required channels and vector memory of tester for complex processor circuit[J]. IEEE Transactions on VLSI System, 2007, 15(5): 531- 540.

作者简介:

方 昊 男, 1981 年出生于浙江杭州, 北京大学计算机系博士研究生, 研究方向为数字电路的可测性设计、时序优化和分析、时钟树调度和生成.
E-mail: fanghao@mprc.pku.edu.cn

姚 博 男, 1983 年出生于河南南阳, 北京大学计算机系硕士研究生, 研究方向为数字电路的可测性设计.
E-mail: yaobo@mprc.pku.edu.cn

宋晓笛 男, 1979 年出生于北京, 英国爱丁堡大学硕士, 长期从事数字信号处理算法及相关超大规模集成电路设计研究. 曾任"十五"八六三项目课题负责人, 并参与过其他多项国家八六三项目. 主要研究方向为软硬件协同设计、数字信号处理和多核系统芯片.
E-mail: songxiaodi@pku.edu.cn

程 旭 男, 1967 年出生于湖北襄樊, 教授, 博士生导师. 北京大学微处理器研究开发中心主任、计算机科学技术系主任、计算机系统结构研究所所长、国家"十五"八六三计划超大规模集成电路设计专项专家组成员、中国通信学会专用集成电路委员会副主任委员. 主要研究方向为高性能微处理器、系统芯片、嵌入式系统、指令级并行、优化编译、软硬件协同设计等.
E-mail: chengxu@mprc.pku.edu.cn