# 使用kmeasn++算法结合位翻转算法进一步提高压缩率

测试数据压缩能减少数据存储开销，降低测试时间。测试集由一系列0、1以及无关位X组成，X、可以被任意填充为0或者1而不影响故障覆盖率，因此往往包含较多无关位的测试集会获得较高的压缩增益。本章将基于这种思想对原测试集进行修改从而提高压缩率。

## 5.1 相关概念

### 5.1.1 故障检测冗余度

故障检测冗余度是指单个故障在测试过程中被重复检测。比如，数据冗余是指在一个集合中相同数据重复出现，那么故障检测冗余度类也是类似的。测试集由ATPG产生，一开始产生的测试集会存在大量的数据，这些数据中有很多无关位，通过位流动态紧缩之后测试集的规模会大大减小，减小的测试集故障覆盖率并不会改变，因为很多无关位会被转化成为确定位，如此，电路中会存在多个故障被重复检测。

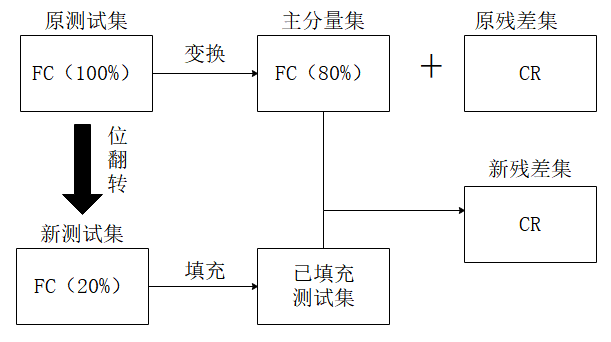
### 5.1.2 位翻转

本文主要使用拆分压缩的方式提高压缩率，拆分压缩会将原测试集拆分成为主分量集和残分量集，如果主分量集和原测试集高度相似便会提高残分量集的压缩率。假设使用主分量集代替原测试对电路进行故障模拟，就会检测出一部分故障，因此在使用原测试集对电路进行故障模拟时便会产生大量的故障冗余度。既然一部分故障已经被主分量集合检测，那么原测试集只需要检测剩余故障即可，假设原测试的总故障覆盖率为C，主分量的故障覆盖率为A，剩余故障覆盖率为C-A。原测试集只需要达到C-A的故障覆盖率即可，因此可以将原测试及中部分确定位翻转位无关位，在总故障覆盖率不变的情况下，提高压缩率。这种确定位转变为无关位的过程称之为位翻转。

### 5.1.3 位翻转应用于压缩

原测试集中包含的确定位会影响最终的压缩率，如果可以将部分确定位翻转成为无关位，将无关位按照有利于压缩的方向填充便能提高压缩率。为了降低硬件开销以及实验复杂度本人使用的是一轮位翻转，具体过程分为下述几个步骤：1、结合第三章与第四章的算法将原测试拆分成为主分量集和残差集。2、将主分量集进行故障模拟，记录能检测的故障数A，在不影响故障覆盖率的前提下将原测试集中的某些确定位翻转成为无关位X，生成新的测试集。3、将新的测试中的无关位进行填充然后与原主分量集进行异或，在本实验中无关位的填充方式主要根据主分量集对应的位进行填充。

位翻转应用于压缩过程如图5.1所示：假设原测试集的故障覆盖率(FC)为100%，拆分压缩之后的主分量集能取得的故障覆盖率可达到70%，那么原测试集只需要检测出剩余的30%的故障即可，在保证故障覆盖率的条件下，对原测试集的确定位进行翻转得到新测试集，根据主分量的确定位对原测试集的无关位进行填充，最后将主分量集与已填充的测试集进行异或得到最终需要进行压缩的新残差集。



位翻转应用于压缩

在论文“多次随机变换拆分测试激励压缩方法研究”[]中提及了多轮为翻转算法，但是由于每一轮翻转，均会产生代价，基本上在进行第二轮反转时提升的压缩率就会大大低于代价，本人化繁为简直接使用一轮压缩之后根据主分量集对原测试集进行二次填充，在保证压缩率的前提下节约了硬件开销以及存储代价。

## 5.2 翻转算法

假设原测试集经过故障模拟后能达到的故障覆盖率为C，对原测试进行位填充之后，拆分为主分量集和残差集，对主分量集进行故障模拟之后能达到的故障覆盖率为B，在对原测试及进行位翻转之后得到了新测试集，假设新测试集能达到的故障覆盖率为A，那么A+B>=C,下文将基于这种思路写出位翻转算法的伪代码。

|  |
| --- |
| 算法5.1 位翻转基本过程 |
| /\*  CoverageC //原测试集的故障覆盖率  CoverageA //拆分原测试集后主分量集的故障覆盖率  \*/  CoverageC ← FaultCoverage (T)  CoverageB ← FaultCoverage (L)  for bit in T  if bit = 1 or 0 //change 1 or 0 to X  bit ←X  /\*If the fault coverage changed, restore\*/  if Coverage < CoverageA - CoverageB  bit ←1 or 0  return T |

算法5.1为翻转测试集的基本过程，首先依次将测试集中的每个确定位变为无关位X，如果影响故障覆盖率，则将无关位X还原为翻转前的确定位。假如测试集总共含有k个确定位，模拟一个向量需要时间为t，则该算法运行的总时间为kt。理论上而言该算法可以通过较小的代价取得较高的压缩率。

## 5.3 实验结果

为了验证预填充方法的有效性，我们对 ISCAS’89[57]中大部分电路进行了实验，包括S5378、S9234、S13207、S15850等电路，本章将挑选其中的电路做具体描述，为了验证本实验是否对多种编码方式均适用，本文选取了FDR、EFDR、ALT-FDR编码方式对变换拆分之后的残差集进行压缩，同时与单纯使用位翻转算法达到的压缩率进行对比。

实验结果如下所示，下表5.1、5.2、5.3分别表示各方法在FDR、EFDR、 ALT-FDR编码下缩能达到的压缩率，其中第一列为电路名称，第二列表示对测试集直接编码所能取得的压缩率，第三列表示使用原测试集集合kmeans++算法所能达到的压缩率，第四列表示对测试集直接反转再编码所达到的压缩率，第五列为使用本方法所达到的压缩率。

表5.1 FDR编码压缩率(%)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 电路名 | 直接编码 | Kmeans++聚类 | 直接翻转 | 本方法 |
| s5378 | 47.98 | 70.76 | 78.06 | 79.69 |
| s9234 | 43.61 | 69.59 | 74.01 | 78.06 |
| s13207 | 81.31 | 92.29 | 91.93 | 94.93 |
| s15850 | 66.21 | 81.75 | 86.04 | 86.27 |
| s38417 | 43.21 | 75.35 | 78.71 | 80.31 |
| 平均 | 56.46 | 77.95 | 81.75 | 83.85 |

表5.2 EFDR编码压缩率(%)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 电路名 | 直接编码 | Kmeans++聚类 | 直接翻转 | 本方法 |
| s5378 | 53.67 | 67.75 | 76.16 | 77.68 |
| s9234 | 48.66 | 66.14 | 71.28 | 75.85 |
| s13207 | 82.49 | 91.60 | 91.53 | 94.52 |
| s15850 | 68.66 | 79.84 | 84.24 | 84.99 |
| s38417 | 62.02 | 74.06 | 78.24 | 79.13 |
| 平均 | 63.0 | 75.88 | 80.29 | 82.43 |

表5.3 ALT-FDR编码压缩率(%)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 电路名 | 直接编码 | Kmeans++聚类 | 直接翻转 | 本方法 |
| s5378 | 49.95 | 65.17 | 72.17 | 76.26 |
| s9234 | 45.14 | 62.72 | 67.61 | 73.50 |
| s13207 | 80.12 | 90.88 | 88.93 | 94.09 |
| s15850 | 65.64 | 77.82 | 81.64 | 83.85 |
| s38417 | 60.52 | 71.23 | 75.32 | 76.93 |
| 平均 | 60.27 | 73.56 | 77.13 | 80.93 |

从上表可以看出使用本方法使测试集的压缩率得到了极大的提升，比对电路直接编码所达到的平均压缩率高22.49%，比对原测试集直接翻转所取得的平均压缩率高2.68%。

## 5.4 小结

位翻转方法在不改变原测试集故障覆盖率的条件下，通过将确定位翻转为无关位来提高压缩率。拆分压缩会将原测试集拆分为主分量集与残差集，主分量本就可以检测一部分故障，因此原测试只需要检测主分量集未检测到的故障即可。当原测试集的某些确定位翻转为无关位后，得到了一个新测试集，新测试集填充无关位之后与主分量异或得到了残差集，此时的残差集包含的0比特位大大增加，从而提高了压缩率。上表中也很好地反映了这一结果，相比于第四章我们所获取的压缩率，本章将压缩率在其基础上提高了7%。