논문 2004-41SD-9-11

SoC환경에서의 저전력 테스트를 고려한 테스트 패턴 압축에 대한 효율적인 알고리즘

(A new efficient algorithm for test pattern compression considering low power test in SoC)

신 용 승*, 강 성 호**

(Yongseung Shin and Shungho Kang)

요 약

최근 반도체 칩의 집적도가 올라가고 System-on-Chip(Soc)환경이 보편화되면서 Automatic Test Equipment(ATE)를 이용한 테스트 수행시 테스트 패턴의 크기 문제와 스캔체인에서의 전력 소모문제가 크게 부각되고 있다. 또한, 테스트 패턴 크기문제를 해결하기 위해 테스트 패턴을 압축하게 되면 테스트 패턴의 소모하는 전력량이 커지게 되어 저전력 테스트를 수행하는데 어려움이 있어 두 가지 문제를 해결할 수 없었다. 본 논문에서는 이러한 문제점들을 동시에 해결하기 위해서 Run-length code를 기반으로 하여 저전력 테스트가 가능하면서 테스트 패턴의 크기도 줄일 수 있는 알고리즘을 제안하였다. 본 논문에서는 기존에 제시되었던 알고리즘과 비교·분석하는 실험을 통하여 이 알고리즘의 효율성을 보여주고 있다.

Abstract

As the design complexity increases, it is a major problem that the size of test pattern is large and power consumption is high in scan, especially system-on-a-chip(SoC), with the automatic test equipment(ATE). Because static compaction of test patterns heads to higher power for testing, it is very hard to reduce the test pattern volume for low power testing. This paper proposes an efficient compression/decompression algorithm based on run-length coding for reducing the amount of test data for low power testing that must be stored on a tester and be transferred to SoC. The experimental results show that the new algorithm is very efficient by reducing the memory space for test patterns and the hardware overhead for the decoder.

Keywords: Test pattern compression, Low power test, Run-length code

I. 서 론

최근 반도체 업계에서 주요 관심사로 떠오르고 있는 SoC(System-on-Chip)는 미리 설계된 여러 개의 IP(intellectual property)들을 이용하여 하나의 시스템을 구성한다. 따라서, SoC는 설계 시간 단축을 통한 시장 진입시간 단축등과 같은 많은 장점을 가지게 된다. 그러나, 이러한 장점들에 비해, SoC를 테스트하기 위해서는 기존의 칩을 테스트 하는 과정에 발생하는 문제이외에 여러 가지 어려움이 발생하여 이러한 장점을 잘 활용하지 못하고 있는 상황이다. 특히, 테스트 수행시 소모되는 테스트 전력 소모 문제와 기존의 칩을 테스트하기 위해 필요했던 테스트 패턴 수보다 기하급수적으로 늘어난 테스트 패턴은 SoC 테스트에 있어서 주요한 문제점으로 떠오르고 있다.

일반적으로 테스트 모드에서의 동작은 정상 모드에서의 동작보다 많은 스위칭 동작으로 인해 소모되는 전

* 정회원, LG전자 System IC 사업담당
(LG Electronics, System IC Division)
** 정회원, 연세대학교 전기전자공학과
(Dept. of Electrical and Electric Engineering Yonsei University)
※ 본 연구는 과학기술부와 산업자원부가 지원하고, 한국반도체연구조합이 총괄주관하는 System IC 2010 사업의 일환으로 수행되었음을 밝힙니다.
접수일자: 2004년1월24일, 수정완료일: 2004년9월1일

력이 대단히 크게 된다[1]. 이로 인하여 불량칩을 검증하기 위해 테스트를 수행하는 도중에 정상적인 동작을 하는 칩이 테스트 작업중에 소모되는 전력을 견디지 못하고 내부에 손상을 입고 불량칩이 되는 일이 발생하기도 한다. 또한, 최근 많이 생산되고 있는 모바일칩은 저전력에 동작하도록 설계된 칩이 대부분이기 때문에 이러한 위험은 더욱 커지게 된다. 따라서, 테스트 과정에서 소모되는 전력량을 최소화하는 부분이 크게 관심사항으로 떠오르고 있다.

테스트 수행시 소모되는 전력중에서 가장 큰 부분을 차지하는 것은 스캔체인에서 데이터들이 이동하면 발생시키는 천이에 의한 전력소모이다[5][8]. 이러한 전력소모 문제를 해결하기 위해서 소모 전력에 대한 스케쥴링 알고리즘이 제안되었고[2], 저전력 BIST(Built-in-Self-Test) 기법[3][4][5][6] 및 Full-scan 회로에 대한 전력소모 감소를 위한 새로운 방법들이 제안되었다[7][8]. 전력 소모를 줄이기 위해 시도되고 있는 여러 가지 방법들은 테스트 수행시간과도 밀접하게 연관이 있기 때문에[9][10] 상당히 중요한 문제라고 할 수 있다.

테스트 패턴의 양이 크게 증가하는 것도 중요한 문제점이라고 할 수 있다. SoC를 테스트 하기 위해서는 SoC를 구성하고 있는 각각의 IP들을 테스트하기 위한 테스트 패턴들이 해당 IP의 입력을 통해 인가되고 SoC 출력을 통해 테스트 결과를 확인할 수 있도록 해야 한다. 그러나, SoC는 많은 IP로 구성되어 있기 때문에 SoC를 테스트 하는데 필요한 테스트 패턴의 양도 그에 비례해서 커지게 된다. 일반적인 ATE (Automatic Test Equipment)는 테스트 패턴의 전송 속도, 채널 용량 및 테스트 패턴을 저장할 수 있는 메모리에 한계가 있기 때문에 테스트 패턴의 양이 많아지고 고성능의 ATE를 구입해야 하거나, 테스트 시간이 크게 증가하게 된다. 이러한 것들은 테스트 비용을 크게 증가시키게 되어 칩의 단가에도 크게 영향을 미치게 되어 SoC의 장점을 활용할 수 없게된다. 따라서, 테스트 시간 및 테스트에 필요한 메모리를 줄여야 할 필요가 있고, 이를 구현하기 위해 여러 가지 해결책이 제시되었다. 가장 먼저 제안된 것은 BIST 기법이다. BIST를 이용하면 각 IP별로 테스트가 진행되고 SoC 내부의 접근문제 또한 해결되어 테스트 시간이 해결되며, 테스트 패턴을 각 IP에서 생성하여 고장을 검출하므로 적절한 방법으로 고려되어 질 수 있다. 그러나, 기존의 IP들의 대부분은 BIST로 구현되지 않아 BIST회로를 추가로 설계하여 하드웨어를 추가하거나 BIST회로를 추가할 수 없는 경우도 생긴다는 문제점이 있어 널리 이용되지 못하고 있다.

또 다른 방법으로 제안된 것이 테스트 패턴을 압축하는 기법이다. IP에 대한 테스트 패턴을 압축함으로써 데이터의 양을 줄여 ATE의 적은 메모리에 패턴을 충분히 저장할 수 있고, ATE에서 칩으로 테스트 패턴이 전달되는 시간을 줄이는 것이다. 압축된 테스트 데이터는 Test Access Mechanism(TAM)을 통해 각 IP의 디코더로 인가되고 디코더에서는 본래의 테스트 패턴을 생성시켜 각 IP에 인가하게 된다.

테스트 패턴을 압축하기 위해서 여러 가지 알고리즘이 제안되었다. 우선, 통계적인 압축기법을 사용하여 테스트 패턴 압축 방법 등이 제안되었다[11][12][13]. 그러나, 이 알고리즘은 디코더가 비교적 크고, 다른 알고리즘에 비해서 큰 압축률을 보여주지는 못했다. 테스트 패턴을 인가할 때 각 테스트 패턴들간에 작은 수의 비트들만이 값이 다르고 나머지 비트들의 값은 동일하다는 점을 이용하여 테스트 패턴을 압축하는 방법도 제안되었다[14][15][16]. 이 기법은 적절한 방법으로 정렬된 테스트 패턴으로부터 얻어진 각 패턴들간의 패턴차를 Run-length code를 이용하여 압축하거나[14], Golomb code를 이용하는 방법[15]이 제안되었다. [15]에서는 패턴차내의 연속된 '0'의 길이를 variable-length code로 변환하여 보다 큰 압축률 얻을 수 있다. 또한 Golomb code와 유사한 FDR (Frequency-directed run-length) code를 이용하여 압축하는 방법이 제안되었으며 이 방법은 기존의 Golomb code보다 더 좋은 압축률을 보여주었다.

지금까지 이야기한 바와 같이 SoC 환경에서 테스트시 소모되는 전력에 대한 문제와 테스트 패턴이 커지며 발생하고 있는 문제는 SoC 테스트에서 매우 중요한 문제로 부각되고 있다. 그러나, 기존의 논문들은 전력 소모와 테스트 패턴 크기를 동시에 줄이지는 못하였다. 테스트 패턴을 저전력 테스트가 가능하게 생성하게 되면 테스트 패턴의 압축률이 떨어지게 되고, 기존에 제안되었던 압축 알고리즘을 적용하게 되면 전력소모가 커지게 되는 문제가 발생하게된다.

따라서 본 논문에서는 ATPG(Automatic Test Pa-ttern Generator)를 통하여 생성된 테스트 패턴을 이용하여 패턴내에 많이 있는 don't care 비트를 이용 저전력 테스트 패턴을 생성하였다. 또한 이렇게 생성된 저전력 테스트 패턴을 Run-length code와 허프만 코드를 적절히 이용하여 압축하였다. 본 논문은 다음과 같이 구성되어 있다. 우선 Ⅱ장에서는 전력소모모델과 저전

력 테스트 패턴 생성에 대해서 설명하였고, Ⅲ장에서는 생성된 저전력 테스트 패턴을 위한 압축 알고리즘 및 이를 디코딩하기 위한 디코더에 대하여 기술하였으며, Ⅳ장에서는 실험결과를 제시하였다.

Ⅱ. 전력소모 모델 및 저전력 패턴 생성 기법

1. 전력소모 모델

CMOS 회로에서 전력소모는 크게 정적 전력소모와 동적 전력소모로 구분할 수 있다. 일반적으로 전력원으로부터의 누설전류(Leakage current)나 계속적인 전류에 의한 정적 전력소모는 회로의 아웃풋 스위칭 동작에 의한 동적 전력소모에 비해서 양이 작기 때문에 본 논문에서는 다루지 않는다[17].

일반적으로 회로에서 스위칭에 의해서 발생하는 동적 전력소모는 스위칭 빈도-즉 천이 수에 비례한다. CMOS 회로의 동적 전력 소모는 다음과 정의할 수 있다.

$$P_d = 0.5 C_{load} V^2_{dd} F_c N_g \quad (1)$$

여기서 C_{load}는 출력의 Load capacitance. V_{dd}는 공급전압, F_c는 인가되는 클럭주파수, N_g는 출력단의 천이 수를 나타낸다. 보통 N_g와 F_c 변수를 제외한 나머지 부분은 회로동작에 있어서 변경할 수 없는 부분이므로 전력에 관한 변수로 이용할 수 있는 것은 N_g와 F_c뿐이다. 그러나, 클럭주파수 F_c는 테스트 수행시간과 밀접한 연관성이 있기 때문에 이를 변경하기는 어렵다고 할 수 있다. 그러므로, 천이 수 N_g가 전력 소모에 있어서 중요한 변수라고 할 수 있다.

일반적으로 스캔기반의 회로 테스트에서 가장 크게 소모되는 전력은 스캔체인에서 데이터들이 이동하면 발생되는 부분이다. 이것은 크게 두 가지로 볼 수 있는데 하나는 테스트 패턴을 입력하는 동안에 스캔인(Scan-in) 과정에서의 전력소모와 테스트 패턴을 가한 후에 나온 결과 값들을 스캔체인을 통해서 결과값을 얻는 동안에 발생하는 스캔아웃(Scan-out) 전력소모라고 볼 수 있다. 그러나, 스캔아웃 전력소모를 줄이는 것은 매우 어렵기 때문에 본 논문에서는 스캔인 전력소모만을 고려 대상으로 한다.

위에서 설명한 것과 같이 스캔데이터의 천이수를 이용하여 전력소모 모델을 만들 수 있으며 이를 이용하여 [8]에서는 WTM(Weighted Transition Metric)모델로 제안했다. 이 모델 기법은 스캔체인 내부의 천이 수를 근거로 하여 만든 것으로 비교적 간단히 전력 소모에 대한 값을 얻을 수 있다. 입력되는 패턴이 $t_1t_2t_3t_4t_5t_6$= 100000이고 t_1부터 스캔체인에 입력된다고 하자. 이 경우 t1과 t_2에서 발생하는 천이(1→0)은 스캔체인에 클럭이 입력될 때마다 총 스캔체인의 길이인 6−1=5 만큼 발생하게 된다. 이에 비해 입력패턴이 $t_1t_2t_3t_4t_5t_6$= 000001의 경우 발생하는 천이는 t_5과 t_6에서 발생하며(0→1) 이것은 스캔체인의 마지막 부분에 생기므로 총 스캔체인의 길이 인 6−5=1 만큼 발생하게 된다. 즉, 스캔체인에 입력되는 패턴중 천이가 발생하는 부분에 따라 각 위치에 가중치를 두어 계산하여야 할 필요가 생긴다. 이를 수식화한 모델이 WTM모델이다.

스캔체인의 길이를 k라고 하고, 각 스캔에 입력되는 패턴 sp를 $s_1, s_2, \cdots, s_k$라고 하자. 이때 각 스캔에 테스트 패턴들이 입력되는 동안 소모되는 전력소모를 WTM으로 표현하면 다음과 같다.

$$WTM(sp) = \sum_{j=1}^{k-1} (s_j \oplus s_{(j+1)})(k-j) \quad (2)$$

그리고 테스트 진행시 사용되는 테스트 패턴의 전체 집합 $sp_{total}=\{sp_1, sp_2, \cdots, sp_n\}$에 대한 전력소모는 다음과 같다.

$$WTM(sp_{total}) = \sum_{i=1}^{n} WTM(sp_i) \quad (3)$$

따라서 식(3)는 테스트 수행시 소모되는 전체전력이라고 할 수 있다. 여기서 n을 전체 테스트 패턴 수라고 하면 다음 식에 의해서 평균 전력소모 및 최대 전력소모를 구할 수 있다.

$$P_{avg} = WTM(sp_{total})/n \quad (4)$$
$$P_{peak} = Max(sp_n) \quad (5)$$

2. 저전력 테스트 패턴 생성 알고리즘

Ⅱ장 1절에서 언급되었듯이 테스트 수행시 소모되는 전력 소모의 대부분은 스캔체인에서 발생하며 이를 줄이기 위해서는 스캔체인에서 발생하는 천이 수를 줄여야 한다. 따라서, 천이가 일어날 수 있는 부분을 가능한 최소화할 수 있게 한다면 저전력 테스트를 위한 테스트 패턴을 생성할 수 있게 된다. 일반적으로 ATPG (Auto-matic Test Pattern Generator)를 통해서 얻은 결정 테스트 패턴을 살펴보면, 테스트 패턴에 특정값이 필요한 입력 값보다는 어떠한 값이 입력되어도 상관없는 don't care 비트가 상당히 많다는 것을 알 수 있다. 따라서, 이러한 don't care 비트에 적절한 값을 할당하여

천이 수가 적게 발생하도록 한다면 저전력 테스트가 가능한 테스트 패턴을 생성할 수 있게 된다.

don't care 비트에 저전력을 위한 테스트가 가능한 테스트 패턴을 만들기 위해 알맞은 값을 할당하기 위해서는 우선, don't care 비트의 앞의 값과 뒤의 값을 고려하여야 한다. 만일 don't care 비트 앞의 값이 don't care 비트가 아니라 특정한 값이 있다면 앞의 값과 같은 값을 don't care 비트에 할당하여 천이가 발생하는 것을 막을 수 있고, don't care 비트 앞의 값이 없을 경우 don't care 비트 뒤의 값과 같은 값을 할당하여 천이가 발생되는 경우를 방지할 수 있게 된다. 이와 같은 방법으로 테스트 패턴에서 발생할 수 있는 천이의 가능성을 최소화하고 천이가 발생하더라도 테스트 패턴의 뒤쪽에서 발생하도록 하여서 테스트 패턴 내의 천이의 수를 최소화하여 저전력 테스트가 가능한 테스트 패턴을 생성하였다.

테스트 패턴이 1XXXX000XXXX1인 경우를 예를 들어 설명해 보자. 만일, 처음에 있는 don't care 비트 값에 0을 입력한다면 가중치가 큰 천이가 생겨서 WTM 모델을 근거로 본다면 값이 상당히 커지게 된다. 따라서, 처음 don't care 비트 열은 don't care 비트열 앞에 있는 1 값을 가져서 가중치가 적은 뒷 부분에 천이가 발생하도록 해야 한다. 다음에 오는 don't care 비트열을 보면 열 앞 부분 값이 0이다. 따라서 이 don't care 비트열도 0 값을 입력하여 천이가 마지막 부분에 발생하도록 만들어 준다. 이와 같은 방법으로 생성된 테스트 패턴에 있는 don't care 비트에 값을 할당하여 최소한의 천이수를 만들어 저전력 테스트가 가능한 테스트 패턴을 생성한다.

Ⅲ. 테스트 패턴 압축알고리즘

1. 저전력 테스트 패턴에 효과적인 압축 알고리즘

기존의 논문들의 경우[13][14][15][16] don't care 비트를 각 논문에서 선택한 알고리즘에 알맞게 처리하여 테스트 패턴에 대한 압축률을 높였다. 그러나, 본 논문에서는 Ⅱ장에서 제시한 방법을 사용하므로 don't care 비트에 대해서 기존의 논문이 제시한 방법과 같이 처리할 수 없게 된다. 기존에 제시되었던 방법으로 테스트 패턴을 압축할 경우 압축률이 현저하게 낮아지게 된다. 그러므로, 기존에 제시되었던 방법을 그대로 적용하지 말고 다른 방법을 선택하여야 한다.

압축 알고리즘을 적용하기 위해서는 우선 압축할 대상의 특징을 알고 그 특징에 맞는 알고리즘을 찾아 적용해야 한다. 본 논문에서 제시한 방법으로 don't care 비트를 처리하여 생성된 테스트 패턴을 살펴 보면 '0'의 run-length와 '1'의 run-length가 대단히 많게 된다. 즉, don't care 비트의 길이만큼 '0'의 run-length 또는 '1'의 run-length가 길어지게 된다. 따라서, 이러한 성질을 활용하여 압축을 시도하면 좋은 압축률을 얻을 수 있다. 이러한 성질에 잘 맞는 비손실 압축알고리즘으로는 Run-length Encode(RLE) 알고리즘을 선택할 수 있다.

Run-length Encode(RLE)는 상당히 직관적이고 이해하기 쉬운 알고리즘이다. 압축할 데이터 시퀀스가 (x_1, x_2, …, x_M)라고 가정하자. 이 데이터를 구성하고 있는 각각의 값을 l_1, l_2, …, l_n이라고 하고, 각 l_1, l_2, …, l_n의 연속되어 나오는 값을 g_1, g_2, …, g_n이라고 하면, 압축되어진 데이터 시퀀스는 (l_1, g_1), (l_2, g_2), …, (l_n, g_n)이라고 표현하는 방법이 RLE 알고리즘이다. 여기서 $l_1+l_2+\cdots+l_n=M$이 되게 된다. 예를 들어 압축해야 하는 테이터 시퀀스가 '6A3838383838 56568686 … 86869875 75'라고 하자(예제 중간의 86은 188번 나온다고 가정한다).. 여기서 입력 데이터 시퀀스를 1 바이트씩 구분하여 보면 '6A 38 38 38 38 38 56 56 86 86 … 86 86 98 75 75'이 됨을 알 수 있다. 이렇게 나누어진 블록값을 연속되어 나온 값으로 구분해서 생각해 보면 6A는 1번, 38은 5번, 56은 2번, 86은 188번, 98은 1번, 75는 2번씩인 것을 알 수 있다. 이를 앞에서 설명한 RLE 알고리즘에 맞추어서 적용해보자. 여기서 각각의 값들이 나온 횟수를 표현하기 위해 1 바이트를 사용한다고 하면 '01 6A 05 38 02 56 BC 86 01 98 02 75'라는 결과를 얻을 수 있게 되어 94%의 높은 압축률을 얻을 수 있음을 알 수 있다.

이러한 RLE 알고리즘을 이용하기 위해서 본 논문에서는 우선 테스트 패턴을 일정한 블록으로 나누어서 각각 나누어진 블록단위로 RLE 알고리즘을 적용했다. '0'의 length와 '1'의 length가 상당히 길게 있기 때문에 각 블록단위로 나누어진 데이터에도 '0'의 length와 '1'의 length가 많이 존재한다. 예를 들어 4비트 단위로 블록을 나누었다면 '1111'과 '0000'이 테스트 패턴 안에 많이 존재하고 또한 이러한 값들이 연속되어 나오는 경우가 많다. 따라서, 이렇게 나누어진 블록단위로 RLE 알고리즘을 적용하여 압축패턴을 생성한다. 이 때 하나 더 생각해 볼 수 있는 것이 있다. RLE 알고리즘에서는 연속적으로 나오는 횟수와 같이 압축되어지는 값도 표시한다. 이를 테스트 패턴 블록에 그대로 사용한다면 '0000'

또는 '1111'도 표시하여야 한다. 하지만, 본 논문에서 RLE 알고리즘을 적용하는 경우는 '0000'과 '1111'이므로 이 값들을 모두 사용할 필요가 없게 된다. '0000' 과 '1111'대신에 이를 '0'과 '1'을 사용하여 표시하여도 상관이 없다. 디코더에서 '0'과 '1'이 들어왔을 경우, '0000' 과 '1111'로 압축을 풀어주기만 한다면 디코딩 과정에서도 문제가 되지 않는다. 본 논문에서 사용한 알고리즘에서 RLE 알고리즘을 적용한 부분과 더불어 다른 알고리즘을 적용하는 부분이 있기 때문에 단순히 '0'과 '1'을 사용하지 못한다. RLE 알고리즘이 적용된 2그룹과 그 외의 그룹-크게 3그룹으로 구분되므로 '0', '10', '11' 중에서 '0'과 '10' 또는 '0'과 '11'의 조합으로 표시할 수 있다. 물론 꼭 '0'에 '0000'을 할당할 필요는 없다. 만일 '1111'의 비율이 '0000'보다 많다면 '1111'에 '0'을 할당하여 압축률을 높일 수 있게 된다. 따라서, 본 논문에서는 위에서 설명한 RLE 알고리즘과는 다르게 압축되어지는 값의 연속 값부터 표기하는 것이 아니라 압축되는 값부터 표시하는 방법을 선택했다. 왜냐하면, 앞에서 이야기했듯이 본 논문에서 사용한 알고리즘에서는 압축되는 값을 나타내는 코드워드가 단순히 값만을 의미하는 것이 아니라 RLE 알고리즘이 적용된 것과 적용되지 않은 것에 대한 구분을 하는 역할도 수행하기 때문이다. 디코딩 작업이 진행될 때 입력되는 값들에 적용된 알고리즘을 파악해야 하기 위해서 우선적으로 알고리즘을 구분하기 위한 코드가 앞에 와야 한다.

이렇게 RLE 알고리즘으로 압축되지 않고 남은 블록들에 대해서는 다른 알고리즘을 적용해야 한다. 남은 블록들은 우선 RLE 알고리즘을 다시 적용하기에는 run-length가 너무 짧아 오히려 압축된 데이터가 원 데이터보다 커지는 단점이 생기므로 RLE 알고리즘을 다시 적용할 수는 없게 된다. 또한, [13]에서처럼 압축을 하지 않고 압축하지 않은 데이터를 그대로 사용하는 경우도 고려해 볼 수 있다. 이 경우는 압축되는 것이 아니라 RLE 알고리즘의 경우와 마찬가지로 데이터가 늘어나는 경우가 발생할 수도 있다. 그러나, 이 경우 디코더를 간단하게 구성할 수 있다는 장점이 있다. 또 하나의 방법으로 고려할 수 있는 것이 허프만 코드를 사용하는 것이다. 이 경우 압축률은 높일 수 있겠지만, 그에 비해 디코더가 좀 더 커진다는 단점이 생기게 된다. 따라서, 이들을 압축하기 위해서 본 논문에서는 허프만 코드를 사용하거나 압축하지 않고 그대로 사용하는 경우를 모두 고려하여 압축률이 높은 쪽을 선택하였다. RLE 알고리즘으로 압축된 데이터와 구분하기 위해 앞에서 이야기했듯이 '0', '10', '11'의 group bit를 할당한 후, 허프만 코드를 사용하여 압축한 경우 허프만 코드를 추가하여 코드워드를 할당한다. 허프만 코드로 압축을 한다고 압축률이 크게 올라간다는 보장은 없지만 허프만 코드가 전혀 압축을 하지 못한 최악의 경우는 RLE 알고리즘으로 압축되지 않은 데이터를 압축하지 않고 그대로 사용하는 경우와 비슷하게 된다. RLE 알고리즘으로 압축되지 않은 데이터가 빈도수에 따라서 압축이 가능할 수도 있으므로 허프만 코드를 사용하여 조금이나마 압축률을 높일 수 있다. 실제로 이 방법을 적용할 때 각각의 상황에 알맞게 선택하여 적용할 수 있다.

본 논문에서 제시한 알고리즘을 살펴보면 RLE 알고리즘과 이를 적용하지 않은 부분과 구분을 위해 '0', '10', '11' 또는 '1', '00', '01'과 같은 group bit가 필요하다. 본 논문의 실험에서는 각 group bit를 '0'의 length와 '1'의 length 중 더 많은 양의 데이터가 있는 쪽에 한 비트의 group bit를 할당하여 압축률을 극대화시켰다. 지금까지 본 논문에서 제시한 알고리즘으로 코드워드를 작성하면 표 1과 같다. 표 1은 데이터 블록을 4비트 단위로 설정하였고 압축되는 데이터의 연속된 값을 표시하기 위해 2비트를 할당하였다.

그림 1-a)와 같은 테스트 패턴이 있다고 한다면 표 1을 근거로 해서 압축된 데이터를 생성하면 그림 1-b)와 같은 패턴이 생성되어 64비트의 데이터가 37비트로

표 1. 제안된 알고리즘의 코드워드
Table 1. Codeword of proposed algorithm.

Test Pattern	Group bit	Tail bit	Codeword
0000	0	00	000
00000000		01	001
000000000000		10	010
0000000000000000		11	011
1111	10	00	1000
11111111		01	1001
111111111111		10	1010
1111111111111111		11	1011
0001 0010 …	11	Huffman code or Non-compressed	'11'+ Huffman / Pattern

a) 저전력 테스트 패턴
0000 0000 0000 0000 1111 1111 1111 0000
1111 1010 1010 1111 1111 0000 0000 1111

b) 압축된 테스트 패턴
011 1010 001 1000 111010 111010 1001 001 1000

그림 1. 제안된 알고리즘 예제
Fig. 1. Example of proposed algorithm.

압축됨을 알 수 있다.

2. 제안된 알고리즘을 위한 디코더

저전력 테스트를 위한 테스트 패턴은 3.1절에서 제안한 알고리즘을 통해서 압축된 후에 ATE 메모리에 저장된다. 저장된 패턴은 디코더를 거쳐 테스트 패턴으로 다시 복원되어야 한다. 디코더는 일반적으로 ATE 내에 추가되어지거나 SoC내부의 각 IP의 입력단 쪽에 추가된다.

본 논문에서는 SoC내에 디코더가 내장되는 형태를 고려하여 하드웨어를 구성했다. 본 논문에서 제안한 압축 알고리즘을 위한 하드웨어는 기본적으로 Finite State Machine(FSM), 4비트 카운터 그리고 컨트롤러로 구성된다. FSM은 압축된 데이터를 받아서 RLE 알고리즘으로 압축된 부분과 압축되지 않은 부분을 구분해주고 RLE 알고리즘으로 압축된 부분은 '0'의 Run-length와 '1'의 Run-length로 구분해준다. RLE 알고리즘으로 압축된 부분은 카운터를 이용해서 압축을 풀어주게 된다. 만일, RLE 알고리즘으로 압축되지 않은 부분을 허프만 코드를 이용해서 추가로 압축할 경우 FSM에 State를 추가하여 하드웨어를 구성한다. 이 경우 하드웨어의 크기가 더 커진다는 문제점이 발생하지만 압축률이 5~10% 정도 향상되는 장점이 있다. 이러한 장·단점을 고려하여 본 논문에서 제안한 압축알고리즘을 적용시킬 하드웨어의 상황에 알맞은 방법을 선택할 수 있다. 허프만 코드를 이용하지 않고 테스트 데이터를 있는 그대로 사용한 경우, RLE 알고리즘으로 압축되지 않은 부분에 디코딩 작업은 카운터와 간단한 컨트롤러를 이용하여 수행하게 된다.

그림 2는 4비트 블록 단위로 테스트 패턴을 구분하고 본 논문에서 제안한 알고리즘으로 압축된 테스트 패턴을 디코딩하기 위한 디코더이다.

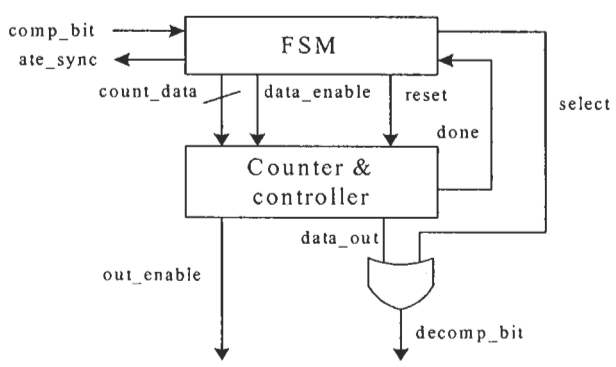

그림 2. 제안된 알고리즘을 위한 디코더
Fig. 2. Decoder for proposed algorithm.

FSM을 통해서 압축이 풀린 테스트 패턴들은 data_out을 통해서 OR 게이트로 입력된다. 이 때, data_out을 통해서 나오는 값은 '0'으로 고정된다. 테스트 패턴의 값은 '0' 또는 '1'이기 때문에 각 값들이 나오는 횟수는 카운터와 컨트롤러를 통해서 조절하고, 실제 스캔체인으로 입력되는 값들-즉, decomp_bit로 나오는 값들에 대한 조절은 FSM에서 나오는 select 신호를 통해서 선택하게 된다. 압축이 풀릴 값이 RLE 알고리즘으로 압축되지 않은 값들이거나 RLE 알고리즘으로 압축된 '0'의 run-length인 경우 select 신호에서 '0'값을 내보내서 decomp_bit으로 '0'값이 전달되게 된다. 만일 '1'의 run-length에 대해서 RLE 알고리즘을 압축된 경우 FSM의 select 신호에서 '1'의 값을 내보내서 decomp_bit로 '1'값이 전달되게 된다.

ATE와 스캔체인과의 신호를 맞추기 위해서 out_enable신호와 ate_sync신호를 추가하였다. out_en able 신호는 스캔클럭과 AND 게이트로 입력되어 스캔체인의 쉬프트 동작을 조절하게 된다. 즉 디코더가 압축된 값을 입력받고 입력된 값이 압축된 값인지 아니면 압축되지 않은 값인지를 체크하는 동작 중에는 디코더를 통해서 나오는 값이 스캔체인에 입력되어서는 안 된다. 따라서, out_enable 신호를 이용해 gated scan clock을 형성해서 이러한 부분을 조절한다. 압축된 데이터가 풀리는 경우, out_enable의 값을 '1'로 내보내서 스캔클럭이 스캔에 전달되게 하여 스캔체인에 테스트 패턴이 전달되고, 입력된 값으로 FSM이 디코딩과정에 필요한 값을 처리하고 있는 동안에는 out_enable 값이 '0'을 내보내서 스캔클럭을 비활성화시킨다. ate_sync신호는 디코더가 디코딩 동작을 하고 있을 때, ATE에서 압축된 데이터가 디코더로 입력되는 것을 방지하는 역할을 한다. 디코더가 ATE에서 받은 압축된 데이터를 바탕으로 디코딩 작업을 수행하고 있을 때 ate_sync 신호는 '1'값을 ATE에 전달하여 ATE에서 더 이상의 압축된 값이 전달되는 것을 막아 잘못된 압축 데이터가 전달되는 것을 방지한다. 디코더가 디코딩 작업이 다 끝나서 새로운 압축 데이터를 입력받을 때가 되면, ate_sync 신호는 '0'값을 ATE로 전달하여 새로운 값들이 디코더로 전달되어 새로운 압축 데이터를 가지고 디코딩 작업을 준비하게 된다.

FSM을 컨트롤하기 위해서 카운터와 컨트롤러의 done 신호를 이용한다. 각 FSM의 상태(State)는 RLE 알고리즘으로 압축된 데이터를 푸는 경우를 제외하고는 각 상태에서 머물러서 같은 동작을 수행해야하는 상황

이 동일하다. 예를 들어 RLE 알고리즘으로 압축되지 않은 압축 데이터를 처리하는 경우 처음 2비트를 읽고 난 후, 이후에 입력되는 4비트의 값을 그대로 data_out에 전달하는 동작만을 수행하게 된다. 따라서, comp_bit를 통해서 입력되는 값을 그대로 decomp_bit로 내보내는 상태를 4번 동작하고 초기상태로 돌아가면 된다. 이 4번의 동작을 카운터를 통해서 체크하고 4번의 동작이 종료하면 다음 상태로 넘어가라는 done 신호를 주어서 초기상태로 이동하게 된다. RLE 알고리즘을 압축된 경우를 살펴보자. 첫 비트를 읽고 다음의 2비트의 값들이 압축된 데이터라는 것이 판단된다면(본 논문의 예제에서는 첫 비트가 '0'인 경우), 2비트 값을 읽고 읽어들인 2비트 값을 근거로 하여 카운터에서 필요한 만큼 '0'의 값을 data_out으로 내보내주고 난 후, 초기 상태로 돌리기 위한 done 신호를 FSM에 입력하게 된다. 이것은 2비트를 읽고 나서 RLE 알고리즘을 압축된 경우(본 논문의 예제에서는 '10'의 경우)도 마찬가지로 이후 2비트 값을 바탕으로 해서 필요한 만큼 카운터를 동작시키고 값을 내보내게 된다. 이 때 count_data의 결과값이 유효한 경우를 알려주기 위해서 data_enable신호를 사용한다. reset 신호는 FSM이 각 상태를 이동할 때마다 카운터 값을 초기화 하기 위해 사용된다.

그림 3은 표 1을 기반으로 하여 RLE 알고리즘을 적용하고 RLE 알고리즘을 적용할 수 없는 데이터는 압축하지 않고 그대로 사용한 데이터의 디코더에 대한 상태 천이도이다.

S1에서 S2, 또는 S3으로 가는 과정을 통해서 디코더는 들어온 입력에 대한 그룹을 판단한다. 표 1와 같이 '0000' 블럭에 대해서 RLE 알고리즘을 적용한 경우 group bit에 '0'을 할당하고, '1111' 블럭에는 '10'을, RLE 알고리즘으로 압축하지 않은 경우 '11'을 group bit로 정했다고 가정하자. 만일 입력된 비트가 '0'인 경우 상태는 S1에서 S2으로 이동하면서 앞으로 들어올 비트가 '0000'에 대한 압축 코드워드라는 것을 알고 이를 count_data에 저장하여 '0000'을 코드워드가 의미하는 만큼 값을 스캔체인으로 전달하게 된다. '1'의 값이 들어올 경우 S1에서 S3로 상태를 옮긴 후 그 다음에 들어오는 비트값으로 '1111'에 코드워드인지 아니면 RLE 알고리즘으로 압축되지 않은 것에 대한 값이 들어오는지를 판단하게 된다. 또한 '0000'에 대한 그룹인지 '1111'에 대한 그룹인지 확인이 되는 순간 select값이 결정되면서 RLE 알고리즘으로 압축된 데이터가 풀려서 data_out으로 나오는 값을 '0000' 또는 '1111'로 결정해 준다. Group bit를 통해서 입력될 데이터에 대한 그룹이 명확하게 정해지고 나면 각각의 상태를 이동하면 그에 알맞은 데이터를 버퍼로 내보내게 된다.

그림 4는 본 논문에서 제안한 알고리즘을 위한 디코더의 동작을 검증하기 위해 사용된 블록도이다. ATE블록에 압축된 데이터를 넣고 이를 디코더를 거쳐서 스캔체인 하드웨어 입력되는 테스트를 하였다. 입력된 압축 데이터는 '100100111001000'를 사용하였다. 이를 의미있는 블록단위로 나누어보면 '1001 001 11001 000'으로 구분할 수 있다. 이 압축된 데이터를 받은 디코더는 스캔체인으로 테스트 패턴을 전달하게 된다. 처음 '1001'은 처음 2비트의 '10'으로 '1111'블록에 대한 압축코드임을 알 수 있다. 또한, 나머지 2비트의 값 '01'은 두 번 블록이 반복되는 것을 의미한다. 따라서, '1001'은 '1111' 블록이 두 번 반복되서 나오게 된다. 같은 방법으로 각각의 블록을 디코딩해보면, 블록 '001'은 '0000'의 블록이 2번 반복되고, '111001'은 '1001'을, '000'은 '0000'블록을 한 번 반복하는 것을 의미한다. 디코더와 ATE 모델 및 스캔체인 모델은 verilog를 이용하여 기술하였고, 각 하드웨어들의 합성은 Synopsys사의 design compiler를 이용해 LSI 10K 라이브러리를 이용하여 수행하였다. 합성된 디코더의 크기는 4장 실험결과에 다른 디코더와

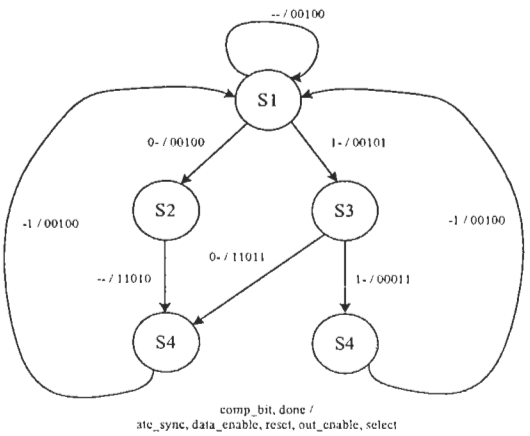

그림 3. 제안된 디코더의 상태 천이도
Fig. 3. State diagram of proposed decoder.

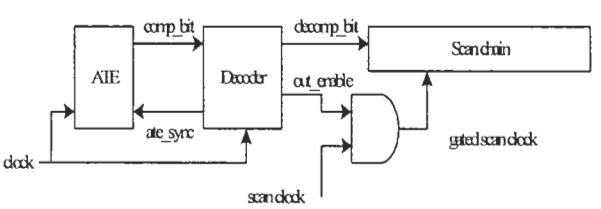

그림 4. 제안된 디코더 실험 블록도
Fig. 4. Block diagram of proposed decoder.

비교하였다.

Ⅳ. 실험 결과

본 논문은 Automatic Test Pattern Generator (ATPG) 프로그램을 이용하여 ISCAS85 및 ISCAS 89회로에 대한 결정 테스트 패턴을 얻었다. ISCAS89의 각 회로는 full-scan을 가정하였다. 본 실험은 CPU Pentium4, 메모리 256MB인 환경에서 Linux(RedHat8) 상에서 수행되었다.

테스트 전력소모에 대한 실험을 하기 위해서 비교대상으로 두 가지 패턴을 생성하였다. 우선, don't care 비트에 모두 '0'을 넣은 패턴과 '1'을 넣은 패턴을 생성하여 2장에서 설명한 WTM 모델에 근거하여 천이수를 비교하였다. 각 파워의 비교치는 다음 식에 의해서 계산되었다.

$$P^{p}_{avg}Reduction = \frac{P^{o}_{avg} - P^{p}_{avg}}{P^{o}_{avg}} \quad (6)$$

$$P^{p}_{peak}Reduction = \frac{P^{o}_{peak} - P^{p}_{peak}}{P^{o}_{peak}} \quad (7)$$

일반적으로 전력소모에 대한 수치는 다음의 두 가지 수치가 주로 사용된다. 평균소모전력(Average power)는 테스트 모드 진행중에 소모되는 전체 전력소모량을 의미한다. 이 수치는 테스트하려는 회로가 테스트 중에 회로에서 발생되는 열로 인해 실리콘이나 와이어 또는 패키지에 구조적인 문제점이 발생하지 않기 위해 어느 정도의 전력을 견뎌야 하는지를 의미한다[19]. 최고소모전력(Peak power)는 테스트 모드 진행중에 가장 크게 소모되는 전력소모량을 의미한다. 이 수치는 회로의 전기적인 한계와 시스템 패키지가 견뎌야할 최대 전력소모량을 나타낸다. 테스트 진행시 회로가 견딜 수 있는 전력소모량보다 순간적으로 더 많은 전력소모가 발생하면 회로가 정상적으로 동작할 수 있다는 보장을 할 수 없게 된다. 물론 회로가 견딜 수 있는 전력소모량보다 커진다고 해서 바로 회로가 고장나거나 하지 않고, 어느 정도 지속되는 평균전력소모 보다 높은 전력소모에 의해 고장날 수도 있으나 본 논문에서는 고려하지 않는다. 위에서 언급한 전력을 측정하는 순간적이라는 시간이 주관적일 수 있으나, 일반적으로 한 클럭을 나타낸다[19].

위에서 설명한 각 전력소모의 비교치는 표 2과 표 3에 표시하였다. s9234, s13207, s15850의 Ppeak가 동일한 것은 이들 테스트 패턴중 don't care 비트가 전혀 없는 테스트 패턴이 가장 많은 천이를 가지고 있기 때문에 본 논문에서 제시한 방법으로는 Ppeak를 최소화할 수 없었기 때문이다.

표 2. Don't care 비트에 '0'을 넣은 패턴과 전력소모량 비교

Table 2. Result for test power comparison with mapping don't care bits to all '0'.

Circuits	'0' bit		Proposed Algorithm			
	P^{o}_{avg}	P^{o}_{peak}	P^{p}_{avg}	P^{p}_{avg}(%) Reducition	P^{p}_{peak}	P^{p}_{peak}(%) Reducition
c2670	952	3537	421	55.78	1961	44.56
c3540	228	601	168	26.32	517	13.98
c5315	561	2258	186	66.84	899	60.19
c6288	54	108	52	3.70	97	10.19
c7552	1797	6020	1065	40.73	5002	16.91
s5378	952	2744	518	45.59	2034	25.87
s9234	5246	18094	3892	25.81	18094	0
s13207	20154	134340	18794	6.75	134340	0
s15850	30827	104386	29448	4.47	104386	0
s35932	3840	10441	1095	71.48	5222	49.99
s38417	4472	23868	1963	56.10	20344	14.76

표 3. Don't care 비트에 '1'을 넣은 패턴과 전력소모량 비교

Table 3. Result for test power comparison with mapping don't care bits to all '1'.

Circuits	'1' bit		Proposed Algorithm			
	P^{1}_{avg}	P^{1}_{peak}	P^{p}_{avg}	P^{p}_{avg}(%) Reducition	P^{p}_{peak}	P^{p}_{peak}(%) Reducition
c2670	984	5529	421	57.22	1961	64.53
c3540	237	573	168	29.11	517	9.77
c5315	370	1634	186	49.73	899	44.98
c6288	60	119	52	13.33	97	18.49
c7552	1969	6466	1065	45.91	5002	22.64
s5378	808	2790	518	35.89	2034	27.10
s9234	4719	18094	3892	17.52	18094	0
s13207	24554	134340	18794	23.46	134340	0
s15850	32833	104386	29448	10.31	104386	0
s35932	2470	7293	1094	55.71	5222	28.40
s38417	4537	38321	1963	56.73	20344	46.91

테스트 패턴에 대한 압축결과는 표 4에 표시하였다. Ⅲ장에서 기술한 방법으로 생성된 저전력 테스트 패턴을 이용하여 본 논문에서 제시한 압축 알고리즘을 이용하여 테스트 패턴을 압축하였다. 압축률은 다음 식에

표 4. 테스트 패턴 압축 결과
Table 4. Result for test pattern compression.

Circuits	Statistical code(%)[13]	Golomb code with Low power(%)[22]	Proposed algorithm(%)
c2670	61.3	N/A	83.84
c3540	40.6	N/A	52.01
c5315	63.6	N/A	83.24
c6288	44.6	N/A	66.69
c7552	56.4	N/A	72.95
s5378	62.0	40.70	83.78
s9234	67.1	43.34	41.93
s13207	83.5	74.78	66.87
s15850	78.9	47.11	46.71
s35932	N/A	N/A	77.16
s38417	53.6	44.12	77.09

표 5. 디코더 크기 비교 결과
Table 5. Result for the size of decoder.

Compression algorithm	Area overhead(LSI 10K libray)		
	Group size		
	4	8	16
Statistical code(%)[13]	349	587	900
Golomb[15]	125	227	307
Proposed algorithm	**142**		

의해서 계산되었다.

$$Comp.\ Ratio = \frac{Original\ Pat. - Compressed\ Pat.}{Original\ Pat.} \quad (8)$$

표 4에 있는 압축결과중 기존의 테스트 패턴 압축에 대한 알고리즘을 저전력 테스트에 대한 고려를 추가하여 압축할 경우 압축률이 많이 떨어지게 되어 제안된 알고리즘에 대한 압축률 평가를 위해서 각각의 알고리즘이 제안한 방법을 그대로 사용한 결과를 이용하였다. 또한, 표 5는 기존에 제안되었던 압축 알고리즘의 디코더에 관한 크기 비교표[21]이다. 저전력을 고려한 Golo-mb code의 경우, [22]에 디코더 크기에 대한 결과가 없으나, 압축 알고리즘 자체에 변화가 없어서 [21]에서 제시한 결과를 사용하였다. 이전에 제안되었던 알고리즘의 결과들과 비교하기 위해 동일한 LSI 10K 라이브러리와 Synopsys사의 Design compiler를 이용하여 합성하였다.

각각의 알고리즘과 본 논문에서 제안한 알고리즘에 대한 비교를 해보자. 우선, 저전력 테스트를 고려한 Golomb code[22]의 경우, 제안된 알고리즘 압축률과 유사한 수준을 보여준다. 그러나, Golomb code의 경우, Cyclical Scan chain Register(CSR)를 기반으로 한 압축 알고리즘이므로 표 5에 제시되고 있는 하드웨어의 크기를 고려해 비교해 보면 Golomb code의 경우 디코더 크기가 본 논문에서 제시한 알고리즘과 유사한 수준이지만 Golomb code에 꼭 필요한 CSR의 오버헤드까지 추가한다면 그 크기는 본 논문에서 제시한 하드웨어의 크기보다 상당히 커지게 될 것이다. CSR의 경우 스캔체인의 길이만큼 필요하고, 경계 스캔셀이나 다른 코어의 스캔체인을 이용한다고 하더라도 그 크기는 무시할 수 없을 정도이므로 실질적인 하드웨어 오버헤드는 본 논문에서 제안한 것이 훨씬 적다는 것을 알 수 있다. 따라서, 본 논문에서 제시한 알고리즘의 경우 저전력을 고려한 Golomb code보다 많은 회로에서 유사하거나 높은 압축률을 보이며, 디코더의 크기도 상대적으로 작음을 확인할 수 있다.

다음으로 통계적인 압축방법의 경우를 살펴보자. 몇 개의 회로에서 압축률이 제안한 알고리즘보다 높게 나온다. 제안한 알고리즘은 '0000' 또는 '1111' 블록을 중심으로 테스트 패턴이 압축되지만, 통계적인 압축방법은 다양한 블록 가운데 최대 빈도수를 보이는 블록을 중심으로 압축이 된다. 따라서, 일부 회로에서 통계적인 압축방법의 압축률이 높은 것은 '0000' 또는 '1111' 이외의 다른 블록이 빈도수가 높기 때문에 제안한 알고리즘보다 압축률이 높게 나온다. 그러나, [13]에서 제시한 방법은 저전력 테스트를 고려하지 않은 방법이고 디코더 크기는 본 논문이 상대적으로 작아 테스트 패턴 압축에 대한 알고리즘으로는 본 논문에서 제시한 방법이 더 우수하다고 할 수 있다. 실험결과들 중에 s9234와 s15850의 경우, don't care 비트가 다른 회로들에 비해서 매우 적게 나왔기 때문에 저전력에 대한 감소률도 낮고 압축률도 표 4에 나온 것과 같이 낮게 나왔다.

V. 결 론

SoC환경에서의 테스트 수행시 문제점으로 부각되고 있는 테스트모드에서의 전력 소모 문제점과 테스트 패턴의 크기 문제를 해결하기 위해 기존의 논문들은 두 가지 문제에 대해 어느 한 부분에 대해서만 문제 해결을 위한 방법들을 제시하였으나, 본 논문에서는 두 가지 문제를 해결하기 위한 새로운 알고리즘을 제시하였다.

테스트 수행시 소모되는 전력중 가장 큰 비중을 차지하고 있는 스캔체인에서의 전력소모를 줄이기 위해서는 스캔체인에서 테스트 패턴의 이동시 발생하는 천이가

적게 발생하도록 해주어야 한다. 이러한 스캔체인에서의 천이는 테스트 패턴이 스캔체인으로 들어가는 스캔인 동작에서 발생하는 스캔인 전력소모와 테스트 수행 후 스캔체인에 있는 결과값을 외부로 보낼 때 발생하는 스캔아웃 전력소모로 볼 수 있다. 여기서 스캔아웃 전력소모에 대해서 저전력을 구현하기가 매우 어렵기 때문에 본 논문에서는 스캔인 전력소모를 최소화하기 위해 입력되는 테스트 패턴에 천이가 발생하는 경우가 최소화되도록 don't care 비트에 값을 할당하였다. 본 논문에서 제시한 방법을 통해 don't care 비트에 단순히 한 가지 값이나 랜덤하게 값을 할당하는 경우보다 전력소모가 줄어드는 것을 WTM 모델을 가지고 실험을 통해 확인할 수 있었다.

또한, 저전력 테스트를 위해 생성된 테스트 패턴은 기존에 제시되었던 테스트 패턴 알고리즘을 그대로 적용하기 어렵기 때문에 이를 해결하기 위해서 본 논문에서는 테스트 패턴을 일정 비트 단위로 블록화 시킨후 블록 단위에 대해서 Run-length Encode(RLE) 알고리즘을 적용하였다. 저전력 테스트를 위해 생성된 테스트 패턴은 천이 수가 매우 적기 때문에 패턴 내에 '0'의 run-length나 '1'의 run-length가 매우 많기 때문에 RLE 알고리즘을 통해서 높은 압축률을 얻을 수 있었으며, 또한, 기존에 제시된 논문들보다 간단한 디코더도 구성할 수 있다는 장점을 실험을 통해 보여 주었다. 본 논문에서는 2, 3장에서 제시한 방법을 사용하여 압축된 테스트 패턴이 기존에 제시된 여러 알고리즘보다 효율적이 우수하다는 것을 검증하였다.

참 고 문 헌

[1] Y.Zorian, "A distributed BIST control scheme for complex VLSI devices," Proc. of IEEE VLSI Test Symp., 1993, pp4-9.

[2] R.M. Chou, K. K. Saluja, and V.D. Agrawal, "Scheduling test for VLSI systems under power constraints," IEEE Trans. VLSI Syst., vol 5, pp.175-185, June 1997.

[3] S.Wang and S.K. Gupta, "LT-RTPG: A new test-per-scan BIST TPG for low heat disspa -tion," Proc. of Int. Test. Conf., 1999, pp.84-94.

[4] S.Gerstendörfer and H.J. Wunderlich, "Minimized power consumption for scan-based BIST," Proc. of Int. Test. Conf., 1999, pp.77-84.

[5] P. Girard, L. Guiller, C. Landrault, and S. Pravos -soudovitchm "A test vector inhibiting technique for low energy BIST design," Proc. of IEEE VLSI Test Symp., 1999, pp.407-412.

[6] F.Corno, M. Rebaudengo, and M. S. Reorda, "Low power BIST via nonliear hybrid cellular automata," Proc. of IEEE VLSI Test Symp., 2000. pp29-34.

[7] S. Wang and S. K. Gupta, "ATPG for heat dissipation minimization during scan testing," Proc. of Design Automation Conf., 1997, pp.614 -619.

[8] R. Sankaralingam, R.R. Oruganti, and N.A Touba, "Static compaction techniques to control scan vector power dissipation," Proc. of IEEE VLSI Test Symp., 2000, pp.35-40.

[9] K. Chakrabarty, "Optimal test access architecture for system-on-chip," ACM Trans. Design Auto -mation Electron. Syst., vol. 6, pp. 26-49, Jan. 2001.

[10] K. Chakrabarty, "Design of system-on-chip test architectures under place-and-route and power constraints," Proc. of IEEE/ ACM Design Auto -mation Conf., 2000, pp.432-437.

[11] Iyengar, V., Chakrabarty, K., Murray, B.T. "Built-in self testing of seq. circuits using pre -computed test sets,"Proc. of VLSI Test Sympo -sium, 1998. pp.418-423.

[12] Hamzaoglu, I., Patel, J.H. "Deterministic test pattern generation techniques for sequential cir -cuits," Computer Aided Design, 2000. ICCAD -2000. pp.538-543

[13] Jas, A. Ghosh-Dastidar, J. Touba, N.A., "Scan vector compression/decompression using statis -tical coding," Proc. of VLSI Test Symposium, 1999. pp.114-120.

[14] Jas, A., Touba, N.A., "Test vector decompre -ssion via cyclical scan chains and its applica -tion to testing core-based design,"Proc. of Test Conference, 1998. pp.458-464.

[15] Chandra, A., Chakrabarty, K., "System-on-chip test data compression and decompression archi -tectures based on Golomb codes," Computer -Aided Design of Integrated Circuits and Sys -tems, IEEE Transactions on, Vol. 20 Issue. 3 , March 2001 pp.355-368.

[16] Chandra, A., Chakrabarty, K., "Frequency -directed run- length(FDR) codes with applica -tion to system-on-chip test data compression," Proc. of VLSI Test Symposium, 2001 pp.42-47.

[17] Girard, P., "Survey of Low-power testing of VLSI Circuits," Design & Test of Computers, IEEE Transactions on, Vol. 19 Issue. 3, 2002, pp.80-90.

[18] Klaus Holtz and Eric Holtz, "Lossless Data Com

-pression Techniques," Proc. of Idea/ Microelec -tronics, 1994, pp.392-397.
[19] B. Pouya and A. Crouch, "Optimization Trade -offs for Vector Volume and Test Power," Proc. of Int'l Test Conf.(ITC 00), IEEE Press, Piscata -way, N.J., 2000, pp.873-881
[20] Wolff F.G., Papachristou C., "Multiscan-based test compression and hardware decompression using LZ77," Proc. of Int. Test Conference, 2002. pp.331-339
[21] Gonciari, P.T., Al-Hashimi, B.M., Nicolici, N., "Improving compression ratio, area overhead, and test application time for system-on-a-chip test data compression/decompression," Proc. of Desi -gn, Automation and Test in Europe Conference and Exhibition, 2002. pp. 604-611
[22] Chandra, A., Chakrabarty, K, "Low-power scan testing and test data compression for system -on-a-chip," Computer-Aided Design of Integra -ted Circuits and Systems, IEEE Transactions on , Volume: 21 , Issue: 5 , May 2002 pp.597 - 604

저 자 소 개

신 용 승(정회원)
2002년 연세대학교 전기공학과 졸업.
2004년 연세대학교 전기전자 공학과 졸업(석사)
현재 LG전자 SystemIC사업담당
<주관심분야: DFT, CAD, Test>

강 성 호(정회원)
1986년 서울대 공대 제어계측공학과 졸업.
1988년 The University of Texas at Austin 전기 및 컴퓨터 공학과 졸업(석사).
1992년 The University of Texas at Austin 전기 및 컴퓨터공학과 졸업(공학박사).
미국 Schlumberger 연구원.
미국 Motorola 선임 연구원.
현재 연세대학교 공과대학 전기전자공학과 교수
<주관심분야: SoC 설계 및 SoC 테스트>