한국산학기술학회논문지
Vol. 5, No. 6, pp. 515-520, 2004

SoC 테스트를 위한 테스트 데이터 압축

김 윤 홍[1*]

Test Data Compression for SoC Testing

Yun-Hong Kim[1*]

요 약 코아(core) 기반의 SoC(System-on-Chip) 설계는 테스트에 관련된 많은 어려운 문제를 일으키고 있다. 그 중에서 방대한 분량의 테스트 데이터와 긴 테스트 패턴 인가시간은 SoC 테스트에서의 2가지 주요 문제로 떠오르고 있다. 많은 양의 테스트 데이터에 대한 저장공간과 인가시간을 줄이기 위한 방안으로서 테스트 벡터들의 반복되는 성질을 이용하여 최대한 효율적으로 압축하는 다양한 방법들이 제시되었다. 본 논문에서는 SoC 테스트를 위한 효율적인 테스트 데이터 압축 방법을 제안한다. 제안된 방법은 테스트 벡터 집합을 분할하고 최대한 반복되는 공통부분을 제거함으로써 테스트 데이터를 압축한다. 이 압축방법은 O(n2)의 시간복잡도를 가지며, 간단한 디코딩 하드웨어를 사용한다. 여기서 n은 테스트 벡터 수이다. 제안된 압축방법은 빠르고 쉬운 디코딩을 함께 사용하여 기존의 복잡한 소프트웨어 방식의 압축방법에 견줄만한 수준의 효율성을 보여준다.

Abstract Core-based system-on-a-chip (SoC) designs present a number of test challenges. Two major problems that are becoming increasingly important are long application time during manufacturing test and high volume of test data. Highly efficient compression techniques have been proposed to reduce storage and application time for high volume data by exploiting the repetitive nature of test vectors. This paper proposes a new test data compression technique for SoC testing. In the proposed technique, compression is achieved by partitioning the test vector set and removing repeating segment. This process has $O(n^2)$ time complexity for compression with a simple hardware decoding circuitry. It is shown that the efficiency of the proposed compression technique is comparable with sophisticated software compression techniques with the advantage of easy and fast decoding.

Key Words : Test data compression, SoC

1. 서 론

하나의 다이(die)에 다양한 기능을 함께 집적시킨 SoC 칩들은 여러 개의 재사용 가능한 IP(Intellectual Property) 코아들로 구성된다. IP 코아들이 점점 더 복잡해지고 한 칩 안에 집적되는 코아 수가 많아짐에 따라 SoC에 대한 테스트 데이터 분량은 급격하게 증가한다. 이런 SoC 칩들을 효과적으로 테스트하기 위해서는 IP 공급자들에 의해 미리 만들어져서 제공되는 테스트 벡터 집합을 이용하여 적절히 각 IP 코아들이 테스트되어야 한다. 그러나 자동 테스트 장비(ATE, Automatic Test Equipment)의 입출력 채널 용량, 속도와 정확도, 데이터 메모리 등은 제한되어 있기 때문에, 방대한 양의 테스트 데이터를 SoC에 인가하는 것이 점점 더 어려워져서 테스트 시간과 테스트 비용이 상당히 증가할 수 밖에 없다.

테스트 데이터 양의 감소는 ATE의 최소 메모리 용량을 낮출 뿐만 아니라 테스트 시간도 줄인다. SoC에 대한 테스트 시간은 테스트 데이터 양, 데이터를 코아에 전송하는 시간, 테스트 데이터 전송속도, 스캔 체인의 최대 길이 등에 영향을 받는다. 전체 테스트 시간은 테스트 데이터 양을 감소시키거나 스캔 체인 길이를 줄이므로서 더 짧아질 수 있는데, 이 중 테스트 데이터 양을 줄이는 것이 더 효과적이다.

압축은 테스트 분야에서 몇 가지 문제를 해결하기 위해서 그 동안 연구되었다. 무손실 압축은 원래의 데이터가 디코더에 의해 그대로 다시 만들어질 수 있도록

이 논문은 2003년도 상명대학교 교내연구비의 지원에 의하여 연구 되었음.
[1]상명대학교 천안캠퍼스 공과대학 컴퓨터시스템공학과
*교신저자: 김윤홍(yunkim@smu.ac.kr)

테스트 벡터를 인코딩하는 과정을 말한다. 무손실 압축에서의 기본적인 이슈는 데이터 입력 집합을 어떤 이벤트들의 연속으로 해석을 하여 이런 이벤트들을 가능한 한 작은 비트 수로 어떻게 인코딩하는가이다. 테스트 컴팩션(test compaction)은 테스트 생성 시에 테스트 벡터 수를 줄이려는 방법인 반면, 데이터 압축은 최소한의 비트 수로 테스트 데이터를 인코딩하는 방법이기 때문에, 서로 다른 목적을 갖고 있다. 데이터 압축은 원래의 데이터에 중복되는 내용이 많을수록 높은 압축율을 보인다. 예를 들어, ATE 상에서 SoC를 테스트할 때, 몇 개의 코아만이 동시에 테스트되고 나머지 코아들은 제어와 관측을 위해서 동작을 하지 않게 된다. 따라서 테스트되는 부분에 해당되는 테스트 데이터만이 바뀔 뿐이며, SoC의 나머지 부분은 동작이 정지되거나 고정된 데이터만을 받게 된다. 바로 이와 같은 테스트 데이터의 고정된 부분이 압축될 첫번째 대상이 된다.

본 논문에서는 ATE를 사용하는 SoC 테스트에 적용할 수 있는 새로운 데이터 압축방법을 제안한다. 제안된 방법은 ATE의 헤드를 설계할 때 도입되면, ATE의 메모리 용량과 테스트 시간을 줄일 수 있고, 이와 함께 off-chip과 on-chip 디코딩 하드웨어가 간단해 진다. 제안된 방법에서는 테스트 데이터를 몇 개의 블록으로 분할하고 각 블록에서 반복되는 부분을 제거하여 압축이 수행된다. 여기서 $O(n^2)$의 시간복잡도를 갖는 분할 방법이 제시된다. 여기서 n은 벡터 수이다. 또한 본 논문에서는 압축을 복원하는 하드웨어를 제안하는데, 디코딩 과정이 간단하고 시프트레지스터에 의해 수행된다. 디코더의 하드웨어 복잡도는 SoC의 스캔 체인의 길이에 선형적으로 비례한다.

2. 테스트 데이터의 압축

ATE를 통한 SoC 테스트에서 접하게 되는 어려운 문제 중의 하나는 방대한 양의 테스트 데이터이다. [1]에서는 테스트 데이터에 다양한 데이터 압축방법을 적용해 보는 연구를 하였다. 이 논문에서 검토해 본 압축방법에는 run-length 코딩, Huffman 코딩, Lempel-Ziv 코딩, arithmetic 코딩 등이 있다. Run-length 코딩에서는 연속적으로 발생하는 심볼을 두가지 요소, 즉 반복되는 심볼과 그 발생횟수로 인코딩한다. run-length 코딩은 동일한 심볼이 연속적으로 오랫동안 발생하는 데이터에 대해 효율적이다.

Huffman 코딩은 run-length 코딩보다 훨씬 더 복잡하다. 이 코딩방식은 심볼의 발생빈도를 조사하여 그 발생빈도에 따라 각 심볼을 최적의 2진수열로 표현한다. 따라서 빈번한 심볼에는 짧은 코드워드를, 가끔씩 발생하는 심볼에는 긴 코드워드를 부여함으로써 압축이 이루어진다. [2]는 SoC에 내장된 프로세서를 사용하여 나머지 다른 부분의 테스트를 수행하는 방법을 제안하였다. 기본 원리는 ATE가 압축 테스트 데이터와 함께 프로그램을 칩 내부의 메모리에 적재하면, 프로세서가 그 프로그램을 실행시켜 압축 데이터를 풀어서 SoC의 다른 부분에 있는 스캔 체인에 인가하는 것이다. 그러나 이 방법은 ATE 구조 내부에 별도의 장치가 있어야 한다는 제약이 있다.

Lempel-Ziv (LZ) [3, 4] 방법은 사전(dictionary) 구성을 기반으로 하여 패턴들의 목록을 만든 후에, 목록의 인덱스에 의해 패턴을 인코딩한다. LZ 압축법은 발생빈도에 관한 정보를 미리 알 필요가 없으며 길이가 긴 패턴에 더 효율적이다. 그러나 이 방법은 상당한 양의 메모리를 필요로 하기 때문에 큰 규모의 설계에 적용하기에는 한계가 있다.

[1]은 테스트 데이터에 Burrows-Wheeler (BW) 변환을 적용하였다. 연속적인 테스트 벡터들은 반복되는 경향이 있다는 분석 결과를 근거로 하고 있는데, 이 방법은 테스트 데이터를 우선 행렬로 표현하고 동일한 크기

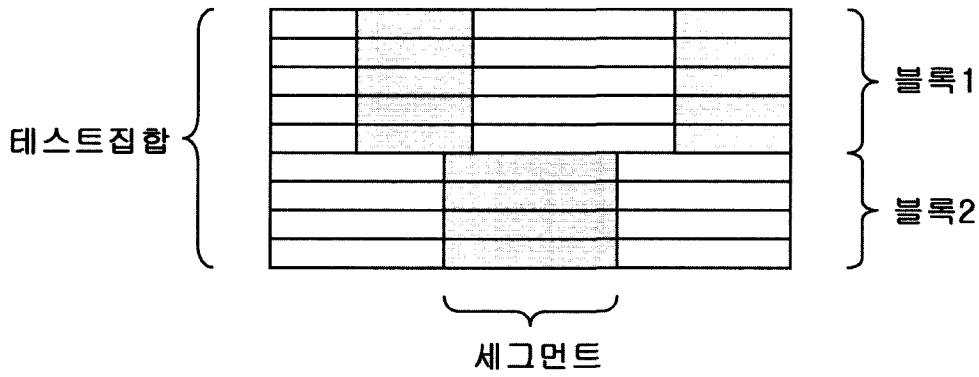

Fig. 1. Segment and block.

의 부분행렬로 나눈 후에 각 부분행렬의 각 열에 BW 변환을 적용한다. 압축율이 높은 것으로 보고된 이 방법은 그러나 디코딩 과정이 느리고 하드웨어로 구현하기에 복잡하다는 단점이 있어서, 디코딩이 소프트웨어적으로 수행되는 다운로드 단계에서나 적용할 수 있다.

3. 제안된 테스트 데이터 압축 방법

연속적인 테스트 벡터들은 일반적으로 공통의 반복적인 부분을 갖고 있다. 앞으로 이 부분을 세그먼트라고 부른다. 동일한 세그먼트를 갖는 테스트 벡터들의 집합을 블록이라고 부른다. 그림 1은 세그먼트와 블록을 보여주고 있다.

테스트 데이터 압축은 결국 중복되는 부분인 세그먼트가 최대화 되도록 하는 블록들의 집합을 찾는 과정과 같다. 이때 블록들은 서로 겹치면 안된다. 여기서 테스트 벡터들은 순서가 정해져 있으며, 모든 값은 X(don't care)없이 0과 1로 모두 지정되어있다고 가정한다.

그림 2는 압축될 수 있는 테스트 데이터 집합과 세그먼트의 예를 보여준다.

그림 2에서 위 부분에 원래의 5개 테스트 벡터가 보이는데, 음영처리된 부분은 5개의 벡터에서 변하지 않고 반복되는 부분이다. 이 부분이 압축할 대상이 된다. 제안된 압축방법에서는 겹치지 않도록 벡터들의 블록을 고려해서 테스트 집합을 분할한다. 그리고 각 블록의 세그먼트를 제거하고 나머지 비트들을 테스트 대상 회로(CUT, Circuit Under Test)에 시프트시킴으로써 압축이 이루어진다. 그림 2의 아래 부분에서 첫번째 벡터는 원래의 벡터와 동일하지만, 나머지 4개의 벡터는 반복된(음영처리된) 부분이 모두 제거되었다. 제거된 부분을 나타내기 위해서 *로 표시하였다. 매 블록의 첫번째 벡터 바로 다음에는 마스크 벡터가 위치한다. 마스크 벡터는 세그먼트의 비트 위치에는 1이, 나머지 다른 비트 위치에는 0이 나타난다. 이와 같이 블록의 처음 2개 벡터는 각 블록의 복원에 필요한 정보를 제공한다.

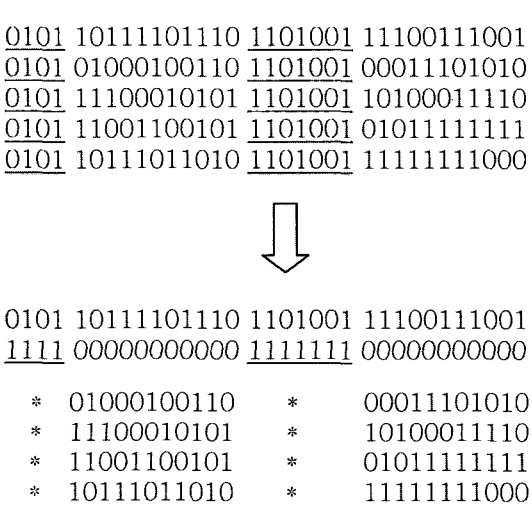

Fig. 2. Removing repetitive data parts.

마스크 벡터에서 1의 비트열은 블록 내에서 반복되는 세그먼트에 해당된다. 압축을 푸는 과정은 첫 벡터의 세그먼트(마스크 벡터의 1 비트열 부분)를 복사해서 압축된 벡터의 적절한 위치(그림 2에서 *로 표시된 부분)에 이들을 넣음으로서 간단히 수행된다.

이 압축방법은 수많은 테스트 벡터 중에서 반복되는 세그먼트를 찾아내는 방법에 의존한다. 벡터들은 가능한 블록에 따라 수많은 방법으로 분할될 수 있고, 각 분할은 서로 다른 압축율을 보이게 된다. 전체적인 압축율은 벡터들이 나뉘고 블록들이 어떻게 파악되었는가에 의해 결정된다.

테스트 벡터들의 순서 집합 $\{v_1, v_2, ..., v_n\}$은 V로 표시한다. 테스트 벡터 부분집합 $\{v_i, v_{i+1}, ..., v_j\}$은 분할 P_l로 나타낸다. P_l의 압축 후의 비트 수로 정의되는 S(i, j) 또는 S_{Pl}은 세그먼트를 확인하여 결정된다. 블록 안의 원래 테스트 벡터로 복원되기 위해서 필요한 마스크 벡터의 비트 수는 S(i, j)에 더해져야 한다. 마스크 벡터의 비트 수는 테스트 벡터의 비트 수와 동일하다. 분할된 블록들의 집합을 $P=\{P_1, P_2, ..., P_k\}$ 라고 할 때, 다음과 같은 조건을 만족하면 V에 대한 최적의 분할이라고 정의할 수 있다: 1) $V = \bigcup_{i=1}^{k} P_i$, 2) $\sum_{i=1}^{k} S_{Pi}$가 최소.

첫 분할된 벡터 집합을 $\{v_i, ..., v_k\}$라고 할 때(이 집합은 더 이상 분할되지 않음), 나머지벡터 집합인 $\{v_{k+1}, ..., v_n\}$은 다시 분할되어야 한다. $\{v_i, ..., v_n\}$에는 더 작은 최적의 분할된 집합이 포함될 수 있기 때문이다. 이것은 다음과 같은 조건에 따른 것이다: 1) 블록들은 서로 겹치지 않는다. 2) 각 블록의 테스트 벡터들의 남은 비트 수가 최소가 되어야 한다.

이 조건들을 이용하면 동적 프로그래밍(dynamic programming)에 의한 해결책을 찾을 수 있다. 테스트 벡터 집합을 $V=\{v_1, ..., v_n\}$이라 하고, 이 벡터들이 분할되었을 때 테스트 벡터 부분집합 $V=\{v_i, ..., v_n\}$에 남은 전체 비트 수를 T(i)라고 나타낸다. 또한 T={T(i)}라고 한다. 여기서 i=n, n-1,..., 1 이다.

제안된 압축방법에서, 벡터들은 T(i)를 계산함으로써 압축된다. T(1)은 결국 압축 후의 남은 비트가 된다. T를 계산하는 문제는 다음과 같이 표현할 수 있다:

$$T(i) = \min_{k \in [i, n-1]} \{S(i, k) + T(k+1)\}$$

모든 S항들은 $O(n^2)$의 시간복잡도 안에 계산이 가능

하다. T는 재귀적으로 계산될 수 있으므로, 이 경우에 T(i)값을 계산할 때 불필요한 중복된 계산이 일어날 수 있다. 예를 들어, T(4)는 T(1), T(2), T(3)을 계산하는 과정 중에 반복해서 계산된다. 이런 불필요한 계산을 없애기 위해서 뒤에서부터 T를 계산해 나간다. T(n)을 한 벡터의 비트 수로 초기화 한 후에 위의 수식을 이용하여 T(n-1), T(n-2) 등을 계산한다. 매 단계마다 모든 필요한 항들이 이미 계산되어 있기 때문에 전체 계산과정은 $O(n^2)$의 복잡도를 갖는다. 주어진 테스트 벡터 집합에 대해 한번씩만 계산이 이루어지기 때문에, 이 정도의 시간복잡도는 실용적이라고 할 수 있다.

본 논문에서 제안하는 압축 알고리즘은 그림 3과 같이 나타낼 수 있다. 테스트 벡터 집합(V)을 입력으로 받고, 분할(P)될 벡터 위치가 정수형 배열로 반환된다. 압축 알고리즘의 첫 부분에서는 V를 두 부분으로 나누는 인덱스가 계산되어 P(i)에 저장된다. 두번째 부분에서는 P를 통하여 재귀적으로 다시 분할이 선택된다.

4. 압축복원을 위한 하드웨어 설계

압축복원용 하드웨어는 ATE에서 필요로 한다. 압축된 형식으로부터 원래의 테스트 데이터를 재구성하기 위해서는 압축된 블록 뿐만 아니라 반복 세그먼트의 패턴과 값들이 각 블록과 함께 제공되어야 한다. 이 절에서는 압축된 테스트 벡터에 대한 하드웨어 디코딩을 위한 하드웨어가 제시된다.

그림 4는 제안된 디코더 구조를 보여주고 있다. 제안된 디코더는 스캔 레지스터와 마스크 레지스터, 방향선택 스위치 S로 구성된다. S 스위치는 스캔 레지스터와 마스크 레지스터 중 하나로 직렬 데이터의 흐름을 결정한다. 각 벡터에 별도의 한 비트를 추가하여 테스트 벡터와 마스크 벡터를 구분하게 된다. 이 비트가 0이면 테스트 벡터, 1이면 마스크 벡터임을 뜻한다. S는 한 압축 블록에서 첫 벡터를 스캔 레지스터로 입력되도록 한 후, 첫 벡터가 모두 시프트되면 마스크 비트를 마스크 레지스터로 입력되도록 한다. 그 후 스위치는 다시 원래의 위치로 돌아오고, 블록의 모든 비트가 시프트 될 때까지 이 상태로 머문다. 이 과정이 끝나면 순환 경로 때문에 마스크 레지스터는 원래의 값을 갖게 되고 스캔 레지스터는 공유되는 세그먼트 위치에 있는 비트를 복원한다.

5. 실험결과

제안된 압축 방법의 효율성을 확인하기 위해 ISCAS'85 벤치마크 회로에 대하여 실험을 하였으며, 실험결과는 표 1과 같다. T는 테스트 집합의 벡터 수이고, $T_{Huffman}$은 Huffman 코딩을 이용하여 압축된 테스트 집합의 벡터 수이다. $T_{proposed}$은 제안된 방법을 사용하여 압축한 테스트 집합의 벡터 수이고, FC는 압축된 테스트 집합과 테스트 집합 T에 대한 고장검출율을 나타낸다. Huffman 코딩 압축에서는 2진 데이터 벡터를 b-비트 세그먼트(b=4, 8)로 쪼개어 각 세그먼트를 해당 정수표현으로 바꾸는데, Huffman 코딩은 바로 이 정수 데이터 벡터에 적용된다. b값이 커질수록 압축효율이 좋아지지만, 디코더의 복잡도가 $O(2^b)$으로 지수함수적으로 증가하기 때문에, b를 8까지로 제한하였다.

```
Compression(V)
{
    i = n-1; k = n – 1; min = 0; T(n) = 테스트 벡터 길이;
    for (i = n-1; i >= 1; i++) T(i)를 계산한다; /* T(n-1), T(n-2), …, T(1)을 계산한다. */
    for (i = n-1; i >= 1; i++)
            for (k = i; k < n; k++) {
              tmp = S(i, k) + T(k + 1);
              if (tmp < min) { min = tmp; j = k; }
            }
    T(i) = min; P(i) = j;
    i = 1;
    while ( i < n) {
      {v_i, v_i+1, …, v_P(i)}를 새로운 분할로 저장한다;
        i = P(i) + 1;
    }
}
```

Fig. 3. Proposed compression algorithm.

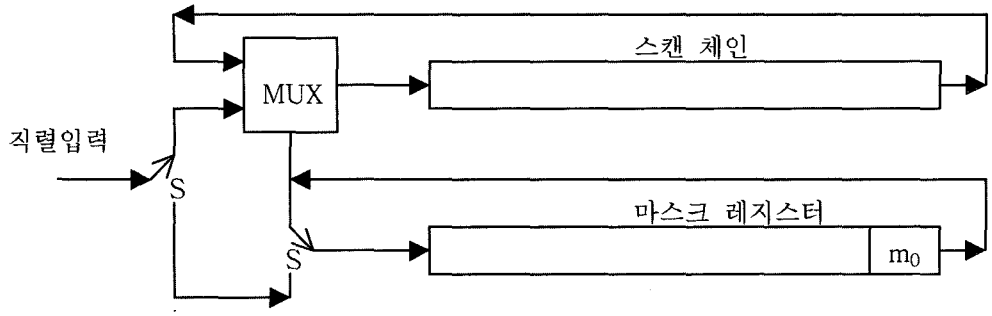

Fig. 4. Decompression decoder.

Table 1. Experimental results for ISCAS'85 benchmark circuits

회로	T	$T_{Huffman}$ (b=4)	$T_{Huffman}$ (b=8)	$T_{proposed}$	FC
C432	102	28.17	35.33	24.66	97.90
C499	140	63.00	71.96	75.16	95.43
C880	139	43.01	48.22	16.53	95.43
C1355	98	15.75	24.73	25.63	95.56
C1908	211	47.09	52.70	46.33	97.44
C2670	229	36.27	38.44	30.70	92.25
C3540	178	34.02	37.67	34.13	90.14
C5315	210	55.74	64.37	53.33	90.28
C6288	64	32.56	41.07	40.77	99.20
C7552	120	38.60	42.90	52.54	90.10

표 1을 보면, b값이 큰 경우 몇 가지 회로에 대해서는 Huffman코딩 방식이 제안된 방법보다 더 좋은 압축율을 보이고 있다. 그러나 하드웨어 오버헤드는 Huffman 코딩방식이 훨씬 많은 비용이 든다. 본 논문에서 제안한 압축방식은 전체적으로 좋은 압축율을 갖고 있으며, 하드웨어 구현에 우수한 특성을 갖고 있다.

본 논문에서 제안한 압축방식은 SoC 테스트 데이터를 압축알고리즘에 의해 사전에 압축하여 적은 용량의 메모리만으로 저장이 가능하고, 간단한 디코딩 하드웨어만으로 압축복원이 가능하다는 장점이 있지만, ATE를 통한 SoC 테스트일 경우에 한하여 적용가능하다는 단점이 있다.

6. 결 론

본 논문에서는 SoC 테스트를 위한 ATE 환경에서 사용되는 효율적인데이터 압축 방식을 제안하였다. 이 방식은 ATE의 헤드 설계 시에 도입되어 장비에서 요구되는 기억용량과 테스트 시간을 최소한으로 낮출 수 있다. 또한 off-chip과 on-chip의 압축복원 시에 간단하게 하드웨어 구현이 가능한 특징을 갖고 있다.

제안된 방법에서 압축은 테스트 데이터를 블록으로 분할하고 각 블록에서 반복되는 세그먼트를 제거하여 이루어진다. 테스트 데이터 양을 줄이기 위하여 $O(n^2)$ 시간 복잡도를 갖는 분할 방법이 제시되었다. 스캔시간에 미치는 영향이 압축율에 비례하여 감소되는 것을 볼 수 있었다. 본 논문에서는 압축방법과 함께 압축복원 하드웨어도 제시하였다. 제안된 하드웨어에서는 디코딩이 간단하고 직렬 채널로의 입력이 시프트레지스터를 통하여 수행된다. 디코더의 하드웨어 복잡도는 SoC의 스캔체인 길이에 선형적으로 비례한다.

참고문헌

[1] T. Yamaguchi, M. Tilgner, M. Ishida, D. S. Ha, "An efficient method for compression test data", International Test Conference, pp. 79-88, 1997.

[2] A. Jas and N. Touba, "Using an embedded processor for efficient deterministic testing of systems-on-a-chip", International Conference on Computer Design, pp. 418-423, 1999.

[3] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression", IEEE Trans. on Information Theory, Vol. IT-23, pp. 337-343, May 1977..

[4] J. Ziv and A. Lempel, "A compression of individual sequences via variable-rate coding", IEEE Trans. on Information Theory, Vol. IT-24, pp. 530-538, Sept. 1978.

[5] A. Chandra and K. Chakrabarty, "Test data compression for system-onchip using Golom codes", VLSI Test Symposium, pp. 113-120, 2000.

[6] A. Jas and N. Touba, "Test vector decompression via

cyclical scan chains and its application to testing core-based design", Internation Test Conference, pp. 458-464, 1998.

[7] S. W. Golomb, "Run-Length encoding", IEEE Trans. On Information Theory, IT-12, No. 4, pp. 399-401, 1996.

[8] R. Gupta and M. A. Breuer, "BALAST: A methodology for partial scan design", Proc. 19th Internation Symposium on Fault Tolerant Computing, pp. 118-125, June, 1989.

[9] J. A. Storer and M. Cohn Eds., "Fast and efficient lossless image compression", IEEE Data Compression Conference, pp. 351-360, 1993.

[10] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression", IEEE Trans. On Information Theory, Vol. IT-24, pp. 530-538, September 1978.

[11] A. Chandra and K. Chakrabarty, "System-on-a-Chip Test-Data Compression and Decompression Architecture Based on Golomb Codes", IEEE Trans. On CAD, Vol. 20, No. 3, March 2001.

[12] K. Chakrabarty and B. T. Murray, "Design of built-in test generator circuits using width compression", IEEE Trans. On CAD, Vol. 17, pp. 1044-1051, Oct. 1998.