2002년 한국정보처리학회 춘계학술발표논문집 제9권 제1호

저전력 BIST를 위한 테스트 스케줄링

배재성*, 손윤식*, 정정화**
*한양대학교 전자공학과
**한양대학교 정보통신공학과
e-mail : jsbae@shira.hanyang.ac.kr

Test Scheduling for Low Power BIST

Jae-Sung Bae*, Yoon-Sik Son*, Jong-Wha Chong**
*Dept. of Electronic Engineering, Han-Yang University
**Dept. of Information & Communication, Han-Yang University

요 약

BIST(Built-In Self-Test)를 이용한 테스트 방식은 정상 동작 모드인 회로에 비해 테스트 모드에서 보다 많은 스위칭이 발생하고, 과도한 전력 소모에 의해 회로가 손상을 받을 수 있는 문제점을 갖고 있다. 본 논문은 test-per-clock BIST 구조에서 전력이 제한되어 있을 때 테스트 적용 시간과 총 에너지 소비를 최소화하기 위한 테스트 스케줄링 알고리즘을 제안한다. 제안된 방법은 테스트 세션을 구성함에 있어 각 세션에 포함되는 각 블록의 테스트 시작 시간을 동적으로 결정하여 기존의 알고리즘에 비하여 전력 소모와 전체 테스트 시간을 줄일 수 있다.

1. 서론

최근 몇 년간 에너지와 전력 소비에 대한 고려는 회로 설계에 있어 중요한 문제로 대두되었다. 휴대용 장비의 경우 한 번 충전으로 동작할 수 있는 시간이 제품의 성패에 가장 중요한 요소가 되고 있으며, 과도한 전력 소모로 발생하는 열로부터 회로를 보호하기 위한 부가적인 시스템 비용을 낮추기 위해서는 회로의 전력 소모를 낮추는 것이 필수이기 때문이다.

회로의 집적도가 증가하고 동작속도가 빨라짐에 따라 외부 테스트 장비에 의한 회로의 검증은 한계에 이르고 있다. 이러한 한계를 극복하기 위한 방법으로 다양한 테스트 기법이 연구되었으며 가장 널리 이용되는 기법이 BIST(Built-In Self-Test)이다. BIST 는 회로 내에 테스트 패턴을 생성하고 출력을 검사하는 전용 회로를 내장하고 있다. 따라서 BIST 는 외부 테스트 장비에 의한 테스트 방식에 비하여 실제 동작속도에서 대상 회로를 테스트할 수 있고 전체 테스트 시간을 줄일 수 있다는 장점을 갖고 있다. BIST 는 테스트 패턴을 인가하는 방식에 따라 매 클럭마다 패턴을 인가하는 test-per-clock 구조와 테스트 패턴을 내부 스캔에 차례로 채우고 난 후에 테스트 패턴이 회로에 인가되는 test-per-scan 구조로 나눌 수 이으며, 본 논문에서는 BILBO[1]와 같은 테스트 레지스터를 이용하여 구성할 수 있는 test-per-clock BIST 구조를 고려하고자 한다.

BIST 구조에서 가장 널리 사용되는 테스트 생성회로가 LFSR(Linear Feedback Shift Register)에 의한 난수 발생기이다. 그러나 LFSR 에서 생성하는 테스트 패턴을 정상 동작 모드에서의 입력 패턴에 비해 회로에 보다 많은 스위칭을 발생시키므로 테스트 모드에서는 회로의 전체 전력 소모가 크게 증가하여 고장의 원인이 될 수도 이다. 이러한 문제점을 해결하기 위하여 전체 회로를 작은 부 회로들(sub-circuits)로 나누어 부 회로 별로 테스트를 수행하는 방식이 고려되었다. 따라서 보다 짧은 시간 내에 부 회로를 적절히 나누어 테스트할 수 있도록 하기 위한 스케줄링 알고리즘에 대하여 많은 연구가 진행되고 있다.

최근 제한된 소비 전력 하에서 총 테스트 적용 시간을 최소화 하기 위한 테스트 스케줄링 방법들이 제시되었다. 블록 테스트[2]와 이를 바탕으로 클리크 분할(clique partitioning)에 근거한 스케줄링을 제안되었는데[3], 테스트 대상 부 회로는 테스트 블록이고 각각의 블록들은 서로에게 영향을 미치지 않는다는 가정 하에 테스트 블록 사이에 공유하는 테스트 레지스터와 같은 테스트 자원(test resource)의 충돌, 즉 하나의 테스트 자원을 둘 이상의 테스트 블록이 동시에 사용하지 않도록 테스트 순서를 부여한다. 이 가정은 테스트 스케줄링을 간단하게 하지만 전체 전력 소모 면에서는 낭비를 가져올 수 있다.

그림 1 과 같은 두 개의 부 회로 C_1 과 C_2 를 순차적으로 테스트하면 C_1 과 C_2 의 겹쳐지는 부분에서 스위칭이 두 번 일어난다. 그러나 C_1 과 C_2 를 동시에 테스트하면 결과적으로 스위칭이 한번만 발생하여 그만큼의 에너지를 절약할 수 있다. 이와 같이 블록간의 겹쳐지는 영역에서 여러 번 스위칭이 일어나는 것을 고려하여 테스트 상태에서 소비되는 에너지를 최소화하도록 스케줄링 하는 방법이 제안되었다[4].

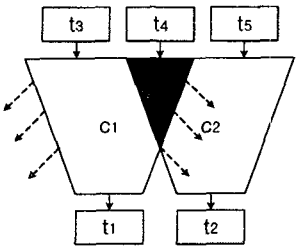

그림 1. 응답 압축기(t_1, t_2)와 패턴 생성기(t_3, t_4, t_5) 로 이루어진 겹쳐진 영역이 있는 부 회로(C_1, C_2)

허용되는 최대 전력 소모와 테스트 자원 충돌을 피하면서 테스트 블록들이 동시에 테스트되는 시간적 구간을 테스트 세션(test session)이라고 한다. 충분한 고장 검출률을 얻기 위해 인가해야 하는 테스트 패턴 수, 즉 테스트 적용 시간은 테스트 블록 마다 다르다. [3]과 [4]의 스케줄링 방법은 테스트 제어의 복잡도를 줄이기 위해서 테스트 세션 내의 모든 테스트 블록에 가장 긴 테스트 패턴을 갖는 테스트 블록과 동일한 테스트 패턴이 인가되는 것을 방치한다. 따라서 패턴수가 적은 테스트 블록에서는 불필요한 스위칭이 발생하여 필요 이상의 에너지가 소모된다.

Clock gating 등을 이용하여 테스트 블록의 테스트가 끝났을 때 무의미한 테스트 패턴이 인가되지 않도록 하여 불필요한 전력 소모를 줄이는 스케줄링 방법이 제안되었다[5]. 이 방법에 의하면 그림 2 와 같은 회로에서 테스트 적용 시간이 $T(C_i)$인 세 개의 부 회로는 동시에 테스트될 수 있음에도 전력 제한에 의하여 C_1 과 C_2 가 동일한 세션에서 테스트 되고 C_3 는 새로운 세션에서 테스트 된다. 따라서 $T(C_1) < T(C_2)$일 때 총 테스트 적용 시간은 $T(C_1)+T(C_2)$가 되고, 에너지 절약 효과는 A 에서만 얻을 수 있다. 즉 C_3 는 새로운 테스트 세션에서 테스트되기 때문에 B 에서 절약할 수 있는 에너지는 낭비된다. C_1 의 테스트가 끝나는 시점에서는 C_1 에 의한 전력 소모가 일어나지 않기 때문에[5] 테스트가 끝나지 않은 블록 전체의 전력과 제한 전력의 차는 그만큼 커져 C_3 에서 소모되는 전력이 이 차이보다 작을 경우 C_3 를 새로운 세션에서 테스트하지 않고 C_1 의 테스트 완료 직후 바로 테스트할 수 있다. 이렇게 부분적으로 C_1 과 C_3 를 동시에 테스트하게 되면 전체 테스트 시간은 C_2 의 남아 있는 테스트 시간만큼 줄어들고 B 에서도 전력 절약 효과를 기대할 수 있다.

본 논문은 위에서 언급한 개념을 이용한 효과적인 테스트 스케줄링 알고리즘을 제안한다. 이 알고리즘은 테스트 적용 시간과 에너지를 동시에 줄일 수 있다.

2 장에서 test-per-clock BIST 구조에 관련하여 전력과 테스트 시간에 대해 효과적인 테스트 스케줄을 찾는 문제를 정형화한다. 3 장에서는 회로의 구분과 이에 대한 구조적인 전력을 평가한다. 스케줄링 알고리즘을 4 장에서 정리하고 마지막으로 제안한 개념을 5 장에서 요약할 것이다.

2. Problem Formulation

Test-per-clock BIST 구조에서 테스트 대상 회로는 테스트 레지스터에 의해서 묶여지는 부 회로, 즉 테스트 유닛(test unit)들로 구분된다. 그리 4 와 같이 테스트 유닛(U3)은 테스트 패턴을 만들어 내는 테스트 레지스터(RC, RD)와 테스트 대상 조합회로(b3), 그리고 테스트 응답을 압축하는 테스트 레지스터(Multiple Input Signature Register, MISR)(RH)로 구성된다.

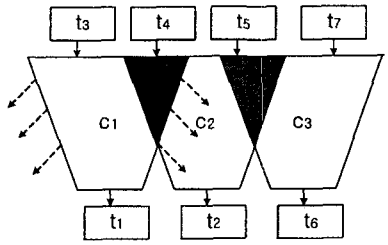

그림 2. 응답 압축기(t_1, t_2, t_6)와 패턴 생성기(t_3, t_4, t_5, t_7)로 이루어진 겹쳐진 영역이 있는 부 회로(C_1, C_2, C_3)

일반적으로 BILBO[1]와 같은 테스트 레지스터는 테스트 패턴 생성과 응답 압축을 동시에 할 수 없다. 서로 다른 두 개의 테스트 유닛이 하나의 테스트 레지스터를 각각 패턴 생성기와 MISR 로 사용한다면 이들 두 테스트 유닛은 동시에 테스트될 수 없기 때문에 순차적으로 테스트되어야 한다. 예를 들어 그림 4 에서 RH 가 U3 에서는 MISR 로 U6 에서는 테스트 패턴 생성기로 사용되기 때문에 U3 와 U6 는 동시에 테스트 될 수 없다. 여러 개의 테스트 유닛들 사이에 테스트 레지스터 공유 관계는 테스트 양립 그래프(Test Compatibility Graph, TCG)로 표현할 수 있다. TCG 를 $G_c = (U, E)$라고 하면 노드 집합 U 는 테스트 유닛을, 에지 집합 E 는 동시에 테스트 가능한 테스트 유닛의 관계를 나타낸다.

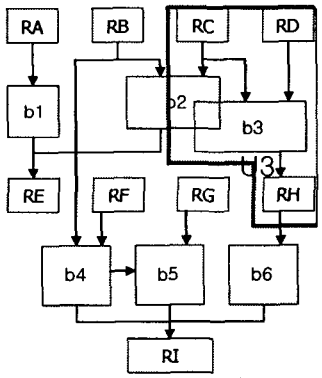

그림 4. 논리 블록(b1, ... ,b6)으로 구성된 회로, 테스트 레지스터(RA , ... ,RI), 간단한 테스트 유닛(U3)

테스트 스케줄은 테스트 유닛의 테스트되는 순서, 즉 테스트 세션의 순서를 결정하는 것이다. 각 테스트 세션에서는 동시에 테스트 가능한 테스트 유닛들을 테스트한다. 테스트 제어를 간단히 하기 위해서 테스트 세션의 모든 테스트 유닛에 같은 길이의 테스트 패턴을 인가한 스케줄링 방법[3][4]을 개선하여 테스트 세션을 서브 세션으로 나누고 충분한 고장 검출률을 얻은 테스트 유닛의 경우 테스트 세션이 끝날 때까지 테스트 패턴을 인가하지 않음으로써 불필요한 에너지를 절약하는 방법[5]은 단순히 각 테스트 유닛마다 테스트가 완료되었을 때 그 테스트 유닛에 해당하는 테스트 패턴 생성 레지스터와 응답 압축 레지스터에 입력되는 클럭을 막음으로써 에너지 소모를 줄였다. 그러나 그림 5 와 같이 $S_{j,1}$ 에서 테스트 유닛 U_3 의 테스트가 완료되었을 때 $S_{j,2}$ 전체의 전력 소모는 U_3 에서 소비되던 전력만큼 감소되어 전력한계까지 여유가 생겨 TCG 에 의하여 현재 진행중인 테스트 유닛들과 동시에 테스트 가능하며 전력한계를 넘지 않는 범위 내에서 아직 스케줄링 되지 않은 새로운 테스트 유닛을 $S_{j,2}$ 에 추가시킬 수 있게 된다.

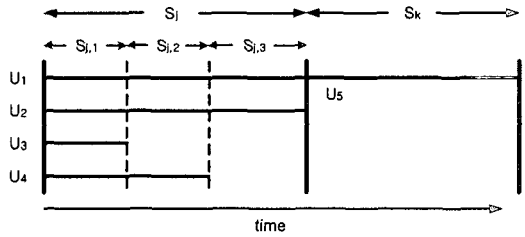

그림 5. 세 개의 부 세션($S_{j,1}$, $S_{j,2}$, $S_{j,3}$)으로 나누어진 테스트 세션 S_j와 단일 테스트 세션 S_k

미리 정해진 전력 한계와 테스트 길이를 고려하여 테스트 세션을 구성하고 동시 테스트 가능한 유닛이 있더라도 전력 한계 때문에 새로운 테스트 세션을 구성하는 이전의 테스트 스케줄링 방법과 달리 우리는 새로운 테스트 세션을 시작하기 전에 세션 내 테스트 유닛의 테스트가 완료될 때마다 현재 세션에 추가될 수 있는 테스트 유닛을 탐색하고 부 세션을 구성한다. 그림 5 와 같이 테스트 유닛 U_3 의 테스트가 끝난 시점에서 나머지 테스트 유닛과 U_5 로 부 세션을 구성하여 세션 S_k 만큼의 테스트 시간을 줄이고 부 세션 $S_{j,2}$ 에 있는 테스트 유닛들과 새로 추가된 U_5 사이에 겹쳐지는 회로 부분이 동시에 테스트 되어 에너지 또한 절약될 것이다.

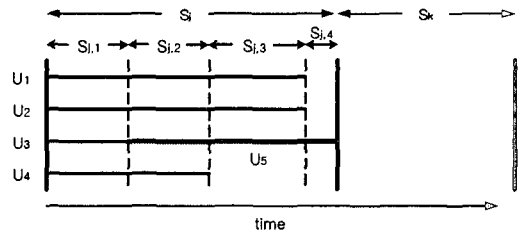

그림 6. 네 개의 부 세션($S_{j,1}$, $S_{j,2}$, $S_{j,3}$, $S_{j,4}$)으로 나누어진 테스트 세션 S_j

이 문제를 정형화하면 다음과 같다. TCG G_c 와 제한 전력이 주어졌을 때 아래의 조건을 만족하도록 G_c 를 m 개의 테스트 세션으로 나누되 하나의 테스트 세션에서 테스트 적용 시간이 짧은 테스트 유닛의 테스트가 완료될 때마다 새로운 테스트 유닛을 추가 시킴으로써 세션 내의 부 세션을 구성한다.

$$P_{peak}(s_j) \le P_{\text{limit}} \quad for\ j = 1,\ldots,m$$

$$\sum_{j=1}^{m} E(s_j) = \sum_{j=1}^{m}\sum_{k=1}^{n_j} P_{avg}(s_{j,k}) \bullet t(s_{j,k}) \ \text{ is minimal}$$

- $E(S_j)$: 테스트 세션 S_j에서의 소모되는 에너지
- C_j: S_j의 부 세션의 수
- $t(S_{j,k})$: $S_{j,k}$에서의 테스트 적용 시간

여기서 우리는 테스트 레지스터에서 발생하는 전력 소모는 고려 대상에서 제외한다.

3. Power Estimation

테스트 유닛들의 소비 전력과 이들 유닛 사이의 겹쳐진 부분이 있어 이를 동시에 테스트할 때 발생하는 전력 절약 효과를 평가하기 위해서 회로를 테스트 유닛보다 더 세부적인 활성영역으로 나누어 이 활성영역에 대해 전력을 계산한다. 회로의 활성영역 $A = \{a_1, \ldots, a_n\}$ 은 다음의 특성을 만족하는 최대 부 회로로 정의한다[4].

첫째, $u_i \in U$ 인 각 테스트 유닛과 $a_r \in A$ 인 각 활성영역에 대해서 a_r 는 u_i 에 완전하게 속해 있거나 u_i 에 완전하게 포함되지 않는다. 즉 활성영역은 서로 다른 테스트 유닛 사이에 겹쳐지지 않는다. 둘째, u_i 가 테스트될 때 a_r 의 모든 부분에 스위칭이 일어나거나 전혀 일어나지 않는다. 마지막으로 모든 테스트 레지스터는 레지스터 자체로 활성영역이다.

이상의 정의에 의해서 각각의 활성영역은 패턴 생성 테스트 레지스터에 의해서만 스위칭이 발생하며 다른 테스트 유닛 사이의 교차에 의해서는 스위칭이 발생하지 않는다. 테스트 유닛 u_i 가 테스트될 때 활성영역 a_r 에서의 평균 소비 전력을 $P_{avg}(a_k)\big|_{u_i}$ 라 하면 테스트 유닛 u_i 가 테스트될 때 소비되는 평균전력은 모든 활성영역의 합과 같다.

$$P_{avg}(u_i) = \sum_r P_{avg}(u_i)\big|_{u_i}$$

부 세션 $s_{j,k}$ 에서의 최대 전력 $P_{peak}(s_{j,k})$ 도 마찬가지로 다음과 같이 계산할 수 있다.

$$P_{peak}(s_{j,k}) = \sum_r \max_{u_i \in s_{j,k}} \{P_{avg}(u_i)\big|_{u_i}\}$$

4. Test Scheduling

각 단계에서 단일 테스트 유닛을 현재 세션에 더하여 첫 번째 세션을 구성한다. 그 다음 현재 세션에서 가장 짧은 테스트 길이를 갖는 테스트 유닛이 끝날 때 마다 세션에서 제하고 아직 스케줄 되지 않은 테스트 중 현재 세션에 남아 있는 테스트 유닛들과 compatible 하면서 제한된 전력을 넘지 않는 테스트 유닛을 더해가는 과정을 반복한다.
일반적으로 절약되는 에너지를 계산하는 식은 다음과 같다.

$$E_\Delta(u_i, s_{j,k}) = E(s_{j,k}) + E(\{u_i\}) - E(\{u_i\} \cup s_{j,k})$$

테스트 세션 $s_{j,k}$ 에 테스트 유닛 u_i 를 더하는데 대한 gain 을 다음과 같이 정의한다.

$$g_1(u_i, s_{j,k}) = \begin{cases} E_\Delta(u_i, s_{j,k}) \\ \quad \text{if } P_{peak}(\{u_i\} \cup s_{j,k}) \le P_{limit} \text{ and} \\ \quad u_i \text{ is compatible to all test units in } s_{j,k} \\ -\infty \quad \text{otherwise} \end{cases}$$

부가적으로 테스트 세션 중 하나의 테스트가 끝난 후 다음 테스트 유닛을 추가함으로써 절약되는 에너지는 다음의 식에 의해서 계산된다.

$$E_\Delta(u_i, s_{j,k}) = E(s_{j,k-1}) - E(\{u_{short}\}) + E(\{u_{short}\} \cap s_{j,k-1}) + E(\{u_i\}) - E(\{u_i\} \cup s_{j,k})$$

테스트 유닛 u_i 현재 테스트 세션에 더하는 것에 대한 gain 을 다음과 같이 정의한다.

$$g_2(u_i, s_{j,k}) = \begin{cases} E_\Delta(u_i, s_{j,k}) \\ \quad \text{if } P_{peak}(\{u_i\} \cup s_{j,k}) \le P_{limit} \text{ and} \\ \quad u_i \text{ is compatible to all test units in } s_{j,k} \\ \quad \text{where, } l = 1,2,3,\ldots,k-1 \\ -\infty \quad \text{otherwise} \end{cases}$$

그림 6의 알고리즘은 테스트 스케줄링 절차를 보였다.

```
/* Input: Set of test units, U */
/* Output: Test schedule (s_1, s_2, ... , s_m) */
u ← test unit of U with largest test length
m ← 1          /* number of current test session */
n ← 1          /* number of current test sub-session */
s_{m,n} ← {u}      /* current test session */
M ← U \ {u}  /* test unit not yet scheduled */
while ( M ≠ ∅ )
   for each u ∈ M
      determine gain for u added to s_m
   end for
   let u_best be the test unit with the largest gain
   if (g_1(u_best, s_{m,n}) ≥ 0)
      s_{m,n} ← s_{m,n} ∪ {u_best}
      M ← M \ {u_best}
   else
      let u_short be the test unit with the shortest length in s_{m,n}
      s_{m,n←n+1} ← s_{m,n} \ {u_short}
      while ( s_{m,n} ≠ ∅ )
         for each u ∈ M
            determine gain for u added to s_{m,n}
         end for
         if (g_2(u_best, s_{m,n}) ≥ 0)
            s_{m,n} ← s_{m,n} ∪ {u_best}
            M ← M \ {u_best}
         else
            u_short ← test unit of s_{m,n} with shortest test length
            s_{m,n←n+1} ← s_{m,n} \ {u_short}
         end if
      end while
      u ← test unit of M with largest test length
      m ← m+1
      s_{m,n} ← {u}
      M ← M \ {u}
   end if
end while
```

그림 7. 제안하는 알고리즘

5. 실험 및 결과

제안하는 알고리즘의 성능을 평가하기 위하여 ISCAS'89 벤치마크 회로를 이용한다. BIST 를 구현하기 위하여 회로 구조의 사이클(cycle)을 제거하는 플립플랍의 수를 하드웨어 오버헤드가 최대한 작도록 결정하고[7] 결정된 플립플랍을 테스트 레지스터로 합쳐서 BIST 를 구현한다.

회로	PIs	POs	FFs	Gate	Test unit	Acivity Regions
S298	3	6	14	119	15	26
S838	34	1	32	446	33	49

표 1. Benchmark 특성

LFSR(Linear Feedback Shift Register)을 구현하여 Pseudorandom 테스트 패턴에 의한 각각의 테스트 유닛에 대한 고장 시뮬레이션을 하여 고장 검출률이 최대가 되도록 패턴의 수를 구하였다.

고장 시뮬레이션은 Fsim 을 이용하였고 전력 평가는 SIS-POSE 를 이용하였다.

회로	Seq	[5]	Proposed
S298	100%	61.5%	53%
S838	100%	64%	59%

표 2. [5]와 제안된 방법의 테스트 길이

회로	Seq	[5]	Proposed
S298	100%	81%	77%
S838	100%	78.5%	72%

표 3. [5]와 제안된 방법의 에너지 소모

6. 결론

본 논문은 원하는 고장 검출률을 얻기 위해 각 테스트 유닛에 인가되는 테스트 패턴의 수가 다르고 전력이 제한되어 있을 때 테스트 적용 시간과 총 에너지 소비를 최소화하기 위한 테스트 스케줄링 알고리즘을 제안하였다. 기존의 테스트 스케줄링 방법은 제어의 복잡도를 줄이기 위하여 단순히 테스트 세션을 하나씩 구성해 가는 단순한 방법을 취한 반면 우리는 테스트 세션을 구성함에 있어 각 세션에 포함되는 각 블록의 테스트 시작 시간을 동적으로 결정하여 기존의 알고리즘에 비해 전력 소모와 전체 테스트 시간을 줄일 수 있다.

참고문헌

[1] B. Koenemann, J. Mucha, and G. Zwiehoff, "Built-in logic block observation techniques," *Proceedings Test Conference*. 1979.

[2] G. L. Craig, C. R. Kime, and K. K. Saluja, "Test Scheduling and Control for VLSI Built-In Self-Test," in *IEEE Transactions on Computers*. 1988.

[3] R. M. Chou, K. K. Saluja, and V. D. Agrawal, "Scheduling Tests for VLSI Systems Under Power Constraints," *IEEE Transactions on VLSI Systems*. 1997.

[4] T. Schuele and A. P. Stroele, "Test Scheduling For Minimal Energy Consumption Under Power Constraints," *Proceedings VLSI Test Symposium*. 2001.

[5] T. Schuele and A. P. Stroele, "Scheduling Tests For Low Power Built-In Self-Test," *IEEE International Symposium on Circuits and Systems*, 2001.

[6] A. P. Stroele and H. –J. Wunderlich, "Configuring Flip-Flops to BIST Register," *Proceedings International Test Conference*. 1994.

[7] A. P. Stroele and H. –J. Wunderlich, "Hardware-Optimal Test Register Insertion," *Proceedings International Test Conference*. 1998.

[8] O. F. Haberl and H. –J. Wunderlich, "The synthesis of self-test control logic," Proc. COMPEURO, 1989.