

테스트 자원 그룹화를 이용한 시스템 온 칩의 테스트 스케줄링

박진성, 이재민
관동대학교 전자공학과
전화 : 033-670-3392

Test Scheduling for System-on-Chips using Test Resources Grouping

Jin Sung Park, Jae Min Lee
Dept. of Electronic Engineering, Kwandong University
E-mail : leejm@kwandong.ac.kr

Abstract

Test scheduling of SoC becomes more important because it is one of the prime methods to minimize the testing time under limited power consumption of SoCs. In this paper, a heuristic algorithm, in which test resources are selected for groups and arranged based on the size of product of power dissipation and test time together with total power consumption in core-based SoCs is proposed. We select test resource groups which has maximum power consumption but does not exceed the constrained power consumption and make the testing time slot of resources in the test resource group to be aligned at the initial position to minimize the idle test time of test resources.

I. 서론

반도체 공정 기술이 발전하고 시스템 복잡도가 증가함에 따라 System-on-Chips (SoCs) 기술의 필요성이 대두되어 이에 대한 활발한 연구가 진행되어 오고 있다. SoC 시스템은 서로 다른 코어 공급자가 제공하는 IP 코어들이 하나의 칩에 기능함수의 블록으로 구성되는 것이므로 개별 코어의 테스트 보다 다양하고 복잡한

테스트 기법을 요구하기 때문에 테스트가 어렵고 많은 수행 시간을 필요로 한다. 테스트 수행시간의 감소를 위한 방법 중에서 테스트 스케줄링은 SoC 설계 자체에는 하드웨어적으로는 부담을 주지 않으면서 테스트 수행 시간을 감소시켜 테스트 비용을 줄일 수 있는 효과적인 방법이다. SoC 테스트에서는 테스트 수행시간의 최소화와 함께 고장검출율과 소비전력의 최적화 문제가 요구될 수 있다[1].

SoC의 테스트 스케줄링을 위한 여러 가지 방법들이 제안되어 왔다[2-6]. 이들은 크게 병렬처리 능력을 높이는 방법이거나, 코어 공급자가 제공하는 테스트 집합들 중에서 효율적인 테스트를 선택하는 방법 또는 효율적인 자원 분배를 수행하여 테스트 충돌을 제거하는 방법 등으로 구분할 수 있다. 기존에 제시되었던 테스트 스케줄링 알고리즘으로서 스케줄링의 기본 구성과 MILP (Mixed-Integer Linear Programming)를 사용하여 테스트 스케줄링을 생성하는 방법[2]는 초기 SoC를 위한 테스트 스케줄링의 개념 정립 및 병렬처리 수행의 가능성을 제시한 알고리즘이다. 그러나 이 알고리즘은 테스트 모드에서 각 코어의 실질적인 전력 소비량을 고려하지 않았으며, 특히 코어와 자원의 수가 증가하면 할수록 코어들의 테스트 시작시간을 찾는 계산량과 테스트의 위치선정에 필요한 계산량이 지수적으로 증가하는 단점을 가지고 있다. 둘째로 Chakrabarty의 논문[3]에서는 MILP를 응용한 3 가지

(Precedence-Based, Preemptive, and Power-Constrained Scheduling) 테스트 스케줄링 방법을 제시하였는데 이러한 방법들은 코어 공급자들과의 긴밀한 협조가 이루어져야만 좋은 효과를 발휘할 수 있고, 각 테스트의 병렬성을 충분히 활용하지 못한 단점을 갖고 있다. 셋째로 동적 분할 한정 전력 동시 테스트 집합 (Dynamically Partition Power-Constrained Concurrent Test Sets) 알고리즘을 이용한 스케줄링 방법[4]는 전력 소비량이 고려된 나무 성장 그래프를 이용하여 충돌하지 않는 코어들을 찾아 다중 배치가 가능하도록 한 테스트 스케줄링 방법인데 코어 내에서 존재하는 개별적인 자원의 세부적인 충돌을 고려하지 않아서 최악의 경우 다른 제안들보다 총 테스트 시간이 더 커질 수 있다는 문제점을 안고 있다.

본 논문에서는 SoC를 위한 테스트 자원의 그룹 중 전력·시간 곱의 크기가 높은 자원을 우선 순위로 배열하여 병렬 처리 능력을 향상시킨 새로운 테스트 스케줄링을 제안한다. 제안하는 스케줄링 방법에서는 스케줄링의 최소단위를 각 자원별 테스트 시간과 전력사용량으로 하여 코어의 충돌이 발생하더라도 자원의 충돌이 최소화되도록 한다. 테스트 스케줄링을 위한 테스트 자원은 자원의 충돌을 고려한 확장 나무성장 그래프 (ETGG : expanded tree growing graph)를 이용하여 각 자원들의 충돌 여부를 조사하고 충돌이 없는 자원을 집산화한 후 이들 사이의 우선 순위에 따라서 테스트 자원을 선택하고 배치함으로써 테스트 시간의 변화에 따른 테스트 자원의 병렬 처리 능력을 높이고 무위 테스트 시간을 줄임으로 안정화된 총 전력사용량 내에서 테스트 시간을 최소화하는 장점을 가진다.

II. 테스트 스케줄링을 위한 SoC 모델링

일반적인 SoC 모델로서 그림 1과 같이 디지털 코어, 아날로그 코어 및 아날로그와 디지털이 복합된 복합 신호 코어 그리고 사용자의 의도에 따라서 설계되는 UDL 등이 포함된 회로를 고려한다. SoC 내의 코어들은 그들의 기능에 의하여 여러 가지의 다양한 테스트 방법들을 필요로 한다. 또한 일정한 고장검출율에 도달하기 위하여 각 코어들을 테스트할 때 서로 다른 자원들이 사용되는 다중 테스트 집합이 필요하다.

(정의 1) SoC 내 구성되어 있는 IP 코어는 C 로 표시하며 $C = \{C_1, C_2, \dots, C_n\}$ 로 나타내고, 테스트 가능한 전체의 자원은 R 로 표시하며 $R = \{R_1, R_2, \dots, R_m\}$ 로 나타낸다. 여기서 C 는 유한개의 원소를 가지는 집합이고 n 은 코어 모듈의 수를

의미한다. 코어의 수를 N_c 로 나타내고 $N_c = n$ 이다. R 은 유한개의 원소를 가지는 집합이고 m 은 자원의 수를 의미한다.

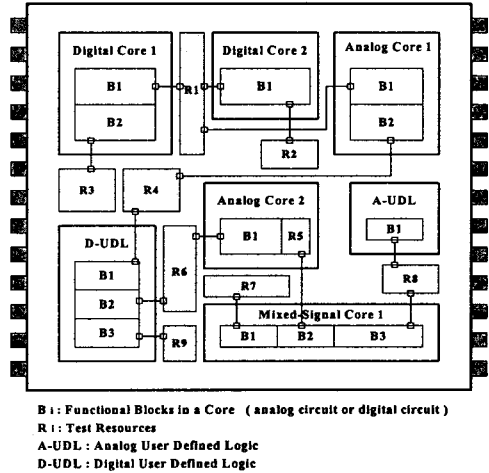


그림 3-1. 일반적인 SoC 모델

1. 테스트 자원 충돌 및 소비 전력의 제한

설계하고자하는 SoC가 n 개의 코어들을 포함하고 테스트가 가능한 m 개의 자원을 가지고 있다고 가정하자. 만약 코어들이 동시에 병렬로 수행된다면 같은 자원을 동시에 사용하는 코어들간에 충돌이 발생한다. 또한 모델 내의 테스트 수행 시 발생하는 전력 소비는 어떠한 시간에서도 반드시 최대 허용 전력을 초과할 수는 없다.

(정의 2) 코어들 사이의 자원 중에서 두 개의 테스트 시간을 비교하여 하나의 테스트 시간 내에 다른 테스트의 시작시간이 위치되고 동일한 자원을 공유하고 있으면 이를 자원의 충돌 (RC)이라 한다.

(정의 3) 각 테스트 자원은 테스트 모드에서 발생하는 전력 소비량 P_m 을 가지며 같은 시간 구역 내에서 $\{P_m, P_{m+1}, \dots, P_{m+k}\}$ 으로 표현된다. 만약 전력 소비량 집합 원소들의 총 합이 SoC의 총 전력 사용량인 P_{max} 를 초과한다면 자원의 충돌 (RC)이라 한다.

정의 1과 2를 다음과 같이 나타낼 수 있다.

$$\text{IF } T_{sm} - T_{sm+a} < 0 \text{ and } T_{sm+a} - T_{fm} < 0 \text{ and } T_{fm} - T_{fm+a} < 0, RC(m, m+a) = 1$$

$$\text{IF } P_{\max} < \sum_{i=0}^k P_{m+i}, RC(m, m+1, \dots, m+k) = 1$$

여기서 s 는 시작시각, f 는 종료 시각이고 a 는 임의의 상수로 a 번째 자원을 의미하며 k 는 같은 시간 구역에서 자원 공유의 충돌이 없는 테스트 자원의 수이다. 그리고 RC 값이 1이면 테스트 자원의 충돌을 의미한다.

2. 테스트 집합 선택 및 자원의 집단화 (grouping)

예제 SoC S 는 5개의 코어와 7개의 테스트 자원으로 구성되어 있으며 각 테스트 자원의 연결 관계는 그림 3과 같다.

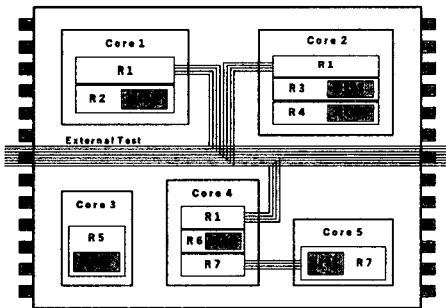


그림 3. 5개 코어 7개 테스트 자원으로 구성된 SoC S 의 연결관계

그림 3과 같이 설계된 SoC S 의 정보인 테스트 시간과 사용전력량 그리고 각 연결 관계를 고려한 정보를 표 1과 같이 나타낸다.

표 1. SoC S 의 테스트 자원 행렬 표현

코어	C1	C2	C3	C4	C5	코어
R1	T_{11}, P_{11} (200, 60)	T_{21}, P_{21} (70, 30)		T_{41}, P_{41} (120, 80)		3
R2	T_{12}, P_{12} (100, 35)					1
R3		T_{23}, P_{23} (25, 100)				1
R4		T_{24}, P_{24} (43, 50)				1
R5			T_{35}, P_{35} (43, 50)			1
R6				T_{46}, P_{46} (50, 100)		1
R7				T_{47}, P_{47} (75, 80)	T_{57}, P_{57} (130, 120)	2
자원수	2	3	1	3	1	

(정의 4) 테스트 자원의 연결관계 집합에서 충돌하는 자원들을 제외하여 그 나머지 자원들로 테스트를 집합을 구성하는 것을 테스트 자원의 집단화 (grouping)라 한다.

3. 확장 나무 성장 그래프 (Expanded Tree Growing Graph)

자원의 충돌 여부와 병렬 처리가 가능한 자원들을 찾아내기 위하여 나무 성장 그래프 (TGG : tree growing graph)를 이용한다. 그림 4는 표 1의 내용을 TGG로 나타낸 것이다. 최상위 레벨은 코어, 그리고 두 번째 레벨은 각 코어 내에 공유되어 있는 테스트 자원, 그리고 그 다음 레벨부터는 각 자원이 공유하는 코어를 확인하기 위한 연결로 구성된다. 각 레벨을 연결하는 연결선은 자원의 공유를 의미하기 때문에 하나의 코어에 동시에 연결되어 있는 자원은 병렬 처리를 수행할 수 없고, 같은 레벨에 속해 있는 자원들 또한 병렬 처리가 불가능하다.

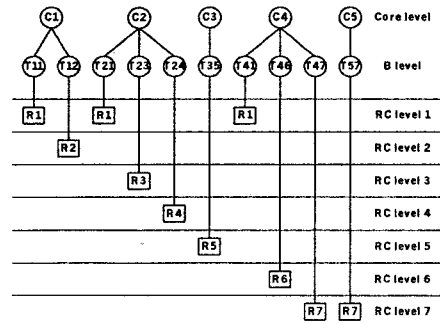


그림 4. 예제 SoC S의 나무 성장 그래프

자원의 공유 관계와 전력사용량을 함께 고려한 나무 성장 그래프를 그림 5에 나타내었고 이것을 확장 나무 성장 그래프 (Expanded-TGG : ETGG)라 한다. 이것은 각 자원들의 테스트 시간과 전력사용량을 나타내며 동시 사용 가능한 자원들의 관계를 연결선으로 표시한 것이다.

III. 제안하는 테스트 스케줄링

1. 전력사용량을 고려한 우선 순위 테스트 집합 생성

시스템 S는 n 개의 코어 C_n 으로 구성되며 단일 코어는 다중 테스트 집합을 포함할 수 있다. 그리고

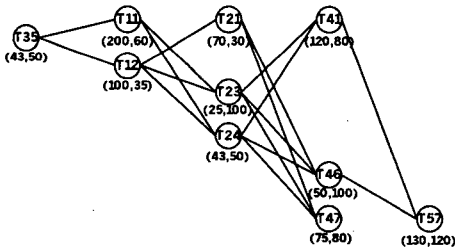


그림 5. 예제 SoC S의 확장 나무 성장 그래프

m 개의 자원 R_m 은 동일한 자원을 공유할 수 있으며 어떠한 시간대에도 자원의 전력 소비량의 합은 P_{max} 를 넘지 않는다고 가정한다.

확장 나무 성장 그래프로 생성된 자원의 충돌이 없는 테스트의 그룹을 G_p ($1 \leq G_p \leq q$)라 표한다. G_p 는 한정된 집합이고 q 는 집합의 수를 의미한다. RC 값이 0인 집합내의 원소들은 각각 테스트 시간과 전력 소비량을 포함하며 전체의 전력소비량의 합을 OPG (Overall Power dissipation Group) 그리고 테스트 시간 중 가장 긴 테스트 시간을 LTG (Logiest Test time Group)라 한다. 우선 순위를 그룹별로 비교하기 위하여 테스트 자원 및 그룹의 전력·시간 곱의 크기 (SUR)를 연산한다.

SoC S에서 주어지는 자원들을 조사하고 전력 사용량이 고려된 확장 나무성장 그래프로부터 각 자원의 요소들에 대한 테스트자원의 행렬표현을 얻은 후 비 충돌 테스트 집합인 G_p 를 생성한다. 다음으로 각 자원의 SUR 및 그룹의 OPG를 계산한다. 만약에 조사

된 G_p 에서 $OPG_p > P_{max}$ 이면 우선 순위에서 제외되고 그렇지 않으면 최고 우선 순위로 선택된다. 반면에 조사된 G_p 들 중에서 $OPG_p < P_{max}$ 인 그룹이 2개 이상일 경우는 그들 중에서 P_{max} 에 가장 근접하는 값으로 선택된다. 모든 그룹의 OPG가 P_{max} 를 초과할 경우는 각 그룹 내에서 가장 작은 P_{nm} 값을 하나씩 제거해 나가면서 P_{max} 값에 가장 근접하는 우선 순위 그룹 PrG를 선택한다. 만약 동일한 OPG값을 가지는 그룹이 하나 이상일 경우는 그것들의 SUG 값을 계산하여서 큰 값을 갖는 OPG를 PrG로 선택한다.

2. 휴리스틱 테스트 스케줄링 알고리즘

테스트의 자원을 배치할 경우 최우선그룹을 PrG라 하고 $PrG = \{g_1, g_2, \dots, g_\gamma\}$ 로 표현한다. PrG는 유한 집합이고, γ 는 테스트 자원의 수이다. 또한 g 는 T_{nm} 과 P_{nm} 값을 가지는 단일화된 변수를 의미한다. 이어서 집합 내 g 의 값들 중 테스트 시간의 값을 서로 비교해서 가장 긴 테스트 순으로 Y축의 영점에 가깝도록 그림 6(a)와 같이 테스트를 배치한다.

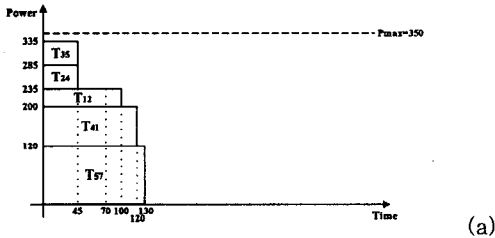
(정의 5) 테스트 시간 구역은 테스트의 시작 시간인 x 좌표의 영점 (zero)부터 배치된 자원의 종료시까지를 의미하며 B로 표시하고 $B = \{B_0, B_1, \dots, B_{a-1}\}$ 로 나타낸다.

자원의 효율적인 배치를 위하여 각 시간구역별 전력·시간 곱의 크기를 조사한다. 정의 5에서 a 는 배치된 자원의 수이고, 원소 중 종료시간이 작은 순으로 배열된다. 일반적으로 B_0 시간대에서는 자원의 전력의 합인 OPG가 P_{max} 와 유사하므로 고려 대상에 들어가지 않고 B_1 시간대부터 여분의 배치 공간 전력 소비 가능량 (MS)을 조사하고 동시에 나머지 선택되지 않은 자원들 중에서 B_1 구역에서의 병렬 처리 가능성을 조사한다. 만약 일치하는 자원을 찾을 수 없다면 구역을 B_2 로 변화하여 동일한 작업을 수행한다. 그러나 충돌이 없는 테스트 자원을 찾을 수 있다면, 자원을 선택하고 임의의 B_a 구역에서 $MSB_a + SUGB_a < P_{max}$ 이면 자원을 선택한다. 만약 선택되는 자원이 2개 이상일 경우는 각 자원의 SUR을 비교하여 큰 값을 우선 배치한다. 이때 시작 시간은 B_a 이고 종료 시간은 B_a 와 T_{nm} 의 합이다. 새롭게 추가되는 종료 시간을 B 집합에 추가한다. 이러한 일련의 과정을 그림 6(b)와

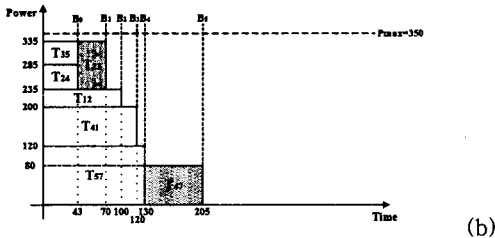
같이 모든 테스트의 집합이 선택되어서 배치될 때까지 반복적으로 수행한다. 표 1에서 나타난 시스템 S는 그림 6(c)와 같은 테스트 스케줄링 결과를 얻는다. 여기서 테스트 총 수행시간인 OTT는 B집합에서 가장 큰 값이다. 위와 같은 테스트 스케줄링 알고리즘을 그림 7에 나타내었다.

IV. 실험 결과

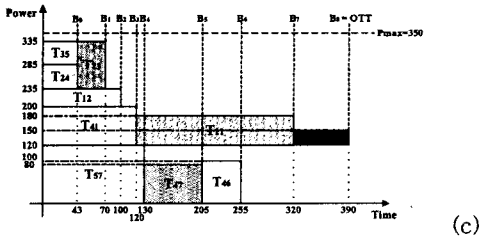
제한한 휴리스틱 테스트 스케줄링 알고리즘을 구현하여 시뮬레이션하고 기존에 제시된 스케줄링 알고리즘과 비교 검토한다. 시뮬레이션을 위해서 ITC'02 (IEEE International Test Conference)에서 지정한 SoC Test Benchmarks 자료[7, 8]를 사용한다. 제한한 알고리즘과 효율성을 비교하기 위한 기존의 스케줄링 방법으로 전력 소비량이 고려된 DP-PCTS 알고리즘



(a)



(b)



(c)

그림 6. 예제 SoC S의 스케줄링 과정

Procedure Test Scheduling()

- ```

(
1. ETGG를 이용하여 생성된 PrG 를 선택 ;
2. Sort (PrG 내부 원소를 테스트가 긴 순서대로);
3. For 미 배치 테스트 자원에 대하여
4. IF PrG 원소와 동일 레벨의 자원이 존재, then
 동일 레벨 자원 선택 연속적으로 배치;
 시작과 종료시간 Update;
Else
 RC , Pnm 조사;
5. For 모든 자원에 대하여
 각 구역별 종료시간 B 와 배치공간 SUGBo 조사;
6. IF RC=0 and MSBo+SUGBo<Pmax , then
 테스트를 배치하고 시작과 종료시간 Update;
7. IF 자원의 수 > 2, then
 자원의 SUR 을 비교 큰 값을 우선 배치;
 시작과 종료시간 Update;
8. B 집합의 최대값을 OTT 에 대입
)

```

그림 7. 휴리스틱 테스트 스케줄링 알고리즘 [4]를 사용한다.

알고리즘의 동작을 설명하기 위하여 예제로 사용한 SoC S를 대상으로 DP-PCTS방법을 적용하여 스케줄링한 결과 총 테스트 시간은 그림 8과 같이 683 UT(Units of Time)을 얻을 수 있었다. 제한한 휴리스틱 테스트 스케줄링 알고리즘으로 시뮬레이션 한 결과를 그림 6(c)에서 보았다. 우선 순위 병렬처리 가능 그룹을 배치한 후에 충돌하는 테스트 자원을 우선하여 배치함으로써 테스트 자원들 사이의 무위 테스트 시간의 발생을 최소화하여 총 테스트 시간 390 UT를 얻었다. 두 가지의 알고리즘으로 테스트 스케줄링한 결과 제안된 알고리즘이 293 UT (42.9%)감소됨을 확인할 수 있다.

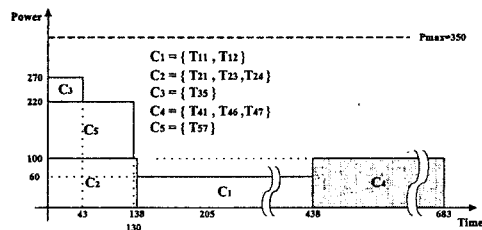


그림 8. 예제 SoC S의 DP-PCTS 알고리즘 시뮬레이션 결과

제안된 알고리즘들을 ITC'02 SoC Test Benchmarks 회로 중 전력사용량이 주어진 h593.soc 회로[7]를 사용하여 실험하였다. h593 회로는 총 9개의 모듈로 구성되어 있으며 그중 모듈 0은 SoC의 외형 입출력만을 나타내는 것이기 때문에 테스트가 수행되지 않는다. 그러므로 시스템 내부의 IP코어는 총 8개이다. 여기서 각 모듈들의 연결 관계와 총 전력 소비량을 다음과 같이 정한다. 코어 2, 5와 7은 BIST를 단독으로 사용한다. 또한 코어 1, 4와 8은 공통 BIST 엔진을 공유하며 코어 4는 이것 이외에 또 다른 BIST 엔진을 하나 더 포함하고 있다. 나머지 코어 2, 5, 6 과 8은 개별된 BIST 엔진을 갖는다. 총 전력소비량은 7300 UP(Unit of Power)로 한다. IP코어 1, 3, 4, 6과 8은 공통 외부 테스트를 공유한다.

벤치마크 정보를 기초로 제안한 알고리즘들을 수행한 결과 그림 9와 같은 ETGG를 얻고, 표 2와 같은 테스트 자원의 시작 시각과 종료 시각을 얻는다. 이 결과를 DP-PCTS 알고리즘과 비교하면 총 테스트 시간이 10292에서 6592로 35.95% 감소됨으로 제안한 알고리즘의 종래 방식에 비해 보다 효율적임을 알 수 있다.

### V. 결론

본 논문에서는 SoC 내 테스트자원의 충돌을 고려하고 확장 나무 성장 그래프를 이용하여 테스트 자원들의 전력·시간 곱의 비가 높은 테스트 자원 그룹을 우선하여 배열하는 자원의 선택 및 집단화를 통해 병렬 처리 능력을 높이고 무위 테스트 시간을 최소화하고 최적화된 테스트 스케줄링 결과를 얻을 수 있게 하였다. 제안한 휴리스틱 스케줄링 알고리즘들은 최악의 경우 복잡도가  $O(n^2)$ 로 다른 알고리즘의 복잡도와 같거나 낮으면서도 ITC에서 지정한 벤치마크 회로를 대상으로 구현한 알고리즘을 실험한 결과 다른 알고리즘

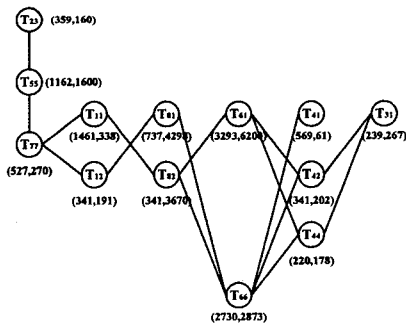


그림 9. 벤치마크 회로 h593의 확장나무

### 성장 그래프

표 2. 벤치마크 회로 h593을 시뮬레이션 결과

| 테스트 자원           | 제안된 알고리즘 |      | DP-PCTS 알고리즘 |       |
|------------------|----------|------|--------------|-------|
|                  | 시작시각     | 종료시각 | 시작시각         | 종료시각  |
| $T_{11}, P_{11}$ | 0        | 1461 | 1098         | 2559  |
| $T_{12}, P_{12}$ | 1461     | 1802 | 2559         | 2900  |
| $T_{23}, P_{23}$ | 0        | 359  | 0            | 359   |
| $T_{31}, P_{31}$ | 1461     | 1700 | 2900         | 3139  |
| $T_{41}, P_{41}$ | 6023     | 6592 | 3139         | 3708  |
| $T_{42}, P_{42}$ | 341      | 700  | 3708         | 4049  |
| $T_{44}, P_{44}$ | 0        | 220  | 4049         | 4269  |
| $T_{55}, P_{55}$ | 341      | 1503 | 0            | 1162  |
| $T_{61}, P_{61}$ | 2730     | 6023 | 4269         | 7562  |
| $T_{66}, P_{66}$ | 0        | 2730 | 7562         | 10292 |
| $T_{77}, P_{77}$ | 341      | 883  | 0            | 527   |
| $T_{81}, P_{81}$ | 1802     | 2539 | 0            | 737   |
| $T_{82}, P_{82}$ | 0        | 341  | 737          | 1098  |
| 테스트 수행 시간        | 0        | 6592 | 0            | 10292 |

에 의해 제공되는 총 테스트 시간보다 30% 이상의 감소된 총 테스트 시간을 얻어 알고리즘들의 효율성을 보였다.

향후 연구 과제로는 제안된 알고리즘들을 효과적으로 검증할 수 있는 벤치마크 회로의 개발과 효과적인 테스트 스케줄링이 생성된 이후에 자동적으로 범용 테스트 컨트롤러 IP를 SoC의 코어로 사용 가능하도록 시스템 레벨의 VHDL 코드로 생성할 수 있는 CAD 툴을 개발하는 것이다.

### 참고문헌

- [1] C. Dislis, J. H. Dick, I. D. Dear and A. P. Ambler (1995). 「Test Economics and Design for Testability」, ELLIS HORWOOD.
- [2] Krishnendu Chakrabarty, "Test Scheduling for Core-based Systems Using Mixed-Integer Linear Programming," *IEEE Trans. On computer-Aided Design of Integrated circuits and Systems*, vol. 19, no. 10, pp. 1163-1174, 2000.
- [3] Vikram Iyenger and K. Chakrabarty, "Precedence-Based, Preemptive, and Power-Constrained Test Scheduling for System-on-a-chip," *19th IEEE Proceedings on VLSI Test Symposium*, pp. 368-374, 2001.

- [4] Zhao D. and S. Upadhyaya, "Dynamically Partitioned Test Scheduling for SoCs Under Power Constraints," *11th IEEE North Atlantic Test Workshop, Montauk, NY, May 2002.*
- [5] Zhao D., S. Upadhyaya and M. Margala, "Minimizing Concurrent Test Time in SoCs by Balancing Resource Usage," *Proceedings of the 12th ACM Great Lakes Symposium on VLSI*, pp. 77-82, 2002.
- [6] Erik Larsson and Zebo peng, "System-on-Chip Test Parallelization Under Power Constraints," *European Test workshop, Stockholm Sweden, 2001.*
- [7] E.J. Marinissen, V. Iyengar and K. Chakrabarty. ITC 2002 SOC test benchmark initiative.  
<http://www.extra.research.philips.com/itc02socbenchm>
- [8] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *Intl. Symp. on Circuits and Systems*, pp. 1929-1934, 1989.