International Journal of Applied Engineering Research
ISSN 0973-4562 Volume 10, Number 11 (2015) pp. 28731-28739
© Research India Publications
http://www.ripublication.com

Static Relaxation Technique With Test Vector Compression

Deepak Rajan
Department of ECE, Amrita School of Engineering
Amrita Vishwa Vidyapeetham, Coimbatore, India
Email : deepu.r257@gmail.com
Dr. J.P.Anita
Department of ECE, Amrita School of Engineering
Amrita Vishwa Vidyapeetham, Coimbatore, India
Email : jp_anita@cb.amrita.edu

Abstract

The main challenges faced in VLSI testing in the current scenario is the large test power compared to normal operational power of a circuit. As the toggling between bits (1's and 0's) pave way for considerable test power, these activities should be reduced to a maximum extent. Also the large volume of test data consumes more memory for storing and also test time. So the proposed methodology aims at reducing the toggling between 2 bits and thereby reducing test time. This algorithm is called Static Technique. This technique is used to convert a test set with partially or fully specified bits into a new test set having large number of unspecified bits. The unspecified bits means don't care bits that can have either a '1' or a '0'. The insertion of a larger number of unspecified bits in the proposed method reduce the toggling activity to a large extent and thus test power is reduced.
In order to reduce the data volumes of test patterns, test compression is done after the relaxed test patterns are obtained from static technique mentioned. There are various compression techniques having unique encoding and decoding schemes based on a particular algorithm. In this work Golomb coding technique has been used for test compression. This technique codes a data depending upon the run length of 0's. Finally a reduction in the number of specified bits is observed using static technique and also reduction in the number of bits to represent a test pattern using Golomb encoding technique.

Keywords: Component; Test Set Relaxation, Test Power, Test Vector Compression

Introduction

A set of inputs applied to test a digital circuit is called test patterns. When these test patterns have large amount of unspecified bits, it is called flexible test set. Many problems occurring in a circuit can get advantage when flexible test sets are used. Flexible test sets provide many advantages like low power dissipation, coincidental detection of fault, favorable for various compression schemes. Many of the prevailing test generation schemes generates fully specified test sets. So there is a need for techniques to increase the number of unspecified bits and maintain the fault coverage.

There are two main compaction techniques used. They are static and dynamic techniques[1]. In static techniques a particular algorithm is applied to the test patterns generated by the ATPG. So the flexibility and amount of specified bits is larger in this case. Dynamic technique forces every ATPG tool to produce set size which is smaller in size. This technique cannot make use of the random test generation. In short dynamic method forces an ATPG to produce patterns with lower specified bits. So new techniques are to be evolved so as to overcome this drawback. So if more and more compaction schemes are implemented along with reduction in specified bits criteria, the entire dynamic technique will increase the number of specified bits. So it is observed that the dynamic scheme mainly concentrates on finding minimum number of tests rather than minimizing the number of specified bits. So static techniques is used for compaction.

After a set of test patterns are obtained after compaction, there is a need to reduce the number of bits representing a test pattern. This reduction can be done through various compression schemes. Large volume of test data usually consumes memory and increases testing time. So significant memory can be saved by using test compression. Golomb coding[2] scheme is used as the encoding and decoding is simpler than other techniques like block merging[3] and frequency directed run length (FDR) coding[4].

In this work a static compaction technique is proposed to produce flexible test set[5] and encode the patterns using Golomb coding scheme so that bit length of pattern is reduced. The static method makes use of coincidental detection of faults. At first it analyses all the test patterns generated by ATPG along with its faults. The main condition of this static scheme is that once a fault is detected by a test pattern, the same fault need not be detected by any other patterns. So the specified bits which is responsible for the extra detection can be converted to don’t care bits. Thus overall number of specified bits is reduced without reducing the overall fault coverage. Now the bit length of each compacted test pattern is reduced using Golomb coding. Golomb coding makes use of the fact that two test patterns in any testing circuit will differ less in bitwise position. That is, a test pattern differs in bitwise from its adjacent test pattern is less. The remaining bits in the test pattern will remain the same. Golomb coding scans the run length of zeros in a data and encodes them. It can be observed that the newly encoded data has lesser bit length and uses less memory for storage.

Static Test Set Relaxation

This algorithm produces flexible test set using a particular scheme. The scheme is defined as changing a fully or partially specified test set with a new set of test set in such a way that each fault is detected only once and maintain the fault coverage. Also test size is not increased. So a fault detected more than once is not essential and the specified bits responsible for that fault are changed to don't care (x) bits.

A few notations and rules to be followed in this static technique is discussed below. Consider a set of patterns P = {p_1, p_2,…..,p_N} for a testing circuit C. each test pattern consist of bits of three types i.e {0,1,x} . Let M be a fault model the list of faults noticed by test set (T) be F. Assume K(T) as the ratio of amount of specified bits to the total quantity of test set bits. The ratio gives a value that shows how elastic a test set is. The range of K(T) is between 0 and 1. The closer the value K(T) to *0*, T is said to be more elastic. The value of K(T) for a fully definite set is unity.

The algorithm by which the static technique works is discussed below. consider a test circuit C in which a fault model M is intriduced. let T be the test set that contains all test patterns and F the list of faults detected by T. The steps involved in this algorithm are

1 : Obtain test set T for fault model M
2 : Obtain F, faults found out by T
3 : Initialize new test set T' and F' to null value
4 : Consider each test pattern t_i (i=1,2,3,…,N) of test set T
5 : Find all faults F_i detected by t_i
6 : Generate test set t' that detects all faults in F_i - F'
7 : Update test set T' with new patterns having less number of specified bits
8 : Update Fault list F'
9 : Stop when all faults are scanned i.e F'=F.

Table 1: An example for Static algorithm:

	Initial Test set (T)				
Fault	***p_1***	***p_2***	***p_3***	***p_4***	***p_5***
x_1	●	●			
x_2		●			
x_3	●		●		
x_4			●		●
x_5		●	●		
x_6	●		●	●	
x_7				●	
x_8				●	●
Sp. bits	24	21	30	23	20
Total	**118**				

(a) Original test set;

	Test set (T')			
p_1	***p_2***	***p_3***	***p_4***	***p_5***
●	-			
	●			
●		-		
		●		-
	●	-		
●		-	-	
			●	
			●	-
24	18	23	20	0
Total	85			

(b) Relaxed test set

The lines 1 and 2 in the above algorithm represents outputs from ATPG. Line 3 just initializes the new fault and test set so that it is updated whenever required. Line 4 starts the process with each test pattern t_i in test set T. Line 5 finds out all the faults F_i detected by t_i (i=1,2,3,…,N) . Here N is the total number of test patterns. Line 6 produces a test replacement t' which detects faults that are not detected by previous faults. If the new test patterns have lesser number of specified bits, T' (the new test set list) is updated as in line 7. The fault lists are updated when it is detected once and the process stops when all faults have been processed.

An illustrative example showing how exactly static algorithm works is discussed in the following section. Table 1(a) shows initial test set with a set of patterns (p_1, p_2..... p_5) detecting its respective faults (x_1, x_2 ,.... x_8). Table 1(b) shows the relaxed test in which a fault is detected only by one pattern. In this table pattern p_2 do not detect x_1 as this fault is already detected by p_1 . So the quantity of specified bits in p_2 is reduced from 21 to 18. Similarly the entire pattern p_5 can be eliminated as its fault x_4 and x_8 are already detected by p_3 and p_4 .In the similar manner the process continues and specified bits are reduced using coincidental detection of fault. Finally total specified bits is reduced from 118 to 85 using static technique.

Compression Using Modified Golomb Coding

As the circuit size increases there will be a large amount of test data to be stored. This leads to high memory usage and increase in testing time[6]. In order to reduce the test data volume encoding is done using Golomb coding. This code uses the fact that there will be very less difference between the adjacent test patterns.

Table 2: Example of codes in golomb coding technique

Group	**Run-length**	**Group Prefix**	**Tail**	**Codeword**
G_1	0	0	00	000
	1		01	001
	2		10	010
	3		11	011
G_2	4	10	00	1000
	5		01	1001
	6		10	1010
	7		11	1011
G_3	8	110	00	11000
	9		01	11001
	10		10	11010
	11		11	11011
------	------	-------	---------	--------

For example consider two test patterns ,

t_1 (x0xxx0xxx0x1x10x00xxx0xxx0x1x1xxx0xx) and

t_2 (x0xxx0xxx0x1x10x00xxx0x1x1x1x1xxx0xx)

Comparing these two patterns bitwise, the difference in bits is only in 24th and 26th bit. So pattern 1 is made as reference. Then exclusive OR with t_1 and t_2 is done. So a stream of zeros as output with bit 1 in position 24 and 26 are obtained. So after doing this logical operation, a pattern having a large number of zeros and some 1's is obtained. This advantage is used in coding these run length of zeros according to a particular coding scheme.

Table 5 shows how a run length of zeros are codes using Golomb coding. The run-length l means the total number of zeros in a single run. The encoding scheme has groups having group size m=4. There is a prefix merely to identify the group. The tail numbers just shows the index inside the group. For example if there are 9 zeros in pattern so l=9 and it belongs to group G_3 and it has prefix value as 110 and tail as 011.

A simple example of how an encoding takes place is shown below :

Consider the pattern to be coded is

0001 000001 1 00001 00001 001 00000001 001.

Total number of bits is 42 and the code have zero run lengths as l_1 =3, l_2 =5, l_3 =0, l_4 =4,l_5 =4, l_6 =6, l_7 =2, l_8 =7, l_9 =3. For example the code for run-length 3 is 011 and code for run-length 5 is 1001. Considering these run length values the coding can be done as shown in Table 2

The encoded code is obtained as

T_E = 011 1001 000 1000 1000 1010 010 1011 001

Consider one more example. Let patterns p_1 and p_2 be as below :

p_1 = x0xxx0xxx0xxx0x1x10x00xxx0xxx0xxx0xx

p_2 = x0xxx0xxx0x1x10x00xxx0xxx0xxx0xxx0xx

Here p_1 is considered as main reference pattern i.e this pattern is used as reference pattern while decompressing. Reference pattern is selected considering the fact that it differs only little in bitwise with other patterns to be compressed. This pattern p_1 contains don't care values. So the reference pattern is modified to a modified pattern (m) so that don't care is changed to logic value similar to the compressing pattern (p_2). Now the modified pattern will be

m = x0xxx0xxx0x1x001010x00xxx0xxx0xxx0xx

so don't care bits are changed to logic values of p_2. This pattern *m* only consumes memory of bits that is changed from don't care to logic values.

Bitwise xor is made between pattern m and p_2 the intermediate code n is obtained as follows :

n = 000000000000 1 0 1 0 11 01 000000000000000

When this code is decoded according to golomb code as shown in fig 1. The encoded code is obtained as follows

T_E = 111001 001 001 000 001 111011

The encoded data has 24 bits which is less than 36 bits of the initial pattern

Decompression

Any compression technique is said to be successful only when the original data can be retrieved from the encoded data. Golomb coding decompression uses a simple logic. Consider the encoded pattern as

TE = 011 1001 000 1000 1000 1010 010 1011 001

First decoding algorithm senses first 0 in the coded data. This zero marks the end of group prefix and the upcoming 2 bits are sensed as the tail. For example, consider the second part of the coded data 1001. The first zero shows the end of group prefix and later two bits are taken as tail. So here group prefix is 10 and tail is 01. Thus the run length is found out as 5. The obtained intermediate code (n) is compared with modified reference pattern (m) and initial reference pattern (p1) to get the required pattern p2.

Experimental Results

The static relaxation method along with Golomb coding compression was carried out in C language and run on 2.3GHz core i5, running linux with 4 GB RAM. Initial test sets are obtained from ATALANTA. Table 3 shows the reduction in number of specified bits when experimented with some ISCAS '85 and ISCAS '89 benchmark circuits .

Table 3: Results of static relaxation Technique

	Initial test set			**Static test set**		
Circuit	***PI's***	***T***	***Sp. Bits***	***T'***	***Sp. Bits***	***Reduction %***
c432	36	30	490	5	86	82.44
c499	41	36	1392	3	39	97.19
c880	60	21	317	10	81	74.4

c1355	41	44	1685	9	150	91.09
c3540	50	29	531	5	89	83.2
s832	23	18	90	3	6	93.3
s1196	32	20	192	9	66	65.6
s1238	32	23	187	7	51	73

In the above table, a comparison is made in the amount of specified bits between initial test set and test set obtained after static relaxation technique. PI’s refers to number of primary inputs, T and T’ are test sets. Sp. Bits refers to the amount of specified bits. Thus it is found that the amount of specified bits are reduced and the reduction percentage is showed in the above table.

The results obtained after compression is shown in Table 4. Test patterns are obtained from c432 benchmark circuit using ATALANTA. As explained earlier reference pattern (p1) is assumed. From this pattern a modified pattern (m) is obtained. in accordance with the pattern to be compressed that is p2.

Table 4: Results showing the reduction in encoded data bits

Reference pattern (p_1)	**x0xxx0xxx0xxx0x1x10x00xxx0xxx0xxx0xx**
Pattern to be compressed (p_2)	x0xxx0xxx0x1x10x00xxx0xxx0xxx0xxx0xx
Modified ref. pattern (m)	x0xxx0xxx0x1x001010x00xxx0xxx0xxx0xx
Bitwise XOR output (n=m XOR p_2)	000000000000010101101000000000000000
Encoded data (T_E)	111001 001 001 000 001 111011
Encoded data bits	24

Table 5 shows the results of encoding of test patterns for different circuits.

Table 5: Results of encoding for different circuits

No.	**Circuit**	**No. of bits before encoding**	**No. of bits after encoding**
1	c432	36	24
2	c499	41	31
3	c880	60	39
4	c3540	50	30
5	s832	23	8
6	s1196	32	18
7	s1238	32	24

It is found that the compression scheme reduced the number of bits to a much extent as shown in table 5.

Conclusions

The work discusses static technique to increase the number of unspecified bits. Also compression is carried out from the obtained test patterns to reduce the number of bits so that memory can be saved.

References

[1] Stelios N. Neophytou, Member, IEEE, and Maria K. Michael, Member, IEEE, "Test Set Generation with a Large Number of Unspecified Bits Using Static and Dynamic Techniques, " *IEEE transactions on Computers*, Vol. 59, No. 3, March 2010

[2] A. Chandra and K. Chakrabarty, "Test data compression and decompression based on internal scan chains and Golomb coding*," IEEE Trans. Computer Aided Design of Integrated Circuits and Systems*, Vol. 21, No. 6, pp. 715-722, June 2002.

[3] Aiman El-Maleh, "An Efficient Test Vector Compression Technique Based on Block Merging" *IEEE International Symposium on Circuits and Systems* , 2006. ISCAS 2006. Proceedings.

[4] A. Chandra, and K. Chakrabarty, "Test data compression and test resource partitioning for system-on-a-chip using frequency-directed runlength (FDR) Codes," IEEE Trans. Comp., Vol. 52, pp. 1076-1088, 2003.

[5] K. Miyase and S. Kajihara, "XID: Don't Care Identification of TestPatterns for Combinational Circuits," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 2, pp. 321-326,Feb. 2004.

[6] M. Bushnell and V. Agrawal, Essentials of Electronic Testing. Kluwer Academic Publishers, 2000.

[7] Kanad Basu, Student Member, IEEE, and Prabhat Mishra, Senior Member, IEEE, "Test Data Compression Using Efficient Bitmask and Dictionary Selection Methods, " *IEEE Transaction on very large scale integration systems,* vol. 18, no. 9, September 2010

[8] Chia-Yi Lin, Hsiu-Chuan Lin, and Hung-Ming Chen, "On Reducing Test Power and Test Volume by Selective Pattern Compression Schemes, " *IEEE Transaction on very large scale integration systems*, vol. 18, no. 8, August 2010

[9] Xiao Liu, Student Member, IEEE, and Qiang Xu, Member, IEEE, "On X-Variable Filling and Flipping for Capture-Power Reduction in Linear Decompressor-Based Test Compression Environment, " *IEEE Transaction on Computer-aided Design of Integrated Circuits and Systems,* vol. 31, no.. 11, November 2012

[10] Jia Li, Student Member, IEEE, Qiang Xu, Member, IEEE, YuHu, Member, IEEE, and Xiaowei Li, Senior Member, IEEE, "X-Filling for Simultaneous Shift- and Capture-Power Reduction in At-Speed Scan-

Based Testing, “ *IEEE Transaction on very large scale integration systems*, vol. 18, no. 7, july 2010

[11] Anshuman Chandra and Krishnendu Chakrabarty, “Test Data Compression and Decompression Based on Internal Scan Chains and Golomb Coding , ” *Transaction on Computer-aided Design of Integrated Circuits and Systems*, vol. 21, no. 6, june 2002

[12] Mohammad Tehranipoor, Member, IEEE, Mehrdad Nourani, Senior Member, IEEE, and Krishnendu Chakrabarty, Senior Member, IEEE, “Nine-Coded Compression Technique for Testing Embedded Cores in SoCs” *IEEE Transaction on very large scale integration systems*, vol. 13, no. 6, june 2005