논문 2011-48SD-2-7

AMBA 기반 SoC의 병렬 코어 테스트를 위한 효과적인 테스트 설계 기술

(An Efficient Design Technique for Concurrent Core Testing of AMBA-based SoC)

송 재 훈*, 오 정 섭**, 박 성 주***

(Jaehoon Song, Jungsub Oh, and Sungju Park)

요 약

본 논문에서는 AMBA 기반 SoC의 코어 테스트 시간을 최소화 하는 것을 목표로 한다. 이를 위하여 테스트 대상 코어에 대해 병렬로 테스트를 수행하며 AMBA를 TAM으로 재사용 하는데 있어서 필요한 기술을 제안한다. 기능 테스트시의 AMBA 버스 제어를 위해 설계 된 TIC를 구조적 테스트 시의 제어에 재활용 하여 병렬 테스트의 제어에 필요한 추가 로직을 최소화 하였으며, 기능적 테스트를 수행할 수 있을 뿐만 아니라 구조적 테스트 시 병렬 테스트를 수행 할 수 있어서 SoC의 신뢰성 확보와 테스트 시간 단축에 기여 할 수 있다.

Abstract

The goal of this paper is reducing the test time for AMBA-based SoC. To achieve this goal, the design technique that can test several cores concurrently by reusing AMBA as TAM is proposed. The additional control logic for structural parallel core test is minimized by reusing TIC which is originally used for functional test of AMBA. SoC reliability and test time reduction can be significantly achieved with the concurrent core test technique as well as functional test.

Keywords : SoC, AMBA, IEEE 1500, parallel test, scan test

Ⅰ. 서 론

반도체 공정 기술의 발전으로 시스템의 주요기능을 하나의 칩에 집적한 SoC 제작이 가능하게 되었다. 기능이 검증되어 있으며 재사용이 가능한 IP 코어 기반의 설계 방식을 사용하면서 SoC의 설계 시간은 단축 되었지만, 회로의 복잡도 증가로 인한 테스트 비용 문제가 대두되고 있다. 테스트 비용 문제를 해결하기 위한 하나의 방안으로 테스트에 소모되는 시간을 줄이는 연구가 이루어지고 있다.[1,2,3]

효과적인 SoC 테스트를 위해 내장된 IP 코어 단위로 테스트 하는 모듈러 테스트 방식이 사용될 수 있다. 모듈러 테스트를 위한 필수 구성 요소로는 SoC 외부에서 테스트 패턴의 인가 및 관측을 위한 테스트 패턴 소스/싱크, SoC의 입출력에서 코어의 입출력으로의 테스트 패턴 이동경로인 Test Access Mechanism(TAM), 코어와 TAM간의 인터페이스를 해 주는 Test Wrapper가 있다.[4]

테스트 패턴 소스/싱크로는 주로 Automatic Test

* 정회원, 트란소노
(Transono)
** 학생회원, 한양대학교 컴퓨터공학과
(Department of Computer Science & Engineering, Hanyang University)
*** 평생회원, 한양대학교 전자컴퓨터공학부
(Department of Computer Science & Engineering, Hanyang University)
※ 이 논문은 반도체설계교육센터 (IDEC) CAD 툴 지원을 받아 수행하였음.
※ 이 논문은 2010년도 정부(교육과학기술부)의 재원으로 한국연구재단(NRF)의 중견연구사업 지원을 받아 수행된 것임.(No. 2010-0026822)
접수일자: 2010년11월2일, 수정완료일: 2011년1월20일

Equipment(ATE)가 사용되며, Test Wrapper에는 IEEE 1500[5] 표준 또는 사용자 정의 방식이 사용 될 수 있다. TAM의 경우는 표준화가 되어 있지 않아서 사용자가 정의하여 사용하고 있으며, 크게 테스트를 위한 전용 TAM을 사용하는 방법[6~7]과 기능 버스(functional bus)를 TAM으로 재사용하는 방법이 연구되고 있다.[8~10]

테스트에 소모되는 시간은 테스트를 위한 설계 구조 뿐만 아니라 테스트 수행 방법에도 영향을 받는다. 테스트 수행방법은 크게 테스트 대상 코어를 하나씩 순차적으로 테스트 하는 순차 테스트와 한 번에 여러 개의 코어를 테스트 하는 병렬 테스트로 나눌 수 있다. 순차 테스트의 경우 각 코어의 특성에 상관없이 SoC의 TAM 폭이 각 코어의 TAM폭으로 결정이 된다. 반면에 병렬 테스트의 경우는 병렬 코어 테스트 스케줄링에 의하여 코어 특성(예. 테스트 패턴)에 따라 SoC의 TAM이 여러 개의 Sub-TAM으로 분할되고 각 Sub-TAM에 코어들이 할당 된다. 이 때 코어의 TAM 폭은 코어가 속해있는 Sub-TAM의 폭과 같으며, 각 Sub-TAM 별로 독립적으로 동시에 테스트 될 수 있다.[3]

두 방식 모두 코어의 TAM 폭이 결정된 후 테스트 시간을 최소화하기 위해서는 TAM 폭에 맞추어 각 코어의 스캔체인[11]을 재구성[12]하는 과정이 필요하다. 코어의 테스트 시간은 스캔 체인의 길이에 의하여 결정되므로 이를 최소화 하는 스캔 체인 재구성을 통하여 TAM wire의 길이는 균형 잡힌(balancing) 상태가 된다. 일반적으로 코어에 할당된 TAM 폭이 증가할수록 이에 비례하여 스캔체인의 길이가 짧아져 코어의 테스트 시간이 감소하지만, 항상 그렇지는 않다. TAM 폭이 일정한 수치를 넘는 경우에만 스캔체인의 재구성이 일어나게 되고 이때 테스트 시간이 감소하게 되기 때문이다. 이러한 성질을 고려하지 않고 코어의 TAM 폭을 결정하는 경우 스캔체인의 재구성을 발생 시키지 않는 여분의 TAM 폭은 낭비된다. 병렬 코어 테스트를 위한 스케줄링 시에는 이러한 성질을 이용하여 여분의 TAM 폭을 이용하여 추가로 TAM을 구성할 수 있게 되므로 병렬 테스트 과정의 제어 오버헤드가 크지 않는 한 순차 테스트 방법 보다 더 작은 테스트 시간을 가진다.

본 논문에서는 SoC의 기능 버스로 널리 사용되고 있는 AMBA를[13] SoC의 내장 코어 테스트 시에 TAM으로 재사용 하면서 병렬 테스트가 가능하도록 하여 테스트 시간을 줄이는 방법을 제시한다.

Ⅱ. AMBA 테스트 인터페이스

AMBA 시스템은 그림 1과 같이 Advanced High performance Bus (AHB)와 Advanced Peripheral Bus (APB)로 구성된다. AHB는 고속의 데이터 송수신을 위해 설계 되어 마이크로프로세서와 같은 고성능 모듈간의 연결에 사용되며, APB는 전송속도가 느린 장치들의 인터페이스에 사용된다. AHB와 APB의 연결 시에는 AHB-to-APB 브리지가 사용된다.

Test Interface Controller (TIC)는[13] AMBA 시스템의 기능 테스트 (Functional Test)를 위한 테스트 인터페이스 제어기이다. 칩 외부에서의 테스트 패턴의 인가/관측은 External Bus Interface (EBI)의 TBUS를 통하여 이루어진다. TIC를 이용하여 기능 테스트 수행 시 높은 고장 검출 율을 얻기 위해 사용되는 많은 양의 테스트 패턴은 테스트 시간을 증가 시킨다. 이러한 이유로 적은 패턴으로 높은 고장 검출 율 획득이 가능한 구조적 테스트(Structural test)를 TIC에 이용 할 수 있는 연구가[14] 진행 되었다. 참고문헌 [14] 에서는 그림 2와 같이 MUX를 이용하여 구조적 테스트 시

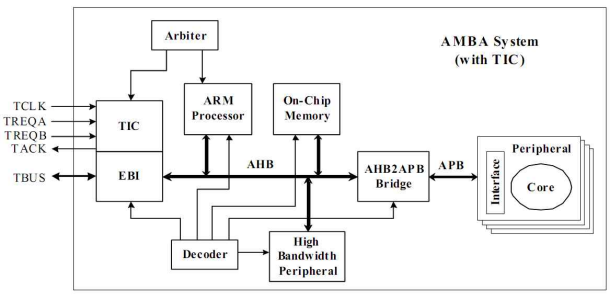

그림 1. AMBA 시스템
Fig. 1. AMBA System.

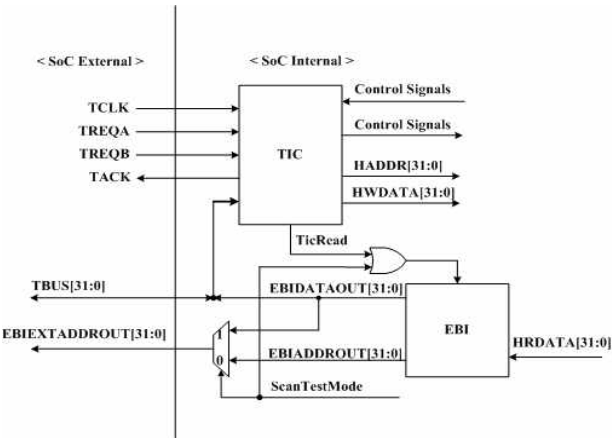

그림 2. 개선된 TIC와 EBI 구조[14]
Fig. 2. Enhanced TIC and EBI structure.

(ScanTestMode = 1)에는 TBUS로는 입력 테스트 패턴 인가를, EBI의 어드레스 버스로는 출력 테스트 패턴 관측이 동시에 수행 가능하게 하고 있다.

본 논문에서도 그림 2의 TIC구조를 구조적 테스트에 사용하며 각 코어는 구조적 테스트를 위한 로직인 스캔 체인을 사용한다고 가정한다.

Ⅲ. IEEE 1500

IEEE 1500은 코어 테스트를 위한 표준으로서 코어 사용자와 코어 제공자 사이의 테스트 인터페이스 역할을 하며 테스트 재사용성을 증가 시킨다. IEEE 1500은 코어 테스트 래퍼를 표준화 하고 있지만, 테스트 제어부와 TAM은 설계자의 몫으로 남겨두고 있다.

TAM은 SoC 외부로부터 테스트 데이터를 받아서 코아 테스트 래퍼에 전달 및 테스트 결과를 SoC 외부로 전달해 주는 역할을 수행하며, 표준에서는 필수 사항인 래퍼 직렬 포트(Wrapper Serial Port(WSP))와 선택 사항인 래퍼 병렬 포트(Wrapper Parallel Port(WPP))를 정의하고 있다.

테스트 제어 부는 표준에서 설계자에게 맡기고 있지만 주로 IEEE 1149.1(JTAG)의 TAP[15]을 이용한 제어 방식[5]이 널리 사용되고 있다.

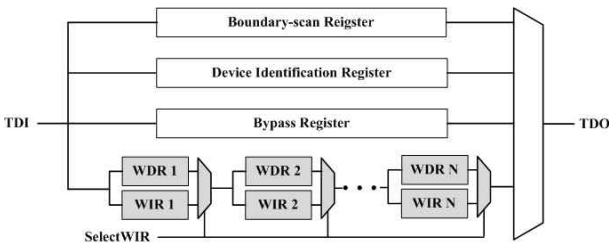

그림 3. IEEE 1149.1의 데이터 레지스터
Fig. 3. Data registers of IEEE 1149.1.

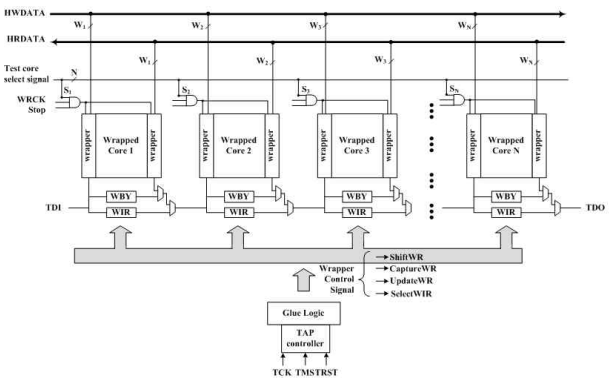

그림 4. 병렬 코어 테스트를 위한 래퍼 연결
Fig. 4. Wrapper connection for parallel testing.

본 논문에서는 코어 테스트 시 IEEE 1500의 표준을 따른다. TAM의 구조는 본 논문에서 제안하는 방식을 사용하는데 WSP는 그림 3과 같이 JTAG의 TDI-TDO를 사용 한다. 이 때 각 코어의 래퍼 레지스터들은 IEEE 1149.1(JTAG)[15]의 사용자 레지스터에 위치(음영 부분) 한다. 그림 3의 구조를 이용하기 위해서는 병렬 테스트 시 JTAG의 데이터 레지스터 중 위의 사용자 레지스터 경로가 선택 되도록 하는 JTAG 사용자 명령어 PROGRAM_WIR가 필요하다. 병렬 테스트 시의 테스트 이동 경로인 WPP에는 AMBA가 이용 되며 구조는 그림 4와 같다.

병렬 테스트 시 테스트 패턴의 인가는 HWDATA로 테스트 패턴의 관측은 HRDATA를 통하여 이루어진다. 테스트 스케줄링에 의하여 결정된 TAM 폭에 맞추어 코어 입력의 래퍼는 HWDATA에 출력의 래퍼는 HRDATA에 각각 연결되어 있으며, WP_INTEST 명령어 로드 후의 제어신호에 의하여 래퍼의 경로가 구성 된다. 코어별 래퍼의 동작 클락은 각 코어 별로 테스트 대상임을 알리는 테스트 대상 코아 선택 신호 (S_N: 그림 4 참조), 테스트 대상 코어 변경 사이 동작 정지를 알리는 신호 (Stop: 그림 4 참조), WRCK를 통하여 생성된다.

테스트 제어부로는 TAP 기반의 제어방식이 사용되며 그림 5는 제어 신호의 하나인 SelectWIR의 각 TAP 상태에 대한 변화를 보여주고 있다.

JTAG의 명령어 레지스터에 PROGRAM_WIR 명령어 로드 후에는 SelectWIR 신호가 high로 된다. 반면 PROGRAM_WIR이 아닌 일반 JTAG 명령어 로드 후와 TAP 상태가 Run-Test / Idle로 진입 후에는 SelectWIR 신호가 low로 변한다. 즉 SelectWIR이 high인 경우에는 TDI와 TDO 사이에는 테스트 대상 코어의 명령어 레지스터의 체인이 연결이 되고, SelectWIR이 low인 경우에는 TDI와 TDO 사이에 테스트 대상 코어의 데이터 레지스터 체인이 연결 된다.

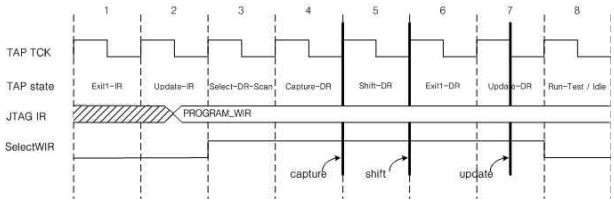

그림 5. TAP 상태에 따른 SelectWIR 신호 변화
Fig. 5. SelectWIR signal of each TAP state.

Ⅳ. AMBA 기반 SoC의 병렬 테스트 설계 기술

1. 병렬 테스트 스케줄링

테스트 스케줄링은 제약 조건(예. SoC의 TAM 폭) 하에서 SoC내의 모든 테스트 대상 코어의 테스트 시간을 최소화 할 수 있는 조합을 찾는 것을 목표로 한다. 병렬 코어 테스트 시 테스트 스케줄링의 적용은 필수적이며, 본 논문에서는 TR-ARCHITECT[3] 알고리즘을 일부 변형하여 사용한다.

그림 6과 같이 SoC에 대한 정보 및 사용자 설정 값(코어 종류, 스케줄 종류, TAM 종류, SoC TAM 폭)을 입력 받아서 SoC 테스트 구조를 최적화 한다. 테스트 스케줄링 적용 후에는 각 코어에 할당이 될 TAM 폭, 동시에 테스트 하게 될 코어, 코어 테스트 순서, 각 코어에 최적화된 래퍼 설계 구조, SoC의 테스트 시간이 결정 된다.

TR-ARCHITECT에서는 전용 TAM을 사용하기 때문에 하나의 코어 테스트가 끝나고, 다른 코어를 테스트하기 위하여 코어를 변경 하는 경우 소모되는 시간은 없다고 가정한다. 반면에 본 논문에서 제시하는 구조는 코어 변경 시에 3 clock cycle이 소모 되므로(4.2 참조) 스케줄링 알고리즘의 변경이 필요하다. 하지만 전체 SoC 테스트 시간에서 코어 변경 시간이 차지하는 비중이 미비하므로 스케줄링 알고리즘의 변경 없이 기존 테스트 스케줄링 결과의 코어 변경 구간마다 3 clock cycle 더하는 방식으로 결과를 얻을 수 있다.

5개의 코어(A~E)와 32bit의 TAM으로 구성된 SoC에 TR-ARCHITECT를 적용하여 그림 7과 같은 테스트 스케줄링 결과를 얻었다고 가정하자. 3개의 TAM

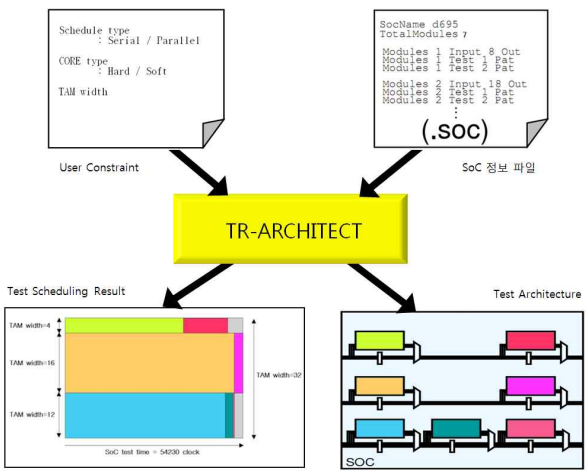

그림 6. TR-ARCHITECT 알고리즘
Fig. 6. TR-ARCHITECT Algorithm.

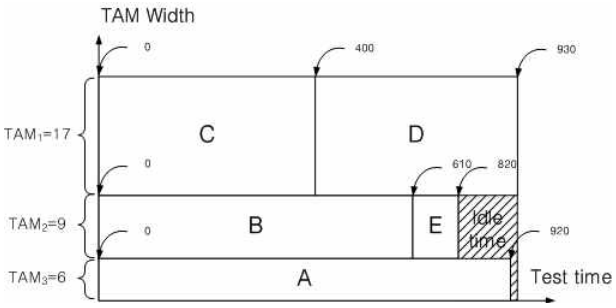

그림 7. 테스트 스케줄링 결과
Fig. 7. Test scheduling result.

(TAM_1, TAM_2, TAM_3)은 독립적으로 병렬 테스트 되며, 각 TAM에서는 여러 개의 코어 중 하나만 테스트가 가능하다. 즉 위의 결과에서 동시에 테스트 되는 코어 수는 3개이며, TAM의 수와 일치한다.

TAM_{start_r}은 TAM r에 속한 코어의 테스트 시작 시간의 집합이라 하면 TAM_{start_1} = {0, 400}이 된다. t(r)을 기존 TAM r의 테스트 시간, c_time(=3 clock cycle)을 코어 변경 시간이라 하면 코어 변경 시간이 반영된 TAM의 테스트 시간 new_t(r)은 다음과 같이 구할 수 있다.

$$new\_t(r) = |TAM_{start\_r}| \times c\_time + t(r) \qquad (1)$$

식에 따라 모든 TAM의 테스트 시간을 새로 구하면, 위 SoC의 테스트 시간은 테스트 시간이 제일 긴 TAM_1에 의하여 결정이 되며 936 clock cycle이 된다.

2. 병렬 테스트를 위한 코어 선택 기술

병렬 코어 테스트 시에는 하나 이상의 코어가 동시에 테스트 되므로 테스트 대상이 되는 코어들을 선택할 수 있는 기술이 필요하다. 본 논문에서는 TIC와 그림 8의 구조를 가지는 TMCS를 이용하여 코어를 선택 하는 방법을 제안한다.

TMCS는 코어에 테스트 대상임을 알리는 신호를 발생 시키는 모듈로 TIC를 통하여 동작 된다. TMCS는 AND 게이트와 총 테스트 대상 코어 수 Cn개의 플립플롭으로 구성이 된다. 각 플립플롭은 테스트 대상 코어

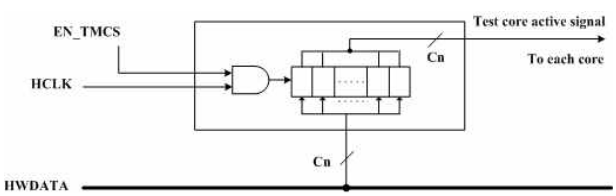

그림 8. 테스트 모드 코어 선택기
Fig. 8. Test Mode Core Selector (TMCS).

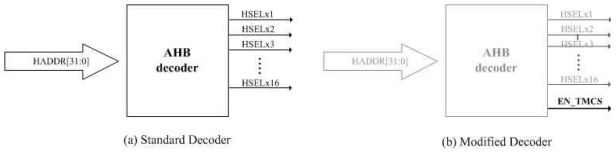

그림 9. AHB 디코더를 통한 EN_TMCS 신호 생성
Fig. 9. EN_TMCS signal generation using AHB decoder.

와 일대일 대응이 되며, 플립플롭에 논리 값 1이 기록이 되면 해당 테스트 대상 코어에 테스트 대상임을 알리는 신호가 전달된다. 플립플롭은 EN_TMCS 신호가 활성화 되어 있는 경우에만 HCLK에 의하여 내부 값이 HWDATA의 값으로 갱신된다.

모듈에 사용되는 신호 중 HCLK와 HWDATA는 AMBA 신호에 속하며, EN_TMCS 신호는 그림 9와 같이 AHB 디코더를 통하여 생성된다. TMCS 로직에는 주소 할당이 되어있으며, 해당 주소가 AMBA 버스에 실리는 경우 AHB 디코더에 의하여 EN_TMCS 신호가 활성화 된다. AHB 디코더의 본래 기능은 버스의 주소를 디코딩하여 해당 슬레이브의 선택 신호 HSELx를 생성해주는 모듈그림 9(a)로서 TMCS의 주소를 받아서 EN_TMCS 신호를 생성하기 위해서는 그림 9(b)와 같은 수정이 필요하다. AHB 디코더의 수정을 원하지 않는 경우에는, 테스트 중에 HSELx 신호를 사용하지 않으므로 신호 중의 하나를 EN_TMCS로 사용 하고, 해당 슬레이브의 주소 범위중의 하나를 TMCS에 할당하여 사용하면 된다.

테스트 대상 코어를 선택하는 과정은 3단계로 이루어진다. 코어가 변경이 될 때 마다 1~3단계의 과정이 반복되며 총 3 clock cycle이 소모 된다.

- 1단계 : TIC를 제어하여 TMCS에 할당 된 주소를 AMBA 버스에 싣는다.
- 2단계 : TIC를 제어하여 Write Transaction을 발생시킨다. 그 결과로 TMCS 내부 플립플롭의 값이 변경되고 해당 테스트 대상 코어 선택 신호가 활성화 된다.
- 3단계 : TIC를 제어하여 TMCS의 주소가 아닌 임의의 주소를 AMBA 버스에 싣는다. 임의의 주소를 사용하는 이유는 코어 선택 이후 이어지는 테스트 벡터 인가를 위해 발생되는 Write Transaction이 TMCS를 변경시키지 않게 하기 위해서이다. 테스트 패턴 인가 시의 Write Transaction 시에는 별도의 버스 주소 없이 TMCS의 주소를 제외한 임의의 주소를 사용하게 된다.

3. 테스트를 정지를 위한 Stop Generator

코어 선택을 통하여 테스트 대상 코어를 변경 중에는 동시에 진행 중인 다른 코어의 테스트에 영향을 주지

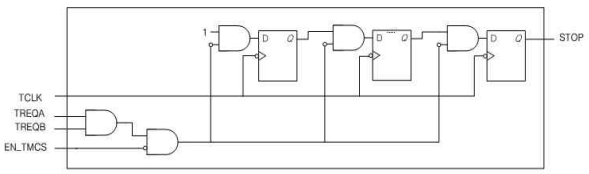

그림 10. 테스트 정지를 위한 Stop Generator
Fig. 10. Stop Generator logic.

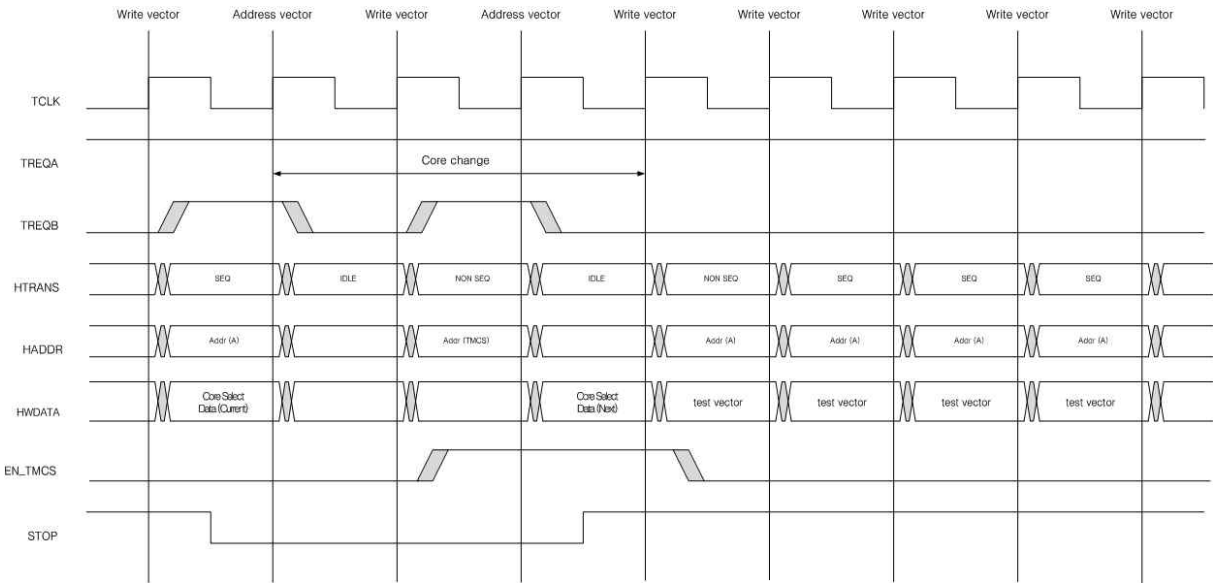

그림 11. Stop generator의 타이밍 도
Fig. 11. timing diagram of Stop generator.

않아야 한다. 본 논문에서는 TIC를 통하여 테스트 패턴의 인가/관측과 테스트 대상 코어 선택을 하고 있다. 하지만 이 두 과정은 동시에 진행 불가능 하므로 코어 변경 시 동시에 진행되고 있는 다른 코어의 테스트에 영향을 주지 않기 위해서는, 코어 테스트를 일시적으로 정지 시켜야 한다.

본 논문에서는 그림 10.의 Stop Generator 로직을 통하여 코어 변경 과정에 소모되는 3clock cycle 동안 테스트가 일시적으로 정지하도록 STOP 신호(= 0)를 발생 시킨다. 이에 대한 동작 파형은 그림 11.에 표시 하였다.

Stop generator에 있는 3개의 플립플롭은 초기 상태로 논리 값 1의 상태를 유지하고 있으며, 코어 변경을 위한 TMCS 주소인가 시에 (TREQA=1, TREQB=1, EN_TMCS=0) 3개의 플립플롭이 리셋 된다. 이후 논리 값 1이 마지막의 플립플롭 까지 도달하는 3 clock cycle 동안 STOP 신호는 논리 값 0을 유지 하게 된다.

4. 병렬 테스트를 위한 TAM으로서의 AMBA 구조

4.4에서는 AMBA 버스를 TAM으로 사용 시에 테스트 벡터의 인가/관측을 위한 경로와, AMBA 표준에 맞는 설계 구조를 제안한다. 설명을 위하여 TIC는 테스트 모드 상태이며, 버스 크기는 32bit 임을 가정한다. 또한 버스에 연결되는 코어들은 그림 7의 테스트 스케줄링 결과를 바탕으로 하고 있다. 또한 전체 설계 구조에서 사용되고 있는 ScanTestMode 신호는 래퍼 명령어 레지스터의 디코더로부터 WP_INTEST 일 때 생성한다.

병렬 테스트 시 테스트 패턴의 인가를 위한 AMBA 버스상의 경로는 그림 12의 구조가 사용된다. 테스트 모드 시에 TIC는 AHB 마스터로서, 테스트 벡터 인가 과정에서 TBUS로 전송 받은 테스트 벡터를 HWDATA로 Write transaction을 발생 시킨다. 테스트 모드 시에는 TIC가 버스 사용 권한을 가지므로 TIC에 연결 된 HWDATA가 Write Data MUX를 지나 브로드캐스팅 된다. HWDATA 구조에 대한 설계 변경은 없으며, 스케줄링 결과에서 같은 TAM에 속해있는 코어들은 HWDATA에서 같은 위치의 bit들과 연결이 된다.

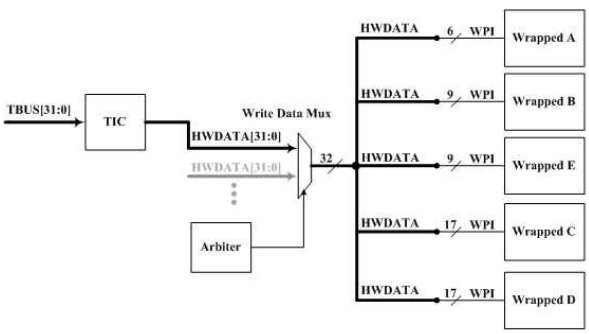

그림 12. 테스트 벡터 인가시의 경로
Fig. 12. path of test vector applying.

테스트 벡터의 인가와 달리 테스트 벡터 응답 관측 시 에는 TIC를 통한 별도의 Read transaction이 필요하지 않다. 매 clock cycle에 테스트 래퍼에 테스트 패턴이 인가 될 때 마다, 응답 결과가 테스트 래퍼의 쉬프트 동작 수행을 통하여 HRDATA에 실리기 때문이다. HRDATA에서는 병렬 테스트를 위하여 약간의 수정이 필요한데, 스케줄링 결과에 따라 각 코어의 테스트 래퍼의 래퍼 병렬 포트 부분을 MUX를 사용하여 32bit의 HRDATA로 재구성하는 작업이 필요하다. 새로 구성된 HRDATA는 ScanTestMode 신호가 활성화 될 때 EBIDATAOUT, EBIEXTADDROUT을 통하여 외부로 연결이 된다.

테스트 패턴의 응답 관측을 위한 경로는 그림 13의 구조가 사용 된다.

AMBA 버스 표준에서는 Write transaction 수행 시 슬레이브의 현재 상태를 HRESP[1:0]와 HREADY 신호를 통하여 마스터에 알리도록 하고 있다. 마스터인 TIC가 테스터 벡터 인가/관측 동안 사용하는 Write

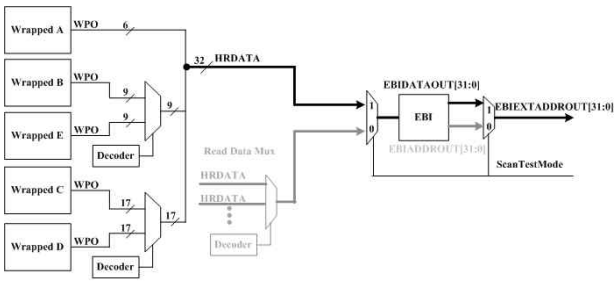

그림 13. 테스트 벡터 관측 시의 경로
Fig. 13. path of test vector observation.

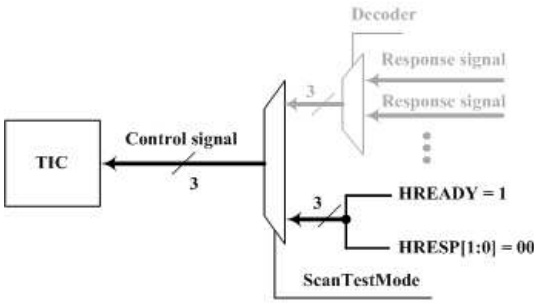

그림 14. 병렬 테스트 시의 응답 신호
Fig. 14. AMBA response signal of parallel testing.

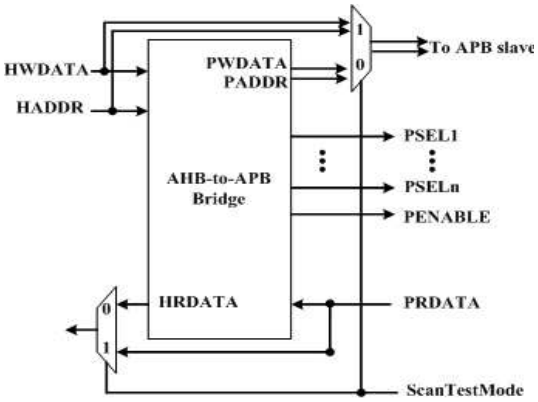

그림 15. AHB와 APB 버스간의 테스트 경로 구성
Fig. 15. test path between AHB to APB.

transaction을 지속적으로 수행하기 위해서는 그림 14와 같이 ScanTestMode 신호 활성화 시 HREADY = 1, HRESP[1:0] = 00을 응답 하도록 해주어야 한다. HREADY = 1은 버스 전송이 끝났음을 알려서 다음 전송이 이루어 질수 있도록 하며, HRESP[1:0]은 전송의 상태에 대한 정보를 나타내며 논리 값 '00'은 버스 전송이 성공적으로 이루어졌음을 나타낸다.

지금 까지는 테스트 대상 코어가 AHB 버스 위에 연결 되어 있다는 가정에서의 테스트 경로를 설명 하였다. 테스트 대상 코어가 저 전력의 주변장치를 위한 버스인 APB에 연결되어 있는 경우에는 AHB와 APB사이의 버스 간 연결을 위하여 AHB-to-APB 브리지가 사용된다. TIC가 브리지를 통하여 APB에 있는 코어에 write transaction을 시도하는 경우 동작 특성 상 데이터를 최소 2 HCLK 동안 유지해야 하며, 이는 병렬 코어 테스트 시 제어를 복잡하게 만든다. 그러므로 그림 15와 같이 MUX를 이용하여 테스트 시 브리지를 우회함으로써 별도의 clock cycle 소모 없이 테스트가 가능하도록 한다.

5. 병렬 테스트를 위한 스캔 제어 신호

본 논문에서는 코어 내부에 구조적 테스트를 위해 스캔 설계를 사용하며, 이러한 로직을 사용하기 위해서는 스캔 제어 신호인 Scan Enable신호를 필요로 한다. 본 논문에서는 병렬 테스트를 위한 Scan Enable 신호를 각 Sub-TAM 별로 하나씩 할당하여 사용한다. 그러므로 필요한 스캔 제어 신호선의 수는 테스트 스케줄링 결과에 따라 동시에 테스트 될 최대 테스트 코어 수와 같다. 모든 코어에 하나의 스캔 제어 신호를 할당 하여 테스트를 수행 할 수도 있지만 이렇게 하는 경우 각 Sub-TAM 별로 스캔 제어 신호를 사용하는 방식에 비

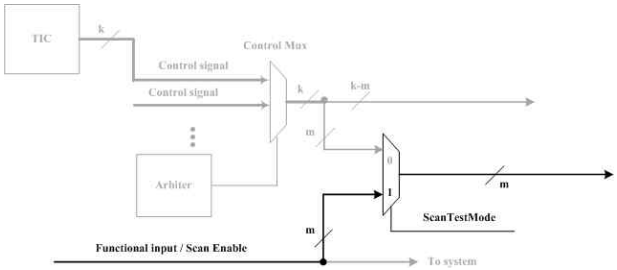

그림 16. Functional input을 스캔 제어 신호로 사용
Fig. 16. reusing functional input to Scan Enable.

하여 테스트 시간이 증가 하게 된다. 증가되는 테스트 시간은 병렬 코어 테스트를 수행하여 감소되는 효과를 감소시키므로 Sub-TAM 별로 스캔 제어신호를 사용하는 것이 좋다. 스캔 제어를 신호를 위하여 전용 신호선을 SoC 외부에 둘 수 있지만, 테스트 시에 사용하지 않는 Functional input을 그림 16과 같이 스캔 제어 신호로 사용 할 수 있다. SoC 내부에서는 TIC에 나오는 AMBA 버스 제어에 사용되는 control 신호의 일부를 스캔 제어 신호로 사용 가능하다.

V. 실 험

제안 하는 AMBA 기반 SoC의 병렬 코어 테스트 설계 기술을 이용 하여 병렬 테스트 수행시의 테스트 시간 감소 효과를 알아보기 위하여 기존 AMBA 기반 SoC의 순차 테스트를 사용한 방법과 테스트 시간을 비교 분석한다. 실험 대상은 ITC'02 테스트 벤치마크 회로[16]를 대상으로 이루어 졌으며, 버스의 폭 Wmax를 16에서 64까지 변화 시켜 가면서 순차 테스트와 병렬 코어 테스트로 나누어 실험 하였다. 순차 테스트를 위해서는 AMBA를 TAM으로 사용하면서 TIC를 통하여 순차 테스트를 수행하는 참고문헌 [14]의 방법을 따른다. 참고문헌 [14]의 실험에서는 코어에 주어진 TAM폭에 대하여 TAM chain 분할을 수행 하지 않으므로, 공정한 비교를 위하여 테스트 대상 코어에 TAM chain 분할을 수행하였다.

표 1과 표 2에서는 ITC'02 테스트 벤치마크 회로에 대한 순차 테스트와 병렬 코어 테스트 시간을 비교 하고 있다. 테스트 시간의 단위는 clock cycle이며, 다양한 테스트 시간 감소 효과를 알아보기 위하여 코어의 스캔 체인 개수 및 길이가 결정되어 있는 경우(하드 코어)와 스캔체인의 변경이 자유로운 경우(소프트 코어)로 나누어 실험 하였다. 본래 ITC'02 테스트 벤치 마크회로는 하드 코어에 대한 정보를 나타내고 있지만 스캔 체

표 1. 벤치마크 회로(소프트 코어)의 테스트 시간
Table 1. Test time of benchmark circuits (soft core).

SoC	W_{max}	순차 테스트	병렬 테스트	감소 비율
d695	16	49105	42046	14.38
	32	27288	21124	22.59
	64	16371	10705	34.61
p22810	16	631976	431409	31.74
	32	397737	219057	44.92
	64	268453	110991	58.66
p34392	16	1105292	971867	12.07
	32	743257	499134	32.85
	64	581105	241296	58.48
p93791	16	2005507	1757689	12.36
	32	1072815	885158	17.49
	64	608300	444282	26.96
u226	16	70260	18669	73.43
	32	59435	10671	82.05
	64	57991	6852	88.18
q12710	16	2016262	1529906	24.12
	32	1025922	766340	25.30
	64	527880	384683	27.13
t512505	16	10991124	10253123	6.71
	32	5542030	5132939	7.38
	64	2819815	2573315	8.74
a586710	16	66461262	41886450	36.98
	32	40727260	21058777	48.29
	64	25905395	11486610	55.66

표 2. 벤치마크 회로(하드 코어)의 테스트 시간
Table 2. Test time of benchmark circuits (hard core).

SoC	W_{max}	순차 테스트	병렬 테스트	감소 비율
d695	16	59291	44328	25.24
	32	45161	21539	52.31
	64	42462	11045	73.99
p22810	16	864474	458146	47.00
	32	701254	222537	68.27
	64	653768	133468	79.58
p34392	16	1916413	1010866	47.25
	32	1769198	551823	68.81
	64	1714304	544624	68.23
p93791	16	2141641	1791725	16.34
	32	1412433	915182	35.21
	64	807039	455813	43.52
u226	16	73224	18666	74.51
	32	60879	10668	82.48
	64	60651	8002	86.81
q12710	16	7261050	2222349	69.39
	32	6755560	2222349	67.10
	64	6501592	2222349	65.82
t512505	16	23189698	10531082	54.59
	32	17754663	5268952	70.32
	64	17651036	5228504	70.38
a586710	16	73608777	41523877	43.59
	32	51338095	22475039	56.22
	64	38247890	12510362	67.29

인 정보를 무시하고 플립플롭 개수 정보를 사용하는 경우 소프트 코어에 대한 실험을 적용 할 수 있다. 또한 ITC'02 테스트 벤치마크 회로는 여러 계층 구조를 가지지만 본 논문의 실험 에서는 TOP 모듈을 제외한 나머지 코어는 같은 계층을 가진다고 가정하였다.

표 1과 표 2의 결과를 보면 버스의 폭이 증가 할수록 병렬 코어 테스트 시간의 감소 효과가 큼을 알 수 있다. 순차 테스트의 경우 코어회로 특성에 상관없이 각 코어에 전체 TAM 버스 폭을 할당하게 되고 버스 폭이 커질수록 병렬 코어 테스트 방식에 비하여 비효율적이 된다. 회로별 감소비율은 테스트 스캐줄링 알고리즘 결과에 의존적이다. 예를 들어 어떤 코아의 스캔체인 길이를 증가시키는 대신 할당된 Sub-TAM 폭을 n-bit 줄이고, 그 줄인 n-bit를 다른 코아의 Sub-TAM에 할당할 경우 테스트 시간이 더욱 줄어들 수 있으며, 이러한 효과가 큰 회로의 경우 감소 비율이 매우 커지게 된다. 또한 표를 비교해 보면 스캔 체인 변경이 가능한 소프트 IP 코아 회로 경우에 비하여 스캔체인이 고정인 하드 IP 코아 회로인 경우가 테스트 감소 효과

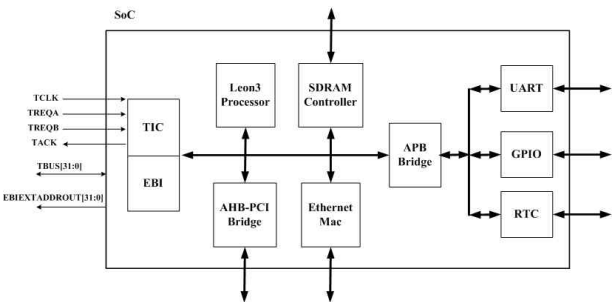

그림 17. AMBA 기반 SoC
Fig. 17. AMBA-based SoC.

가 더 큼을 알 수 있다. 따라서 하드 IP 코어 환경의 경우 더욱 효과적으로 테스트 시간을 감소시킬 수 있음을 알 수 있다.

앞부분에서는 테스트 벤치 마크회로에 대한 실험을 하였으며 그에 대한 결과를 나타내었다. 두 번째 실험에서는 실제 AMBA 기반의 SoC 회로를 바탕으로 테스트 시간과 면적을 순차 테스트 방법을 수행하는 참고문헌 [14]와 비교 한다. 실험을 위한 AMBA 기반 시스템의 구조는 그림 17과 같으며 SoC의 코어 특성은 표 3

표 3. 테스트 대상 코어 특성
Table 3. Core characteristics.

코어 이름		면적		PI	PO	DFF	테스트 패턴	고장 검출율
		스캔 적용 전	스캔 적용 후					
AHB	Leon3 Processor	41901	46303	252	148	1166	386	99.75
	SDRAM Controller	3701	4115	93	119	212	68	99.45
	AHB-PCI Bridge	6364	7055	40	145	275	79	99.92
	Ethernet MAC	32737	35580	109	243	1339	485	99.99
APB	UART	9308	10523	69	32	524	231	98.99
	GPIO	4922	5107	78	104	96	13	100
	RTC	7566	9067	47	32	340	130	99.99

표 4. AMBA 기반 SoC 테스트 시간 비교
Table 4. Test time of AMBA-based SoC.

W_{max}	순차 테스트	병렬 테스트	감소 비율
16	114406	99455	13.07
32	61467	50122	18.46
64	35236	25467	27.72

에 나타내었다.

테스트 패턴은 자동 테스트 패턴 생성(ATPG) 툴을 사용하여 생성하였으며, RTL 코드는 0.25 ㎛ 공정 라이브러리를 사용하여 합성하였다. 세 번째와 네 번째 열은 각 코어의 면적을 2입력 NAND 게이트 기준으로 나타내었다. 열 5, 6, 7은 각각 PI, PO, D-flipflop의 수이며 테스트 벡터의 수와 고장 검출율에 대한 정보는 열 8, 9에 나타내었다. 실험은 테스트 벤치마크 회로와 같은 방식으로 이루어 졌으며, 실험에 사용된 SoC의 코어는 스캔 체인 변경이 자유로운 소프트 코어에 해당한다. 테스트 시간에 대한 실험 결과는 표 4에 나타내었으며, 앞에서와 마찬가지로 버스 폭이 증가 할수록 높은 테스트 시간 감소율을 보이고 있다.

병렬 테스트를 이용하여 테스트 하는 경우에는 기존의 순차 테스트 방식에 비하여 테스트 시간은 감소 하지만 병렬 코어 테스트를 위한 추가 로직으로 인하여 면적이 증가 하게 된다. 전용 테스트 하니스를 사용한 참고문헌 [14]와 본 논문에서 사용한 표준 래퍼의 면적을 표 5에 비교하였다.

참고문헌 [14]에서는 코어의 출력에 테스트 하니스를 연결하지 않는 반면에 본 논문에서는 입력과 출력 모두

표 5. AMBA 기반 SoC 테스트 래퍼 면적 비교
Table 5. Comparison of AMBA-based SoC area.

코어 이름		전용 테스트 하니스 면적	표준 래퍼 면적	증가 비율
AHB	Leon3 Processor	3975	4648	16.93
	SDRAM Controller	2041	2643	29.50
	AHB-PCI Bridge	1891	2355	24.54
	Ethernet MAC	3027	4136	36.64
APB	UART	1271	1459	14.79
	GPIO	1785	2323	30.14
	RTC	988	1224	23.89
합계/평균 증가율		14,978	18,788	25.44

에 표준 래퍼를 연결하고 있기 때문에 표준 래퍼의 면적이 크게 나왔다. 래퍼 면적을 고려하여 실제적인 적용 시에는 면적 오버헤드와 테스트 타임 비용간의 trade-off를 고려하여야 한다. 예를 들어 SoC die 면적이 테스트 래퍼에 의해 증가하지 않는 다면 테스트 비용을 감소시키기 위해 실제적인 적용이 가능하다. 최근에는 코아 테스트 래퍼를 EDA tool에서 지원해 주고 있다. 래퍼를 제외하고 기존 방식과 달라지는 면적을 비교 한 결과 기존 방식 보다 게이트 수로 224 만큼 증가하였다. 이를 통하여 제안 하는 AMBA 기반의 병렬 코어 테스트 설계 기술은 기존의 방법에 비하여 적은 면적 증가로 높은 테스트 시간 감소 효과를 낼 수 있음을 알 수 있다.

Ⅵ. 결 론

본 논문에서는 AMBA 버스 기반의 SoC에서 AMBA를 코어 테스트 시의 TAM으로 재사용 하는데 있어서 AMBA 통신 규약을 어기지 않으면서 병렬 코어 테스트가 가능 하도록 한 설계 기술을 제시 하였다.

기능 테스트시의 AMBA 버스 제어를 위해 설계 된 TIC를 구조적 테스트 시의 제어에 재활용 하여 병렬 테스트의 제어에 사용되는 추가 로직을 최소화 하였으며, 병렬 테스트를 위한 TAM 구조, 테스트 수행에 필요한 추가 로직을 제안하고 있다. 이를 통하여 기능적 테스트를 수행할 수 있을 뿐만 아니라 구조적 테스트 시 병렬 테스트를 수행 할 수 있어서 SoC의 신뢰성 확보와 테스트 시간 단축에 기여 할 수 있다.

참 고 문 헌

[1] S. Narayanan, R. Gupta, M.A. Breuer, “Optimal configuring of multiple scan chains,” IEEE Trans. Computers, pp. 1121 - 1131, Sept. 1993.

[2] S. Zhang, M. Choi, N. Park, F. Lombardi, “Cost-Driven Optimization of Coverage of Combined Built-In Self-Test/Automated Test Equipment Testing,” IEEE Trans. Instrum. Meas., pp. 1094-1100, June. 2007.

[3] S.K. Goel, and E.J. Marinissen, “Effective and efficient test architecture design for SOCs”, Proc. IEEE International Test Conference, pp. 529-538, Oct. 2002.

[4] Y. Zorian, E. J. Marinissen and S. Dey, “Testing Embedded-core based System Chips,” Proceedings of IEEE International Test Conference, pp. 130-143, Oct. 1998.

[5] IEEE std 1500 Standard for Embedded Core Test, http://grouper.ieee.org/groups/1500/

[6] J. Aerts, E.J. Marinissen, “Scan Chain Design for Test Time Reduction in Core-Based ICs,” Proceedings of IEEE International Test Conference, pp 448-457, Oct. 1998.

[7] P. Varma, S.Bhatia, “A structured Test Re-Use Methodology for Core-Based System Chips,” Proceedings of IEEE International Test Conference, pp 294-302, Oct. 1998.

[8] S. Jaehoon, M. Piljae, Y. Hyunbean, P. Sungju, “Design of Test Access Mechanism for AMBA Based System-on-a-Chip,” IEEE VTS, pp. 375-380, May. 2007.

[9] C. Lin and H. Liang, “Bus-Oriented DFT Design for Embedded Cores,” IEEE Asia-Pacific Conference, pp. 561-563, Dec. 2004.

[10] C. Feige et al, “Integration of the Scan-Test Method into an Architecture Specific Core-Test Approach,” Journal of Electronic Testing, pp. 125-131, July. 1998.

[11] M. Abramovici, M. Breuer, and A. Friedman, “Digital Systems Testing and Testable Design”, Computer Science Press, New York, 1991.

[12] E.J. Marinissen, S.K. Goel, and M. Lousberg, “Wrapper design for embedded core test”, Proc. International Test Conference, Oct. 2000, pp 911-920.

[13] ARM, “AMBA specification (rev. 2.0),” May 1999.

[14] 민필재, 송재훈, 이현빈, 박성주, “AMBA 기반 SoC테스트를 위한 접근 메커니즘 설계”, 대한전자공학회 논문지, 43권, 10호, 2006년 10월.

[15] IEEE std 1149.1, “IEEE Standard Test Access Port and Boundary-Scan Architecture”.

[16] E. J. Marinissen, V. Iyerngar, and K. Charkrabarty, ITC'02 SOC Test Benchmarks, http://itc02socbenchm.pratt.duke.edu/

저 자 소 개

송 재 훈(정회원)

2000년 한양대학교 전자컴퓨터 공학과 학사 졸업.
2002년 한양대학교 컴퓨터공학과 석사 졸업.
2003년 서울대학교 SoC 설계 센터 연구원.
2009년 한양대학교 컴퓨터공학과 박사 졸업.
2009년～현재 트란소노 책임연구원
<주관심분야 : SoC 설계 및 테스트, 테스트를 고려한 설계>

오 정 섭(학생회원)

2010년 한양대학교 전자컴퓨터 공학부 학사 졸업.
2010년～현재 한양대학교 컴퓨터 공학과 석사 재학.
<주관심분야 : Scan Design, Network on Chip, SoC DFT>

박 성 주(평생회원)

1983년 한양대학교 전자공학과 학사 졸업.
1983년～1986년 금성사 소프트웨어개발 연구원.
1992년 Univ. of Massachusetts 전기/컴퓨터공학과 박사 졸업.
1992년～1994년 IBM Microelectronics 연구스텝.
1994년～현재 한양대학교 전자컴퓨터공학부 정교수.
<주관심분야 : 테스트 합성, Built-In Self Test, Scan Design, ATPG, ASIC 설계, 고속 신호처리 시스템 설계, 그래프 이론>