2014년 5월 전자공학회 논문지 제 51권 제 5 호
Journal of The Institute of Electronics and Information Engineers Vol. 51, NO. 5, May 2014 http://dx.doi.org/10.5573/ieie.2014.51.5.106

논문 2014-51-5-13

Frequency-ordered 기반 FDR 테스트패턴 압축 알고리즘

(FDR Test Compression Algorithm based on Frequency-ordered)

문 창 민*, 김 두 영**, 박 성 주***

(Changmin Mun©, Dooyoung Kim, and Sungju Park)

요 약

최근 반도체 업계에서 주요 관심사로 떠오르고 있는 SOC(System-on-a-chip) 테스트는 비용 및 시간 절감을 위해 여러 종류의 FDR(Frequency-directed run-length) 기술이 제안되었다. 기존의 FDR보다 압축률을 향상시키는 EFDR(Extended-FDR)과 SAFDR(Shifted-Alternate-FDR), VPDFDR(Variable Prefix Dual-FDR)이 있다. 본 논문에서는 제안한 Frequency-ordered 방식은 FDR, EFDR, SAFDR, VPDFDR에 적용시켜 상당한 압축률 개선을 보인다. 본 기술을 사용하면 압축률을 극대화할 수 있고, 결과적으로 전체적인 양산 테스트 비용 및 시간을 크게 절감할 수 있게 한다.

Abstract

Recently, to reduce test cost by efficiently compressing test patterns for SOCs(System-on-a-chip), different compression techniques have been proposed including the FDR(Frequency-directed run-length) algorithm. FDR is extended to EFDR(Extended-FDR), SAFDR(Shifted-Alternate-FDR) and VPDFDR(Variable Prefix Dual-FDR) to improve the compression ratio. In this paper, a frequency-ordered modification is proposed to further augment the compression ratios of FDR, EFDR, SAFRD and VPDFDR. The compression ratio can be maximized by using frequency-ordered method and consequently the overall manufacturing test cost and time can be reduced significantly.

Keywords : Test Pattern Compression, FDR Algorithm

Ⅰ. 서 론

최근 SoC(System-on-a-Chip) 설계는 제조 공정이 초미세공정으로 발전하면서 수많은 IP(Intellectual Property)와 module은 하나의 칩에 집적화되고 있다. IP 기반의 SoC 설계는 검증된 설계를 재사용함으로써 비용과 개발기간은 단축 할 수 있다. 하지만 SoC 디자인의 복잡도와 동작속도가 점점 증가함에 따라 결함 발생률과 테스트 복잡도 및 시간이 증가하는 문제가 발생하게 되었다. 특히 ATE(Automatic Test Equipment)로부터 채널의 수와 테스트 데이터를 저장하는 메모리 용량과 ATE로부터 SoC로 전송되는 bandwidth가 제한적이기 때문에 테스트의 비용은 점차 증가하고 있다. 이로 인해, ATE 메모리의 사용을 줄이고, SoC로 전송되는 테스트 데이터를 효율적으로 사용하기 위해서 많은 테스트 기술이 제안되었다.

SoC 테스트 비용을 해결하기 위한 첫 번째 방법은 BIST(Built-In-Self-Test)이다. 하지만 현재 SoC는 기존 Application을 수행하기 위한 면적이 대부분을 차지하고 있기 때문에, SoC를 테스트하기 위해서 내부에

* 정회원, ** 학생회원, *** 평생회원, 한양대학교 컴퓨터공학과
(Department of Computer Science & Engineering, Hanyang University)
© Corresponding Author(E-mail: muncm8728@mslab.hanyang.ac.kr)
※ 본 연구는 2013년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업 (No. NRF-2013R1A1A2059326)
접수일자: 2013년10월21일, 수정일자: 2014년4월2일
수정완료: 2014년 4월28일

BIST를 추가시키면 칩의 소요 면적을 커지게 하는 단점이 발생한다.[1]

다른 방법으로는 테스트 데이터 압축 기술이다. ATE의 메모리 사용과 테스트 시간을 줄이고 ATE에서 SoC로 전송되는 데이터의 제약을 해결하기 위한 기술이다. 테스트 데이터 압축 기술 중에는 Huffman code, Complementary huffman code, Selective huffman code, Golomb, FDR, EFDR, SAFDR, VPDFDR이 있다.[2-8] Huffman code는 높은 압축률을 보이지만 디코더의 사이즈가 잠재적으로 커질 수 있는 단점을 갖고 있다.

따라서 본 논문에서는 기존의 정렬방식의 FDR보다 압축률이 뛰어난 새로운 정렬방식의 FDR 테스트 압축 방식을 제안한다. 본 논문의 구성은 다음과 같다. 우선 본론 Ⅱ장에서는 기존의 FDR 방식과 종류, 새로운 정렬 방식을 적용한 FDR, 디코더의 구조에 대해서 설명한다. Ⅲ장에서는 ISCAS 89' 벤치마크 회로로 실험을 통한 정량적 분석 결과를 설명하고, 마지막으로 Ⅳ장에서는 결론을 도출할 것이다.

Ⅱ. 본 론

1. 테스트 데이터 압축 기술

본 논문에서는 하드웨어 비용절감을 위해 디코더 크기를 줄일 수 있는 비손실 압축알고리즘으로 코드 기반 구조 기술인 FDR(Frequency Directed Run-length)을 사용해서 테스트 데이터를 압축한다.

가. FDR

FDR은 Run-length를 기반으로 variable-to-variable 압축방식이다.[5] Run-length는 1로 끝나는 연속된 0의 개수로 정의된다. 표 1은 FDR 알고리즘의 분석 결과이다. FDR은 크게 2 분류의 그룹을 포함한다. Prefix 그룹은 연속된 패턴의 배열이 어느 그룹에 속하는 지를 나타내고, 또 다른 Tail 그룹은 그룹 내에서의 각 패턴을 구분하기 위해서 사용되어 진다. FDR 코드는 다음처럼 생성된다. 연속된 0 패턴은 A_1, A_2, A_3,…,A_k의 그룹으로 나뉜다. 일반적인 경우 A_i의 그룹은 i와 동일한 크기를 가진 Prefix를 부여받는다. A_i의 그룹에서 A_{i+1}의 그룹으로 바뀔 때, Prefix와 Tail이 1bit씩 더 추가되기 때문에 최종적으로 부여받는 Codeword는 전 그룹보다 2bit가 더 늘어난다. 그리고 Run-length가 j인 연속 패턴이 i = log2(j+3)-1인 그룹에 매핑되며. Ai의 그룹은 2^i개수를 포함한다. FDR 알고리즘의 filling 방식은 첫 번째 vector에 Zero-filling을 한 후 그 다음 패턴을 CSR(Cyclical Scan chain Register) 과정을 통해 모든 패턴을 변환한다.

// Test Vector

1	0	1	1	0	0	0	0	0	0	0	1	0	0	1	0
0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1

그림 1. 기본 테스트 집합
Fig. 1. Original test set.

표 1. FDR Code
Table 1. FDR Code.

Group	Run length	Prefix	Tail	Codeword
A_1	0	0	0	00
	1		1	01
A_2	2	10	00	1000
	3		01	1001
	4		10	1010
	5		11	1011
A_3	6	110	000	110000
	7		001	110001
	8		010	110010
	…		…	…
	13		111	110111
…	…	…	…	…

그림 1은 ATE의 메모리에 저장되어 있는 테스트 집합이다. 그림 2는 그림 1을 FDR 알고리즘을 적용한 그림이다. 그림 2이 화살표 위에 해당되는 부분은 Run-length로 나누어진 그룹을 나타내며 화살표 아래에 해당되는 부분은 FDR 알고리즘 적용하여 최종적으로 압축된 비트 수를 나타낸다. 압축률은 (1)번식에 근거하여 나타낸다.

$$\%Compression = (1 - \frac{Compression\ data}{Original\ data}) \times 100 \quad (1)$$

```
// Test Vector  Partitioning
1    0 1    1    0 0 0 0 0 0 0 0 1    0 0 1
0 0 0 0 1    0 0 0 0 0 0 0 0 0 1    0 0 0
0 0 0 0 0 0 0 0 0 0 1    0 0 0 0 0 0 0 0 1
0 0 0 0 0 1    0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 1

// Encoded Test Vector
0 0    0 1    0 0    1 1 0 0 1 0    1 0 0
0    1 0 1 0    1 1 0 0 1 0    1 1 0 1 1
0    1 1 0 0 0 1    1 0 1 0    1 1 0 1 1
0    1 1 0 1 1 0
                                Compression = 32%
```

그림 2. FDR을 적용한 런-랭쓰 그룹과 압축된 데이터 비트 결과

Fig. 2. FDR applying group divided Run-length & compressed data bit conclusion.

```
// Test Vector Partitioning
1 0    1 1 0    0 0 0 0 0 0 1    0 0 1
0 0 0 0 1    0 0 0 0 0 0 0 0 1    0 0 0
0 0 0 0 0 0 0 0 0 1    0 0 0 0 0 0 0 1
0 0 0 0 1    0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 1

// Encoded Test Vector
1 0 0    1 0 1    0 1 0 1 1    0 0 1    0
1 0 0 1    0 1 1 0 0 0 1    0 1 1 0 1 0
1    0 1 1 0 0 0 0    0 1 0 0 1    0 1 1
0 1 0 0    0 1 1 0 1 0 1
                                Compression = 26%
```

그림 3. EFDR을 적용한 런-랭쓰 그룹과 압축된 데이터 비트 결과

Fig. 3. EFDR applying group divided Run-length & compressed data bit conclusion

나. Extended FDR

EFDR 알고리즘은 기존의 FDR 알고리즘에 기반한 알고리즘으로 0의 Run-length와 1의 Run-length을 모두 사용하여 나타낸다.[6]

표 2를 보면 기존의 FDR 알고리즘과 동일한 방식으로 Prefix와 Tail를 할당을 하지만, Run-length가 1부터 시작한다는 점과 최종적으로 인코딩되는 Codeword가 다르다. Codeword에 추가된 1bit는 0의 Run-length와 1의 Run-length 중에 어떤 Run-length로 인코드 되었는지를 나타내 준다. 만약 추가된 비트가 0을 나타내면 Codeword가 인코딩된 종류는 0의 Run-length를 가리킨다. 반대로 추가된 비트가 1을 나타내면 Codeword가 인코딩된 종류는 1의 Run-length를 가리킨다.

그림 3은 그림 1을 EFDR 알고리즘을 적용한 그림이다. 그림 3의 화살표 위에 해당되는 부분은 Run-length로 나누어진 그룹을 나타내며 화살표 아래에 해당되는 부분은 EFDR 알고리즘을 적용하여 최종적으로 인코딩된 비트 수를 나타낸다. 압축률은 (1)식에 근거하여 나타낸다.

표 2. Extended FDR Code

Table 2. Extended FDR Code

Group	Run length	Prefix	Tail	Codeword Runs of 0s	Codeword Runs of 1s
A1	1	0	0	000	100
	2		1	001	101
A2	3	10	00	01000	11000
	4		01	01001	11001
	5		10	01010	11010
	6		11	01011	11011
A3	7	110	000	0110000	1110000
	8		001	0110001	1110001
	9		010	0110010	1110010
	…		…	…	…
	14		111	0110111	1110111
…	…	…	…	…	…

다. Shift Alternate FDR

SAFDR 알고리즘 또한 FDR 알고리즘에 기반한 알

표 3. Shifted Alternate FDR Code

Table 3. Shifted Alternate FDR Code.

Group	Run length	Group Prefix	Tail	Codeword
A1	1	0	0	00
	2		1	01
A2	3	10	00	1000
	4		01	1001
	5		10	1010
	6		11	1011
A3	7	110	000	110000
	8		001	110001
	9		010	110010
	…		…	…
	14		111	110111
…	…	…	…	…

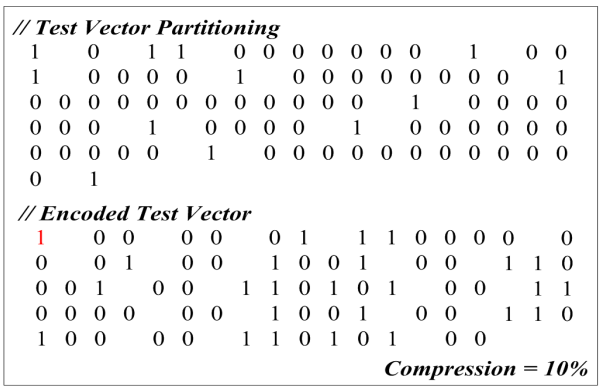


그림 4. SAFDR을 적용한 런-랭쓰 그룹과 압축된 데이터 비트 결과
Fig. 4. SAFDR applying group divided Run-length & compressed data bit conclusion.

고리즘으로 0의 Run-length와 1의 Run-length을 번갈아 가며 인코딩을 하는 방식이다.[7]

표 3을 보면 SAFDR은 기존의 FDR 알고리즘과 동일한 방식으로 Prefix와 Tail를 할당되는 최종적인 Codeword는 동일하지만, 인코딩을 하는 방식이 다르다. SAFDR 알고리즘은 EFDR 알고리즘과 동일하게 0의 Run-length와 1의 Run-length를 구분하기 위해서 한개 비트를 추가한다. 만약 추가된 비트가 0을 나타내면 Codeword가 인코딩된 종류는 0의 Run-length를 가리킨다. 반대로 추가된 비트가 1을 나타내면 Codeword가 인코딩된 종류는 1의 Run-length를 가리킨다. SAFDR은 단 1개의 비트를 추가를 하여 0의 Run-length와 1의 Run-length의 구분자 없이 인코딩을 하는 방식이다. 따라서 SAFDR의 장점은 단 1개의 비트로 압축률을 높일 수 있다는 장점이 있다.

그림 4는 그림 1을 SAFDR 알고리즘을 적용한 그림이다. 그림 4에서 화살표 위에 해당되는 부분은 Run-length로 나누어진 그룹을 나타내며 화살표 아래에 해당되는 부분은 SAFDR 알고리즘을 적용하여 최종적으로 인코딩된 비트 수를 나타낸다. 압축률은 (1)번식에 근거하여 나타낸다.

라. Variable Prefix Dual FDR

VPDFDR 알고리즘 또한 FDR 알고리즘에 기반한 알고리즘으로 0의 Run-length와 1의 Run-length을 번갈아 가며 인코딩을 하는 방식이다.[8]

VPDFDR 코드는 다음처럼 생성된다. 연속된 0 패턴은 A_1, A_2, A_3,…,A_k의 그룹으로 나뉜다. 일반적인 경우

표 4. Variable Prefix Dual FDR Code
Table 4. Variable Prefix Dual FDR Code.

Group	Run length	Group Prefix	Tail	Codeword
A1	1	10	0	100
	2		1	101
	3	01	0	010
	4		1	011
A2	5	110	00	11000
	6		01	11001
	7		10	11010
	8		11	11011
	9	001	00	00100
	10		01	00101
	11		10	00110
	12		11	00111
…	…	…	…	…

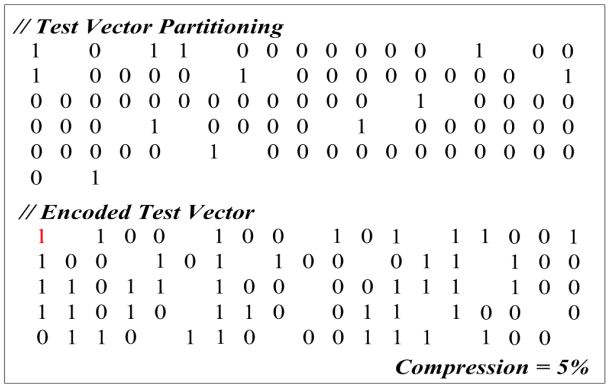


그림 5. VPDFDR을 적용한 런-랭쓰 그룹과 압축된 데이터 비트 결과
Fig. 5. VPDFDR applying group divided Run-length & compressed data bit conclusion.

A_i의 그룹은 i+1의 크기를 가진 Prefix를 부여받는다. A_i의 그룹에서 A_{i+1}의 그룹으로 바뀔 때, Prefix와 Tail이 1bit씩 더 추가되기 때문에 최종적으로 부여받는 Codeword는 전 그룹보다 2bit가 더 늘어난다. 그리고 Run-length가 j인 연속 패턴이 i = [log2(j+4)-2]인 그룹에 매핑된다. Ai의 그룹은 2^{i+1}개수를 포함하며, Codeword는 2i+1의 길이를 갖게 된다. VPDFDR의 filling 방식은 Dynamic Programming을 이용한다.

VDPFDR 알고리즘은 SAFDR 알고리즘과 동일하게 0의 Run-length와 1의 Run-length를 구분하기 위해서 한개 비트를 추가한다. 만약 추가된 비트가 0을 나타내면 Codeword가 인코딩된 종류는 0의 Run-length를 가

리킨다. 반대로 추가된 비트가 1을 나타내면 Codeword가 인코딩된 종류는 1의 Run-length를 가리킨다. 1개의 비트를 앞 단에 추가를 하여 인코딩을 하는 방식이다.

그림 5는 그림 1을 VPDFDR 알고리즘을 적용한 그림이다. 그림 5에서 화살표 위에 해당되는 부분은 Run-length로 나누어진 그룹을 나타내며 화살표 아래에 해당되는 부분은 VPDFDR 알고리즘을 적용하여 최종적으로 인코딩된 비트 수를 나타낸다. 압축률은 (1)번 식에 근거하여 나타낸다.

2. Frequency-ordered FDR (FOFDR)

기존의 FDR 정렬방식은 Run의 길이 순으로 정렬한 후 변환을 위한 그룹을 만들고 빈도수의 합으로 Prefix를 부여하게 된다. 반면, 제안하는 방식은 Run의 빈도수에 근거하여 Run을 재 정렬한 후 FDR 변환을 수행함으로써 빈도수가 높은 Run이 빈도수가 낮은 Run보다 짧은 Codeword를 갖게 되며 결과적으로 압축률을 개선한다. Theorem과 예를 통해 제안한 방식의 압축 효율성을 증명한다.

Theorem. FOFDR은 빈도수가 높은 Run-length에 짧은 Codeword를 부여함으로써 압축률을 극대화한다.

Proof.

1) FDR에서 A(A_1, A_2, …, A_n)는 기존의 데이터 비트를 나눈 Run의 길이 L(L_1, L_2, …, L_n)을 포함하며, 각각의 빈도수는 F(F_1, F_2, …, F_n)를 갖는다.

2) FDR은 2가지의 그룹으로 나누어진다. Prefix 그룹은 연속된 패턴의 배열이 어느 그룹에 속하는 지를 구분하기 위한 P(P_1, P_2, …, P_n)를 나타내며, Tail 그룹은 그룹 내에서의 각 패턴을 구분하기 위한 T(T_1, T_2, …, T_n)를 나타낸다.

3) P와 T는 L을 2^n개수를 포함하며, P가 높은 순으로 짧은 Codeword를 부여받는다.

4) FDR의 인코드 된 최종 bit는 (2)와 같다.

$$\begin{aligned} Final\,bit = & \{(f_1+f_2)\times(p_1+t_1)\}+ \\ & \{(f_3+f_4+f_5+f_6)\times(p_2+t_2)\}+ \\ & \cdots+\{(f_{n-2}+f_{n-1}+f_n)\times(p_n+t_n)\} \end{aligned} \quad (2)$$

5) A에 속한 F는 (3)식을 항상 만족하지 않는다.

$$f_{n-1}\times(p_{n-1}+t_{n-1}) > f_n\times(p_n+t_n) \quad (3)$$

6) 따라서 F_i가 높은 L_i는 F_i가 낮은 L_i보다 낮은 A_i에 속해야만 (3)식을 항상 만족한다.

예를 들어, L_1~L_6이 갖는 빈도수가 다음과 같다고 가정을 한다.(f_1=1500, f_2=400, f_3=520, f_4=380, f_5=320, f_6=200) 기존의 FDR 정렬방식으로 Final bit {(1500+400)*2}+{(520+380+320+200)*4} = 9560 bit를 얻는다. 하지만 기존의 FDR 정렬방식은 (3)을 만족하지 못한다. 하지만 FOFDR 정렬방식은 높은 빈도수부터 낮은 그룹에 포함을 하기 때문에, 그룹 A1의 f_2와 그룹 A2인 f_3의 위치가 바뀌어야한다. 따라서 final bit {(1500+520)*2}+{(400+380+320+200)*4} = 9240 bit를 얻는다. 따라서 320bit만큼의 최적의 압축 효율을 보인다.

그림 6은 Theorem에 근거한 FOFDR을 나타내며 C 언어에 기반한 Psuedo code로 표현하였다. 먼저 Test Vector 파일을 분석하여 입력값을 정의하고, 정의된 입력값에 의해 출력값을 도출한다. 입력 L은 Run-length, F는 빈도수, N은 집합 원소의 수를 나타낸다. 출력 C는 치환되는 Codeword를 나타낸다. 빈도수가 높은 F_i를 재정렬하기 위해서 F_1부터 F_n까지 비교한다. 만약 F_i 값이 F_j 값보다 작다면 F_i 값과 F_j 값을 바꿔주고, 또한 해당되는 L_i 값과 L_j 값도 바꿔준다. 따라서 재 정렬된 L, F는 해당되는 C로 매핑된다.

그림 7은 ISCAS '89 벤치마크 회로 중에서 s5379와

Algorithm : Frequecy-ordered

Input : *1. Run-length entries L*
2. Frequency of Run-length entries F from test vector
3. No. of entries allowed N
Output : *Codeword entries C*

```
For the following n decision;
    for i from 1 to n;
        for j from i+1 to n;
            if (Fi < Fj)
                replace Fi with Fj;
                replace Li with Lj;
    for i from 1 to n;
        Ci correspond to Li;
```

그림 6. Frequency-ordered 알고리즘
Fig. 6. Frequency-ordered algorithm.

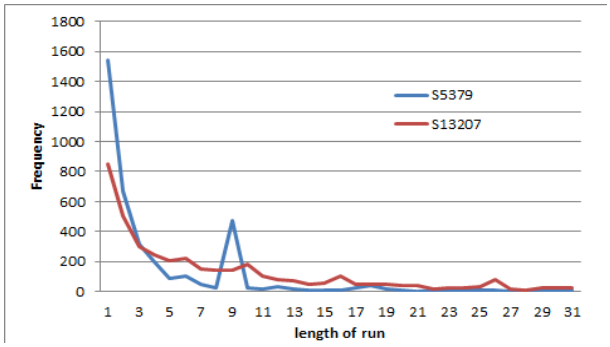


그림 7. ISCAS 벤치마크 회로 s5379, s13207의 런 길이 0의 분포도
Fig. 7. Distribution of Run-length of 0s for the ISCAS benchmark circuit s5379, s1320.

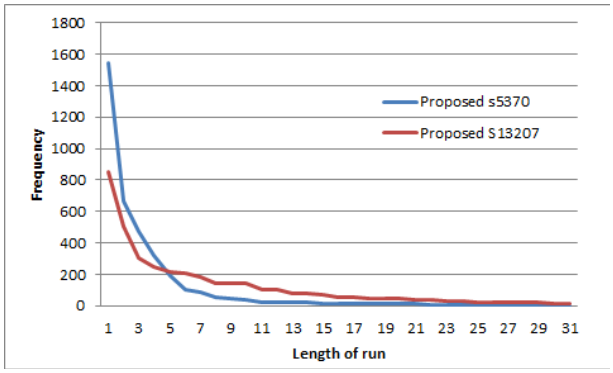


그림 8. FOFDR을 적용한 ISCAS 벤치마크 회로 s5379, s13207의 런 길이 0의 분포도
Fig. 8. Proposed FOFDR applying distribution of Run-length of 0s for the ISCAS benchmark circuit s5379, s13207.

s13207의 테스트 패턴에 기존의 FDR 정렬방식을 적용한 0의 Run-length의 분포도이다. 이 그림에서 보면 짧은 0의 Run-length에서부터 순서대로 그룹을 짓기 때문에 (3)식을 만족하지 못하며 그래프가 완만하지 않고 중간에 가끔씩 0의 Run-length가 높은 빈도수를 갖게 된다.

그림 8은 FOFDR을 적용한 0의 Run-length에 대한 분포도이다. 빈도수가 높은 0의 Run-length로 정렬을 하기 때문에 (3)식을 만족하는 완만한 그래프가 형성된다.

3. Decompression Architecture

테스트 데이터 압축 기술을 사용함에 따라 디코더는 필수적이다.

그림 9는 FDR의 디코더 구조를 나타낸다. FDR 알고리즘으로 압축된 테스트 데이터가 디코더를 통하여 T_{diff}로 변환된다. 그리고 변환된 T_{diff}는 CSR과정을 통해 모든 패턴을 복원하여 각각의 해당되는 Core를 테스트한다.

그림 10은 FOFDR 사용 시 필요한 매핑 로직과 디코더 구조이다. 일반적인 FDR을 사용 시 FSM과 카운터를 이용하여 디코더 구조를 갖게 된다. 하지만 FOFDR을 적용하게 되면, 모든 패턴에 의존적이기 때문에 부가적인 매핑 로직이 필요하여 그림 10의 디코더 구조를 갖는다. Input 값은 인코딩된 비트의 입력값을 의미하며, 매핑 로직은 재 정렬되기 전의 원래 주소값을 갖게 된다. Bit-in은 매핑 로직에서 바뀐 인코딩된 값, Sync

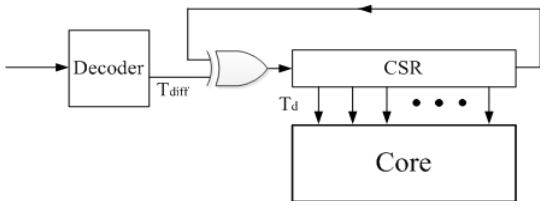


그림 9. 디코더의 구조
Fig. 9. Decompression Architecture

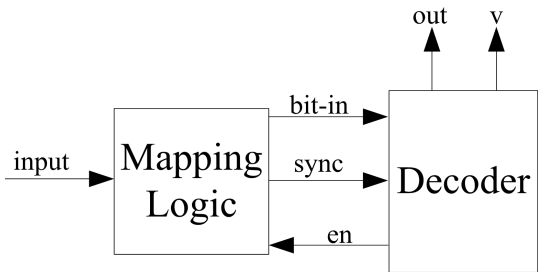


그림 10. FOFDR 사용 시 필요한 디코더와 매핑 로직
Fig. 10. The mapping logic and Decoder Used for FO.

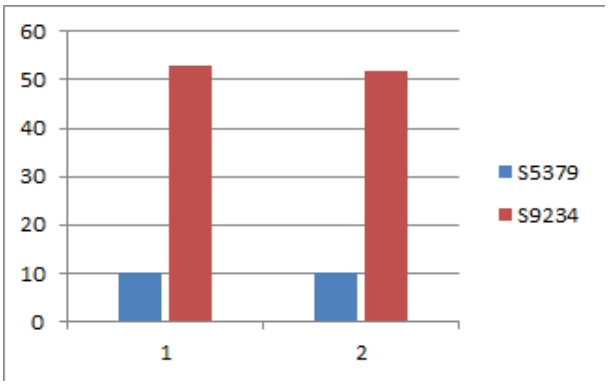


그림 11. ISCAS 벤치마크 회로 s5379, s9234에 적용한 FDR과 FOFDR의 state 수
Fig. 11. Number of state using FDR and FOFDR for the ISCAS benchmark circuit s5379, s9234.

는 디코더와 매핑 로직의 동기화, En는 디코더의 활성화, out은 디코딩된 데이터, v는 디코딩된 테스트 패턴의 이상 유무를 나타낸다.

그림 11은 ISCAS '89 벤치마크 회로인 s5379, s9234에 FDR과 FOFDR을 적용하여 state 개수 비교를 나타낸 그래프다. 기존 FDR을 적용 시는 디코더 구현에 단 10개의 state가 필요하다. 하지만 FOFDR을 적용하게 되면 모든 패턴에 의존적이기 때문에 부가적인 매핑로직이 필요하다. 따라서 기존 방법에 비하여 추가적인 state가 필요로 하지만, 이에 디코더 overhead는 높은 테스트 패턴 압출율에 의하여 상쇄될 것으로 판단한다.

Ⅲ. 실 험

본 논문에서는 제안한 새로운 정렬방식의 테스트 압축 기술의 효율성을 검증하기 위해 ISCAS '89 벤치마크 6개 회로의 테스트 패턴을 사용한 정량적 분석을 실행하였다.

표 5는 각 회로에 대한 테스트 압축 결과를 보여준다. 기존의 정렬방식으로 구현한 FDR, EFDR, SAFDR 알고리즘의 압축률과 FO로 구현한 FDR, EFDR, SAFDR, VPDFDR 알고리즘의 압축률을 보여준다.

본 논문에서 기존의 정렬방식보다 제안한 FOFDR은 적게는 1%에서 많게는 3%의 테스트 압축률을 향상시키는 실험결과를 보여준다.

표 6은 각각의 회로에 대한 인코딩 된 최종 비트수를 보여준다. CRD은 최종적으로 나온 비트수의 대비율 보여준다. 평균적으로 약 6%로의 높은 압축률 개선을 보여주며, EFDR 알고리즘 경우 제안한 FOFDR을 사용하면, 적게는 4%에서 많게는 11%로의 테스트 압축률 개선을 보여준다.

표 5. FDR과 FOFDR의 압축률
Table 5. Compression of FDR & FOFDR.

ISCAS CirCuit	% Compression Ratio											
	FDR	Proposed	CRD	EFDR	Proposed	CRD	SAFDR	Proposed	CRD	VPDFDR	Proposed	CRD
S5378	48.03	49.33	1.30	51.97	54.39	2.41	49.95	50.93	0.98	47.99	50.10	2.11
S9234	43.64	44.09	0.45	45.92	49.42	3.50	45.06	45.72	0.66	47.58	50.52	2.94
S13207	81.31	82.39	1.08	81.85	83.67	1.82	80.11	81.34	1.23	82.23	83.62	1.46
S15850	66.21	66.65	0.44	67.99	69.23	1.23	65.63	66.20	0.57	67.75	69.50	1.76
S38417	43.27	44.63	1.36	60.57	63.47	2.90	60.54	61.91	1.37	62.30	65.05	2.76
S38584	60.93	61.07	0.15	62.91	64.41	1.50	61.09	61.29	0.20	63.96	64.46	0.50

CRD : Compression Ratio Difference

표 6. FDR과 FOFDR의 최종 비트수와 압축률
Table 6. Encoded bit and Compression of FDR & FOFDR.

ISCAS CirCuit	No. of Test Vector	No. of Bits/ Vector	No. of Total Bits	% Compression Ratio & Encoded Bit											
				FDR	PFO	CRD	EFDR	PFO	CRD	SAFDR	PFO	CRD	VPDFDR	PFO	CRD
S5378	111	214	23754	12356	12048	2.56	11419	10845	5.29	11900	11666	2.01	12366	11864	4.23
S9234	159	247	39273	22148	21972	0.80	21250	19876	6.91	21588	21330	1.21	20597	19443	5.94
S13207	236	700	165200	30880	29092	6.15	29992	26977	11.18	32864	30836	6.58	29366	26953	8.95
S15850	126	611	76986	26016	25676	1.32	24643	23694	4.01	26460	26022	1.68	24833	23480	5.76
S38417	99	1664	164736	93452	91214	2.45	64962	60177	7.95	65012	62756	3.59	62107	57568	7.88
S38584	136	1464	199104	77798	77508	0.39	73853	70865	4.22	77480	77080	0.52	71761	70765	1.41

CRD : Compression Ratio Difference

PFO : Proposed Frequency-ordered

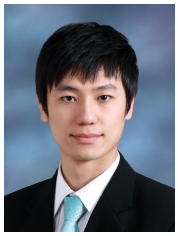
Ⅳ. 결 론

본 논문에서는 FOFDR를 사용한 테스트 데이터 압축 기술을 제시하였다. 제안하는 방법을 이론으로 정리하고 증명함으로 항상 기존 기술을 개선할 수 있다는 것을 보였다. ISCAS '89 벤치마크 회로 6개의 테스트 패턴을 사용하여 기존의 정렬방식보다 제안한 FOFDR이 압축률을 높일 수 있음을 실험을 통하여도 확인하였다. 본 기술을 사용함으로써 테스트 데이터를 압축률을 극대화하여 ATE에서 저장할 수 있는 메모리의 양을 줄이고, 또한 bandwidth의 사용 제한을 해결하여 전체적인 양산 테스트 시간 및 비용을 크게 절감할 수 있게 한다.

REFERENCES

[1] Laung-Terng Wang, Xiaqing Wen and Shianling Wu, "Using Launch-on-Capture for Testing BIST Designs Containing Synchronous and Asynchronous Clock Domains" IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, pp. 229-312, Feb 2010.
[2] Xtysovalantis Kavousianos, Emmanouil Kalligeros, Dimitris Nikolos, "Optimal Selective Huffman Coding for Test-Data Compression," IEEE Trans. on Computer, Vol. 56, no. 8, pp. 1146-1152, August 2007.
[3] Shyue-Kung Lu, Hei-Ming Chuang, Guan-Ying Lai, Bi-Ting LaiYa-Chen Huang, "Efficient Test pattern Compression Techniques Based on Complementary Huffman coding" IEEE Circuits and Systems International Conference. ICTD 2009.
[4] Chandra. A, Chakrabarty. K, "System-on-a-chip test-data compression and decompression architectures based on Golomb codes" IEEE Trans, Computer-Aided Design of Integrated Circuits and Systems, Vol, 20, pp. 355-368, Mar 2001.
[5] Chandra, A, Chakrabarty, K, "Frequency-directed run-length (FDR) codes with application to system-on-a-chip test data compression" VLSI Test Symposium, IEEE VTS 2001.
[6] El-Maleh, A.H. "Test data compression for system-on-a-chip using extended frequency -directed run-length code", Computers & Digital Techniques, IET, Vol. 2, pp. 155-163, May 2008.
[7] Chandra. A, Chakrabarty. K, "Reduction of SOC test data volume, scan power and testing time using alternating run-length codes", Design Automation Conference, 2002. Proceedings. 39th,
[8] Yang Yu, Zhiming Yang, Xiyuan Peng, "Test Data Compression Based on Variable Prefix Dual-Run-Length Code", IEEE International I2MTC, pp. 2537-2542, May 2012.

저 자 소 개



문 창 민(학생회원)
2012년 한양대학교 컴퓨터공학과 학사 졸업.
2012년～2014년 한양대학교 컴퓨터공학과 석사 졸업.
<주관심분야 : SoC 설계 및 테스트, FPGA 설계>

김 두 영(학생회원)
2004년 한양대학교 전자컴퓨터공학부 학사 졸업.
2006년 한양대학교 컴퓨터공학과 석사 졸업.
2006년～2012년 LG전자 SIC연구소 연구원.
2012년～현재 한양대학교 컴퓨터공학과 박사 과정 재학.
<주관심분야 : SoC 설계 및 테스트, Scan Design, Low Power Design>

박 성 주(평생회원)-교신저자
1983년 한양대학교 전자공학과 학사 졸업.
1983년～1986년 금성사 소프트웨어 개발 연구원.
1992년 Univ. of Messachusetts 전기/컴퓨터공학과 박사 졸업.
1992년～1994년 IBM Microelectronics 연구스텝.
1994년～현재 한양대학교 컴퓨터공학과 정교수.
<주관심분야 : 테스트 합성, Built-In Self Test, Scan Design, ATPG, ASIC 설계, 고속 신호처리 시스템 설계, 그래프 이론>