# 应用混合游程编码的 SOC 测试数据压缩方法

方建平, 郝 跃, 刘红侠, 李 康
(西安电子科技大学微电子学院宽禁带半导体材料与器件重点实验室, 陕西西安 710071)

摘 要: 本文提出了一种有效的基于游程编码的测试数据压缩/解压缩的算法: 混合游程编码, 它具有压缩率高和相应解码电路硬件开销小的突出特点. 另外, 由于编码算法的压缩率和测试数据中不确定位的填充策略有很大的关系, 所以为了进一步提高测试压缩编码效率, 本文还提出一种不确定位的迭代排序填充算法. 理论分析和对部分 ISCAS 89 benchmark 电路的实验结果证明了混合游程编码和迭代排序填充算法的有效性.

关键词: 测试数据压缩; 不确定位填充; system on a chip (SOC) 测试; 混合游程编码
中图分类号: TP391.76 文献标识码: A 文章编号: 0372-2112 (2005) 11-1973-05

# A Hybrid Run-Length Coding for Soc Test Data Compression

FANG Jian-ping, HAO Yue, LIU Hong-xia, LI Kang
*(Key lab of ministry of education for wide Band-Gap Semiconductor Materials and Devices School of Microelectronic, Xidian University, Xi'an, Shaanxi 710071, China)*

**Abstract:** This paper presents a compression and decompression scheme based on run-length codes for reducing the amount of test data. We refer to this hybrid run-length codes, it has the excellent advantages of high compression ratios, robust and low area overhead. Since the compression ratios strongly depend on the strategy of mapping which doesn't care in the original test set to zeros or ones, we also present an Iterative Sort Filling algorithm to find the best assignment method which minimizes the total size of the test data and achieves higher compression ratio. The theoretical analysis and the experimental results for ISCAS 89 benchmark circuits demonstrated the compression efficiency of the proposed hybrid run-length codes and Iterative Sort Filling algorithm.

**Key words:** test data compression; don't care assignment; system on a chip test; hybrid run-length codes

## 1 引言

随着 SOC (system-on-a chip) 芯片规模的不断增加, 以及功能复杂度的不断提高, SOC 芯片的测试成本和难度直线上升. 而直接影响 SOC 测试成本的根本原因是测试数据容量过于庞大. 目前解决的主要方法有: 压缩测试数据和内建自测试法 (BIST: Build in self test). BIST 很好的解决了芯片测试成本的问题, 但是 BIST 存在硬件开销大、具有难测故障、以及对逻辑电路的测试覆盖率低等缺点. 所以, 目前人们常常通过测试资源划分和测试数据压缩的方法来减小 SOC 芯片的测试数据容量和测试成本[1~9]. 压缩测试数据的方法具有实现简单, 效果明显等特点. 目前已经提出的压缩编码算法有: 统计编码[1]、Golomb 编码法[2]、FDR 编码法[3]、选择哈夫曼编码 (Selective Huffman Coding)[5] 和交替游程编码 (Alternating Run-Length)[9] 等. 这些算法有一个共同缺点, 就是无法解决压缩率和硬件开销之间的矛盾. 为了弥补和克服这个问题, 本文提出了混合游程编码, 即 Hybrid Run Length coding, 它具有很高的压缩率, 同时硬件开销很小. 并且为了进一步提高压缩率, 本文还提出了一种合理的不确定位的填充方法和向量排序算法, 实现对测试数据中不确定位的动态填充以及对测试数据中各测试向量的合理排序, 使测试数据压缩效果达到最佳. 文章安排如下: 第二部分, 引出编码算法, 理论分析了压缩算法的有效性; 第三部分介绍迭代排序填充算法对测试数据的优化处理; 第四部分是相应的解码电路实现; 第五部分给出具体实验数据.

## 2 混合游程编码算法

### 2.1 混合游程编码

混合游程编码是一种不等间距的变长-变长的编码方式. 所谓不等间距编码方式是指在进行分组时每个组的大小根据其出现频率进行了适当的调整. 表 1 是后缀位宽 $L_t = 1$ 时混合游程算法的编码表, 可以看到从 $A1$ 到 $A3$, 每个分组的容量是依次递增的, 这样的分组更符合实际数据中游程的分布[2,3]. 同时为了使混合游程编码算法相应的解码电路更加简单, 定义了后缀标识位, 通常取常数"0", 这样在解码过程中可以通过标识位来识别当前送入的位串是前缀还是后缀部

收稿日期: 2005-04-11; 修回日期: 2005-07-14
基金项目: 国家高技术研究发展计划 (No. 2003AA1Z1630); 自然科学基金 (No. 60206006)

分; 另外, 前缀的最小组成元素的最高位必须与后缀标识位相反, 即取常数"1", 而且前缀的组成元素的位宽应该满足: $L_p = L_t + 1$. 所以, 混合游程编码算法的分组策略会更加灵活, 可以通过取不同的 $L_t$ 值来改变分组的规模, 提高压缩率.

表 1 混合游程编码表($L_t = 1$)

| 游程长度 | 组号 | 前缀 (元素位宽= 2) | 后缀 | | 代码字 |
|---|---|---|---|---|---|
| | | | 标识 | 后缀码 $L_t$ | |
| 0 | $A1$ | 无 | 0 | 0 | 00 |
| 1 | | 无 | 0 | 1 | 01 |
| 2 | $A2$ | 10 | 0 | 0 | 1000 |
| 3 | | 10 | 0 | 1 | 1001 |
| 4 | | 11 | 0 | 0 | 1100 |
| 5 | | 11 | 0 | 1 | 1101 |
| 6 | $A3$ | 1010 | 0 | 0 | 101000 |
| 7 | | 1010 | 0 | 1 | 101001 |
| 8 | | 1011 | 0 | 0 | 101100 |
| 9 | | 1011 | 0 | 1 | 101101 |
| 10 | | 1110 | 0 | 0 | 111000 |
| 11 | | 1110 | 0 | 1 | 111001 |
| 12 | | 1111 | 0 | 0 | 111100 |
| 13 | | 1111 | 0 | 1 | 111101 |
| 14 | …… | …… | …… | …… | …… |

## 2.2 编码算法的理论分析

根据编码表可知, 混合游程算法根据测试数据中位串的长度分布进行分组, $A1$、$A2$、$A3$、…、$An$, 其中 $n$ 由最长的位串长度 $L\max$ 确定, 每组成员个数由 $L_t$ 决定, 假定 $L_t = 1$, 则可以得出 $n$ 和 $L\max$ 满足如下的关系式:

$$n = \lceil \log_2^{(L\max + 3)} \rceil - 1 \tag{1}$$

游程长度 $L$ 的字串与代码字长度 $E_{h1}$ 之间的关系为:

$$E_{h1} = 2n = 2 \times \lceil \log_2^{(l+3)} - 1 \rceil \tag{2}$$

同理可以求出 $L_t = 2$ 时, 游程长度 $L$ 的字串与代码字长度 $E_h$ 之间的关系为:

$$E_{h2} = \frac{3}{2} \lceil \log_2^{(3l+7)} - 1 \rceil - 3 \tag{3}$$

而对于传统游程算法和 Golomb 算法, 游程长度 $L$ 的字串与代码字长度 $E_h$ 之间的关系分别如公式(4)和公式(5)所示:

$$E_r = (\lceil L/(2^b - 1) \rceil + 1) \times b \tag{4}$$

$$E_g = (L/m + 1) \times \log_2^m \tag{5}$$

对比式(2~5)可以发现, 传统的游程算法和 Golomb 算法中, 代码字长度与原始数据长度的比例都是类似线性的关系. 而混合游程编码算法中代码字长度与原始数据长度则是类似对数的关系. 图 1 给出了三种算法的代码字长度曲线. 其中传统游程算法的块参数 $b$ 选定为 3, Golomb 算法的编码参数 $m$ 选定为 4.

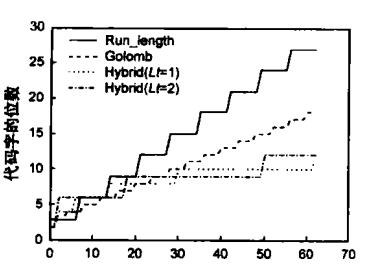


图 1 代码字长度和游程长度关系曲线

从图 1 可以看到当位串的长度小于 20 时, 三种算法的代码字长度比较接近, 相互交错. 但当位串长度大于 20 之后, 混合游程的算法表现出明显优势. 实际电路的测试数据中, 位串的最大长度一般都远远超过 20[2, 3]. 而且电路规模越大, 其可能的位串长度也越长. 由于长位串的压缩率要远大于短位串, 因此在实际电路的压缩中混合游程算法的压缩效率要优于传统游程算法和 *Golomb* 算法. 可以通过概率论对算法的压缩效果进行进一步的分析. 假设测试向量中某一特定位为"0"的概率为 $p$, 则为"1"的概率为 $(1-p)$ $(0 \leqslant p \leqslant 1)$, 这是因为最后进行编码压缩时所有的不确定位已经被填充为确定值.

首先, 根据压缩编码表 1, 可以求出第 $n$ 组中游程长度 $L$ 的范围为:

$$2^n - 3 < l \leqslant 2^{n+1} - 3 \tag{6}$$

而游程长度为 $i$ 的字串属于第 $n$ 组的概率为:

$$p(i, n) = \sum_{(i = 2^n - 3)}^{(2^{(n+1)} - 3)} p^i (1-p) = p^{(2^n - 2)} (1 - p^{2^n}) \tag{7}$$

根据编码表 1, 也可以求出第 $n$ 组的代码字的长度 $C = 2 * n$, 然后根据式(7)可以求出混合游程编码后得到的平均代码字长度:

$$\overline{H_{lt=1}} = \sum_{n=1}^{\infty} 2n * (p^{2^n - 2} (1 - p^{2^n})) = 2 * \sum_{n=1}^{\infty} p^{2^n - 2} \tag{8}$$

对于给定了"0"的概率 $p$ 的某个具体测试数据, 可以算出它们实际数据的平均位数:

$$\lambda = 1/(1 + \sum_{i=1}^{\infty} i * p^i (1-p)) = 1/(1-p) \tag{9}$$

根据式(8)和式(9), 可以算出混合游程编码的压缩增益值:

$$\beta_1 = \lambda / \overline{H_{lt=1}} = 1/(1-p) * (2 * \sum_{n=1}^{\infty} p^{(2^n - 2)}) \tag{10}$$

同理可以求出 $L_t = 2$ 时, 混合游程编码的压缩增益:

$$\beta_2 = \lambda / \overline{H_{lt=2}} = 1/(1-p) * (\sum_{n=1}^{\infty} 3n * (p^{\lfloor (2^n - 7)/3 \rfloor} * (1 - p^{2k}))) \tag{11}$$

为了更形象的说明混合游程编码的压缩效率, 可以进一步根据式(10)和式(11)以及文献[2]和[3], 画出各种算法的相关增益曲线图, 如图 2 所示, 由于 FDR 算法以及混合游程编码的增益函数中都存在无限求和的因素, 所以画图的时候都是取了前面 1000 项作为近似逼近. 从图中可以看出, 当 $L_t = 1$ 时, 混合游程的压缩增益曲线和 FDR 增益曲线基本重合, 说明它们的压缩效果相当; 而随着测试数据中"0"的概率的增加, $L_t = 2$ 时的混合游程编码的增益要明显好于 Golomb 编码和 FDR 编码. 图 2 中, 只取"0"的概率为 0.9~1.0 之间的值, 这是因为在原始测试数据进行优化后, $p$ 一般都在 0.93 以上. 所以从理论分析结果可以看出使用混合游程编码

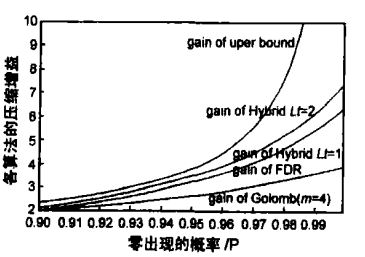


图 2 各种主要压缩编码算法的压缩增益曲线

压缩效果要优于使用 Golomb 编码和 FDR 编码的效果.

## 3 测试数据的优化

从上面的分析可知, 可以通过提高测试数据中零的概率 $p$ 和游程长度来进一步提高编码算法的压缩率; 本文采用的迭代排序填充算法是通过对测试向量的顺序进行重排序, 并把原始的数据转换为差分数据以后才进行压缩操作. 由于是对差分数据进行压缩, 所以提高零的概率 $p$ 就可以等效于减少相邻两个测试向量之间的海明距离. 即, 要想提高压缩率就要尽量减少待测向量上的海明距离, 这是迭代排序填充算法的主要任务. 整个海明距离排序的主功能是由 CStringManage 类的成员函数 HammingSortAl 来完成. 假定 *Array_ input* 和 *Array_ ouput* 分别为排序前后的测试向量集合数组, 排序过程如下: 首先完成第一个向量的排序 *Array_ output*[0] = *Array_ input*[0]; 然后在 *Array_ input* 中找第一个未排序的向量行, 记为 *Li*; 计算 *Li* 与 *Array_ output* 中最后一个向量行(记为 *Ki*)的海明距离, 计算时会记下最小和最大海明距离; 同时, 记下 *Li* 中有 *X* 但 *Ki* 中没有 *X* 的位的个数及索引(*num_ x_ input*); *Ki* 中有 *X* 但 *Li* 中没有 *X* 的位的个数及索引(*num_ x_ output*); 和 *Li* 中有 *X* 且 *Ki* 中也有 *X* 的位的个数及索引(*num_ x_ comm*), 如果 *Ki* 和 *Li* 中各位的数值如下所示:

| 位置 | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| *Ki* | *x* | *x* | 0 | *x* | 1 | *x* | 0 |
| *Li* | *x* | 0 | 0 | *x* | *x* | 1 | *x* |

则 *num_ x_ input* = {4, 6}, *num_ x_ output* = {1, 5}, *num_ x_ comm* = {0, 3}; 然后填充 *Li*, *Ki* 中的 *X* 值, 使 *Li*, *Ki* 的海明距离等于阈值, 并将填充后的 *Li* 放到 *Array_ Input* 中; 然后继续在 *Array_ Input* 中找下一个未排序的行; 最后根据 *num_ x_ input* 和 *num_ x_ output* 判断 *Array_ ouput* 中最后一行的 *X* 是否已经改变过, 如果没有先不回追, 先改变一下 *Array_ ouput* 中最后一行的 *X*, 否则回追. 在这个算法中, 首先需要设定两个初始值: 允许接收的相邻向量海明距离的阈值 *Hamm_ D* 和最大回追深度 *Hamm_ L*(回追的级数). 初始化时需要从文件中读出向量行、得到海明距离阈值、以及回追次数等参数. 伪代码如下所示:

```
Sort(Hamm_D, Hamm_L)
Initial: x< - 0
Array_Output[0] < - Array_Input[x]
if totallines is 1 return true
while true
label1: i< - Find an unsorted Line
label2: Calculate Hamming Distance for Array_Input[i],
Array_Output[j]
  if (Min_D > Hamm_D)
    i < - Find another unsorted line
    if (i is valid) goto label2
    else
      if(last_sort_line_changeable is true)
        change the x value using diffent bit index for Array_Output[j]
        goto label1
      else
        Backtrtace
        goto label1
    else
      Fill X bit value for Array_Input[i], Array_Output[j]
      Array_Output[j] < - Array_Input[i]
      if (all lines are sorted) return true
      else goto label1
    endif
  endwhile
```

## 4 硬件解码单元实现

硬件解码电路是测试数据压缩算法中一个不可或缺的组成部分. 因为如果采用了测试数据压缩算法, 那么在可测性设计中必须要增加一个解码单元来恢复原始的测试数据. 这个解码单元必须控制在一定的规模范围内, 来减小对功能电路的影响. 混合游程算法的构建过程中特别注重考虑了硬件实现的便利性和硬件的开销. 不失一般性, 考虑 $L_t = 1$ 的情况. 从混合游程的编码表可以看出, 给定一个代码字, 可以通过如下的式子求出原始数据的游程长度:

$$l = \{[x(2n) + x(2n+1)] \cdot 2^n + [x(2n-2) + x(2n-3)] \cdot 2^{n-1} + \cdots + [x(2) + x(1)] \cdot 2^1 + index\} \quad (12)$$

其中 $n$ 代表前缀的组成元素的个数, $x(i)$ 表示代码字中前缀部分的第 $i$ 位的值; *index* 表示后缀的值. 例如, 某个代码字为 10_ 11_ 10_ 01, 可以看出该代码字中含有三个前缀组成元素 (10, 11, 10), $n = 3$, 后缀的值为 1, 所以对应的原始数据的游程长度 $L = (2^3 + 2\times 2^2 + 2^1) + 1 = 19$. 根据上面的式子, 可以设计出如图 3 所示的硬件解码电路. 不失一般性, 这里只给出最长代码字长度为 6 位的解码电路, 位数更多的电路结构可以通过它很容易扩展得到.

从结构框图可以发现, 整个解码电路具有结构简练、电路规模小、实现简单等特点. 与文献[1~3, 8]中提到的算法相应的解码电路相比具有明显的优势. 为了使解码电路的控制部分尽可能的简单, 而不需要复杂的状态机(FSM)来实现, 在进

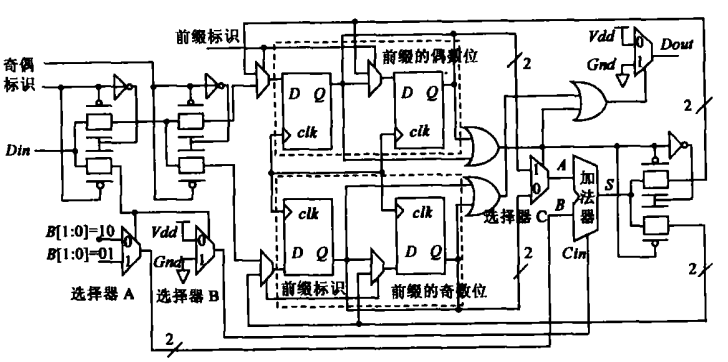


图 3 混合游程算法相应的解码电路框图

行编码算法研究的时候, 引入了前后缀标识位的概念, 所以在解码过程中可以很容易区分出代码字的不同部分, 控制信号简单. 另外在[1~3, 8]中, 解码电路内都需要一个规模很大的计数器来实现连续零个数的计数, 本文中的解码电路通过两组具有选择输入端的移位寄存器和一个公用的加法器来实现对零的计数, 从而省去了硬件开销很大的计数器. 解码电路的工作过程如下: 首先根据标识位, 把前缀的奇数和偶数部分, 即 *Din* 输入, 通过 4 个传输门分别保存在两组移位寄存器中; 然后根据式子(12)可知, 原始数据的游程长度是前缀的奇数和偶数位的值, 加上后缀部分表示的偏移量所得的"和"; 可以分别对这三个量进行计数, 先把后缀的偏移量通过多路选择器 *A* 和 *B*, 与前缀的奇数位的值相加, 然后通过选择器 *C* 和加法器, 对奇数位和偶数位的值开始减 1 操作; 如果在减 1 操作过程中, 奇数位和偶数位的值不全为零, 则从 *Dout* 端输出连续的零, 当且仅当奇数位和偶数位的值都为零时, 在 *Dout* 端输出一个 1, 从而完成一次完整的解码过程.

## 5 实验结果

该部分将通过实验数据来说明混合游程编码的压缩效率, 以及迭代排序填充算法对压缩率的影响. 为了增加实验结果的可比性, 本文采用了美国 Duke 大学提供的 MinTest 产生的测试向量集(与文献[2,3,9]中采用的测试向量数据一致), 并分别对 ISCAS 89benchmark 电路中几个规模较大的时序电路进行了实验.

首先, 表 2 中的实验结果说明了对同样的测试数据, 混合游程编码的压缩效率要比 Golomb 编码, FDR 编码、Alt Rurr length 编码效果好, 压缩效率的计算公式为: $\lambda = [1 - (TE/TD)] \times 100\%$, 其中 *TD* 表示测试数据的原始位数, *TE* 表示编码压缩后测试数据的位数. 表 2 中, 在进行混合游程编码压缩之前, 原始的测试数据首先通过迭代排序填充算法进行了不确定位的填充, 并进行了差分处理得到一组新的差分测试数据. 从表 2 中可以发现, 对于七个不同的时序电路, 混合游程编码的压缩率都有明显的改善: $L_t = 1$ 时混合游程编码与 Golomb 编码的平均差值压缩效率(两种算法针对于七个不同电路的压缩率差值求平均)为 16.74%; 与 FDR 编码的平均差值压缩效率为 11.04%; 与 Alt Rurr length 编码的平均差值压缩效率为 5.81%. 而当 $L_t = 2$ 时, 这些值分别为: 17.88%, 12.18%, 以及 6.87%. 这些数据都充分说明了混合游程编码算法的有效性. 另外, 对于 Alt Rurr length 编码, 它的压缩效率是以高昂的硬件开销为代价的, 而由第四部分的讨论可知, 硬件开销小, 电路实现简单是混合游程压缩编码一个突出特点.

表 2 各种编码算法的压缩率比较

| 电路 | 原始数据位数 | Golomb[2] | | FDR[3] | Alt Rurr length[9] | 混合游程编码 $L_t=1$ | | 混合游程编码 $L_t=2$ | |
|---|---|---|---|---|---|---|---|---|---|
| | | *m* | 压缩率(%) | 压缩率(%) | 压缩率(%) | 编码位数 | 压缩率(%) | 编码位数 | 压缩率(%) |
| *S*5378 | 23754 | 4 | 40.70 | 48.19 | *N/A* | 10347 | 56.44 | 9987 | 57.96 |
| *S*9234 | 39273 | 4 | 43.34 | 44.88 | 44.97 | 15376 | 60.85 | 16068 | 59.09 |
| *S*13207 | 165200 | 16 | 74.78 | 78.67 | 80.23 | 25093 | 84.81 | 21847 | 86.78 |
| *S*15850 | 76986 | 4 | 47.11 | 52.87 | 65.83 | 22371 | 70.94 | 21201 | 72.46 |
| *S*35932 | 28208 | *N/A* | *N/A* | 10.19 | *N/A* | 14739 | 47.75 | 15928 | 43.53 |
| *S*38417 | 164736 | 4 | 44.12 | 54.53 | 60.56 | 60187 | 63.47 | 58072 | 64.75 |
| *S*38584 | 199104 | 4 | 47.71 | 52.85 | 61.14 | 76208 | 61.73 | 71641 | 64.02 |

接下来, 用表 3 中的实验结果来说明迭代排序填充算法的有效性. 在进行实验的时候, 首先采用本文中提出的迭代排序算法对原始测试数据进行排序、不确定位填充、以及进行差分处理, 然后根据文献[2]中的 Golomb 编码算法和文献[3]中的 FDR 编码算法对处理后的测试数据进行压缩. 通过比较表 3 中的实验结果和文献[2~3]中提供的实验数据(表 2 所示), 可以看出迭代排序填充算法有效的提高了编码算法的压缩效率. Golomb 编码算法对于同一组实验电路和原始测试数据, 运用迭代排序填充算法前后, 平均差值压缩效率为: 7.49%; FDR 编码算法的平均差值压缩效率为 13.20%. 这些数值有力的证明了迭代排序填充算法的有效性. 从表 3 中还可以发现, 迭代排序填充算法是通过尽可能的增加被测电路的测试数据中零的概率和游程长度来提高编码压缩效率, 所以, 表中也给出了运用填充算法后各电路相应的测试数据中零的概率和游程长度, 就是为了进一步说明排序算法的本质作用.

表 3 运用迭代排序填充算法后编码算法的压缩效率

| 电路 | 原始数据位数 | 最长游程 | 平均游程长度 | 零的概率(%) | Golomb 编码 | | FDR 编码 | |
|---|---|---|---|---|---|---|---|---|
| | | | | | 编码位数 | 压缩率(%) | 编码位数 | 压缩率(%) |
| *S*5378 | 23754 | 153 | 10.11 | 90.10 | 12085 | 49.13 | 10090 | 57.52 |
| *S*9234 | 39273 | 226 | 12.25 | 91.83 | 18009 | 54.14 | 16826 | 57.16 |
| *S*13207 | 165200 | 443 | 36.11 | 97.63 | 51387 | 68.89 | 26516 | 83.95 |
| *S*15850 | 76986 | 573 | 17.19 | 94.18 | 30596 | 60.26 | 22744 | 70.46 |
| *S*35932 | 28208 | 1601 | 3.91 | 74.39 | 26687 | 5.40 | 18024 | 36.10 |
| *S*38417 | 164736 | 1366 | 12.32 | 91.88 | 74727 | 54.64 | 57500 | 65.10 |
| *S*38584 | 199104 | 746 | 12.91 | 92.25 | 88255 | 55.67 | 71168 | 64.26 |

综上所述, 混合游程编码算法通过灵活的分组策略, 提高了测试数据的压缩率; 其次, 运用迭代排序填充算法对测试数据中不确定位进行填充, 同样可以在很大程度上帮助混合游程编码算法提高压缩率; 另外, 本文提出的迭代排序填充算法不仅适用于混合游程编码算法, 而且也可以提高其它压缩算法的压缩效率.

## 6 结论

为了进一步降低 SOC 芯片的测试成本, 本文提出了混合游程编码算法和迭代排序填充算法. 理论分析和多组实验结果表明: 混合游程编码算法具有高压缩率的特点, 并且具有很好的压缩稳定性; 同时, 该算法引入了前后缀标识位的概念, 可以有效的减小硬件解码电路的规模和设计复杂度, 而不再需要复杂的有限状态机(FSM)来控制压缩数据的解码过程. 另外, 把迭代排序填充算法应用于 Golomb 和 FDR 算法的实验中验证了该排序算法具有很好的适用性和有效性, 使各种算法的压缩率都得到了很大的提高.

致谢 感谢中科院计算所的韩银和博士和董捷在百忙中对本文的算法提出了很多宝贵的意见, 并提供了实验用的测

试向量和数据.

参考文献:

[1] A Jas, J Ghosh-Dastidar, N A Touba. Scan vector compression/decompression using statistical coding[A]. Proceeding of 17th IEEE VLSI Test Symposium [C]. Dana Point, California, USA, 1999. 114-120.

[2] A Chandra, Kchakrabarty. System-on-a-Chip test data compression and decompression architectures based on Golomb codes[J]. IEEE Trans. on CAD of Integrated Circuits and System, 2001, 20(3): 355-368.

[3] A Chandra, K Chakrabarty. Frequency-Directed run length (FDR) codes with application to system-on-a chip test data compression[A]. Proceeding of 20th IEEE VLSI Test Symposium[C]. Marina Del Rey, California, USA, 2001. 42-47.

[4] P T Gonciari. Improving compression ratio, area overhead, and test application time for soc test data compression[A]. Proc. Design and Test in Europe[C]. Paris, France, March 4-8, 2002. 604-611.

[5] A Jas, J Ghosh-Dastidar. An efficient test vector compression scheme using selective huffman coding[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2003, 22(6): 797-806.

[6] L Li, K Chakrabarty. Test data compression using dictionaries and fixed-length indices[A]. Proceeding of IEEE VLSI Test Symposium[C]. Napa Valley, California, USA, 2003. 219-224.

[7] Yinhe Han, Yongjun Xu. Test resource partitioning based on efficient response compaction for test time and tester channels reduction[A]. Proceeding of Asian Test Symposium[C]. Xi'an, ShaanXi, P R China, 2003. 440-445.

[8] 韩银和, 李晓维. 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 Yin-he, LI Xiao-wei, et al. Test resource partitioning using variable-tail code[J]. Acta Electronica Sinica, 2004, 32(8): 1346-1350. (in Chinese)

[9] A Chandra, K Chakrabarty. Reduction of SOC test data volume, scan power and testing time using alternating run-length codes[A]. Proceeding of IEEE/ACM, Design Automation Conference[C]. New Orleans, Louisiana, USA, 2002. 673-678.

作者简介:

方建平　男, 1978 年生于浙江省义乌市, 博士研究生, 2003 年获得西安电子科技大学工学硕士学位, 并于同年进入西安电子科大微电子学院攻读博士学位, 主要研究方向是: 数字集成电路设计, SOC 测试资源划分和优化, 数字电路的可测性及测试技术. E-mail: fjp0828@yahoo.com.cn.

郝　跃　男, 1958 年出生, 教授, 博士生导师, IEEE 高级会员, 主要的研究方向有: 超深亚微米器件可靠性, 新型宽禁带半导体材料, VLSI 集成电路设计技术.