논문 2011-48SD-7-9

RISC 프로세서의 디버거를 위한 변형된 JTAG 설계

(Design of Modified JTAG for Debuggers of RISC Processors)

허 경 철*, 박 형 배**, 정 승 표*, 박 주 성***

(Jingzhe Xu, Hyungbae Park, Seungpyo Jung, and Jusung Park)

요 약

SoC 설계기술이 발전함에 따라 디버깅이 차지하는 비중은 더욱더 증가되고 있으며 사용자는 빠르고 정확한 디버거를 원하고 있다. 본 논문에서는 새로 설계되는 RISC 프로세서에 적용할 디버거를 위한 변형된 JTAG을 제안 및 설계하여 디버깅 기능 수행에 필요한 사이클을 줄임으로써 빠른 디버거를 구현하였다. 구현된 JTAG은 Core-A의 OCD에 내장하여 SW 디버거와 연동하여 FPGA 레벨까지 검증 마치고 디버거로서의 기능 및 신뢰성을 확인하였다. Core-A의 OCD에 내장된 제안한 JTAG은 기존의 JTAG과 비교하였을 경우, 디버깅 수행 사이클은 수행되는 디버깅 기능에 따라 약 8.5~72.2% 감소되고 추가적으로 게이트 카운트도 약 31.8%감소되었다.

Abstract

As the technology of SoC design has been developed, the debugging is more and more important and users want a fast and reliable debugger. This paper deals with an implementation of the fast debugger which can reduce a debugging processing cycle by designing a modified JTAG suitable for a new RISC processor debugger. Designed JTAG is embedded to the OCD of Core-A and works with SW debugger. We confirmed the functions and reliability of the debugger. By comparing to the original JTAG system, the debugging processing cycle of the proposed JTAG is reduced at 8.5~72.2% by each debugging function. Further more, the gate count is reduced at 31.8%.

Keywords: JTAG, TAP block, On-Chip Debugger, Core-A, RISC processor

Ⅰ. 서 론

최근 반도체 설계 및 공정기술의 발달로 인하여 SoC(System on Chip)가 보편화되고 있다. SoC에서 가장 핵심적인 부분은 내장형 프로세서이고 특히 RISC 프로세서가 가장 많이 사용되고 있다. 그 이유는 수많은 SoC칩이 RISC 프로세서를 내장하고 있고 프로세서가 두뇌 역할을 하기 때문이다. 프로세서의 디버거는 프로세서뿐만 아니라 시스템에 내장된 주변장치도 디버깅 및 제어할 수가 있다. 그러므로 SoC 를 디버깅함에 있어서 프로세서를 디버깅할 수 있는 디버거가 매우 중요하다[1].

고집적도의 SoC 및 SoC의 성능을 최대로 발휘하여 효율적으로 구동시키는 어플리케이션(application)을 검증함에 있어서 소모되는 비용과 시간은 점점 늘어나고 있다. 즉 이는 SoC 설계기술의 발전으로 인하여 전체 개발시간이 줄어들고 있지만 반대로 디버깅이 차지하는 시간은 더 늘어나고 그 비중이 증가하고 있다는 것을 말해준다. 그러므로 이런 원인 때문에 SoC를 설계함에 있어서 보다 빠르고 정확한 디버거를 구현하는 것이 중요한 문제로 부각되고 있다. 그리하여 본 논문에서는 RISC 프로세서 디버거의 디버깅 속도를 향상시킬 수 있는 방법-변형된 JTAG을 제안하였다.

본 논문에서는 IEEE 1149.1에서 제시한 JTAG(Joint

* 학생회원, ** 정회원, *** 평생회원, 부산대학교 전자전기공학과
(Department of Electrical & Electronic Engineering, Pusan National University)
※ 이 논문은 부산대학교 자유과제 학술연구비에 의하여 연구되었음.
접수일자: 2010년12월31일, 수정완료일: 2011년6월17일

Test Action Group)[2],[3]을 RISC processor의 디버거에 적합하게 변형시킴으로써 디버깅 속도를 빠르게 하여 디버거의 성능을 향상시켰을 뿐만 아니라 추가적으로 게이트 카운트도 감소시켰다. 그리고 설계된 JTAG은 특허청 사업의 지원 하에 개발된 한국형 임베디드(embedded) 프로세서인 Core-A[4]를 타깃(target) RISC 프로세서로 선택하고 Core-A의 On-Chip Debugging System[5]에 적용하여 검증을 진행하였다. Core-A는 32bit RISC 타입 하바드(harvard) 구조, 5단 파이프라인(pipeline)을 가지는 프로세서이다.

본 논문의 구성은 다음과 같다. Ⅱ장에서는 기존의 JTAG을 설명하고 Ⅲ장에서는 제안하는 변형된 JTAG에 관하여 자세히 소개한다. Ⅳ장에서는 변형된 JTAG의 구현 및 성능비교를, Ⅴ장에서는 구현된 JTAG의 검증에 대해 소개하고 마지막 Ⅵ장에서는 결론을 맺는다.

Ⅱ. JTAG

시스템 설계는 점차적으로 칩 수준의 설계로 진화됨에 따라 디버깅 기법 또한 기존의 bed-of-nails와 같은 보드 수준의 방식이 아닌 JTAG 기반 방식이 많이 사용되고 있다. JTAG은 IEEE 1149.1 “Standard Test Access Port and Boundary Scan Architecture” 표준이다. JTAG은 in-circuit 검증에 있어서 지배적인 표준으로 사용되고 있다. 그 원인은 JTAG이 경제적이고 시스템에 쉽게 적용되며 사용자의 수요에 따라 용이하게 확장할 수 있기 때문이다. 그리고 JTAG은 4개의 입력 핀 TDI, TMS, nTRST 및 TCK와 1개의 출력 핀 TDO로 이루어져 있어 적은 핀으로 SW(software) 디버거와 통신하여 HW(hardware) 디버거를 제어할 수 있는 장점도 있다.

Core-A의 디버그 시스템은 참고문헌 [5]에서 설명된 바와 같이 프로세서에 내장되는 OCD, 프로세서 상태 및 디버깅 결과를 모니터링(monitoring)하는 SW 디버거 및 OCD와 SW 디버거를 연결해주는 Emulator Board로 구성되어 있으며 그 구조도는 아래의 그림 1과 같다. Core-A에 내장된 OCD는 참고문헌 [1]에 자세히 설명되어 있다. OCD는 DCU(Debug Control Unit), OCE(On-Chip Emulator)[1],[6] 및 JTAG으로 구성되어 있다. 그중 JTAG은 OCD와 Emulator Board를 연결해주어 SW 디버거와 통신한다. SW 디버거에서 디버깅 명령을 보내면 Emulator Board가 그 명령을 해석하고

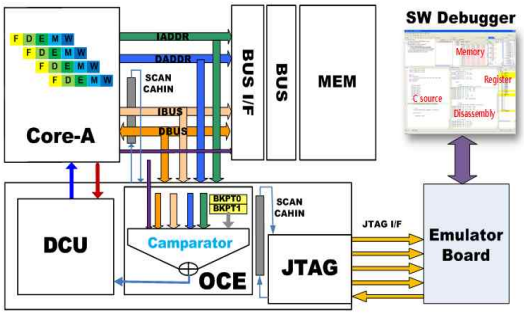


그림 1. Core-A 디버그 시스템
Fig. 1. Debug system of Core-A.

JTAG 신호로 변환시켜 준다. 이어서 변환된 JTAG 신호는 OCD의 JTAG 블록으로 전달되고 JTAG은 그 신호에 따라 OCD를 제어하여 디버깅 기능을 구현한다.

RISC 프로세서의 디버거를 제어하기 위한 JTAG 블록도는 아래의 그림 2와 같다. JTAG 블록은 TAP(Test Access Port), TAP 제어기, 명령어 디코더(decoder), 2개의 스캔 체인(scan chain) 및 특수목적 레지스터(register)로 구성되어 있다. 스캔 체인은 타깃 프로세서에 따라 서로 다르게 구현된다. 때문에 JTAG은 스캔 체인을 제외한 나머지 블록들을 포함하는 TAP 블록 및 스캔 체인 등 2개의 파트(part)로 나눌 수 있다. 스캔 체인은 타깃 프로세서에 따라 설계가 달라질 수가 있기에 본 논문에서 제안한 JTAG은 스캔 체인을 제외한 TAP 블록을 변형하여 설계를 진행한다.

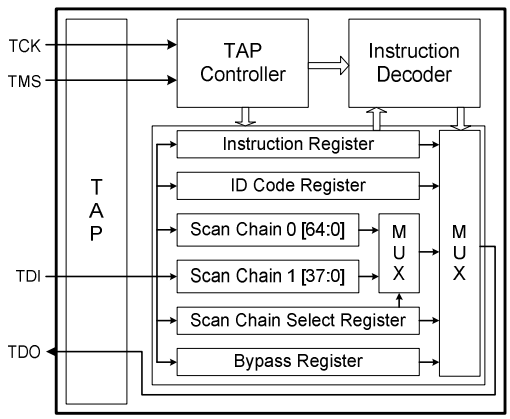


그림 2. JTAG 블록도
Fig. 2. JTAG block diagram.

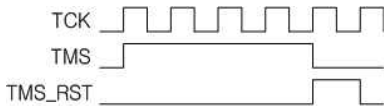


그림 3. TMS_RST 생성
Fig. 3. Generation of TMS_RST.

TAP 블록은 TAP, TAP 제어기, 명령어 디코더 및 특수목적 레지스터들로 구성되어 있다. TAP은 5개의 포트를 지원하지만 구현되는 TAP은 3개의 입력 포트 TDI, TMS, TCK와 1개의 출력 포트 TDO 즉 nTRST 포트를 제외한 4개의 포트로 구성되어 있다. nTRST는 JTAG 블록을 초기화하는 포트이지만 TAP 제어기에 nTRST와 같은 동작을 하는 TMS_RST신호를 구현하여 nTSRT 포트를 제거하였다. TMS_RST신호는 아래의 그림 3과 같이 TCK 4 사이클(cycle)동안 TMS를 인가하면 생성된다.

TAP 제어기는 JTAG 블록의 모든 동작을 제어하며 그 동작은 아래의 그림 4에서 보이는 상태 천이에 따라 구현된다. TAP 제어기 상태는 Test Logic Reset, Run Test/Idle 상태와 DR(Data Register) 제어, IR (Instruction Register) 제어를 위한 상태 각각 7개의 상태로, 총 16개의 상태로 구성되어 있다. IR은 JTAG 명령어를 저장하는 특수목적 레지스터이고 DR은 IR를 제외한 나머지 특수목적 레지스터 및 스캔 체인을 가리킨다. 본 논문에서는 DR에 값을 인가하는 과정을 데이터 이동과정, IR에 값을 인가하는 과정을 JTAG 명령어 디코딩 과정이라고 정의한다.

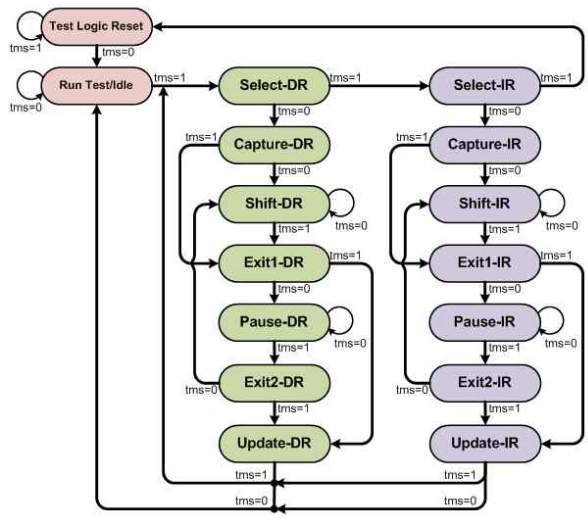


그림 4. TAP 제어기 상태도
Fig. 4. State diagram of TAP controller.

표 1. JTAG 명령어
Table 1. JTAG instruction.

JTAG 명령	IR Value	Register
INTEST	4'b1100	스캔 체인 0/1
EXTEST	4'b0000	스캔 체인 0/1
IDCODE	4'b1110	ID register
BYPASS	4'b1111	BYPASS register
SCAN_N	4'b0010	스캔 체인 선택 register
RESTART	4'b0100	BYPASS register

Core-A의 OCD에 내장된 기존의 JTAG이 지원하는 JTAG 명령어와 해당 명령어에 따라 TDI, TDO사이에 연결되는 레지스터는 표 1과 같다

SCAN_N 명령어는 스캔 체인 선택을 위한 명령어로서 스캔 체인 선택 레지스터를 TDI와 TDO사이에 위치하게 하고 스캔 체인 0과 스캔 체인 1을 선택하는 기능을 수행한다. INTEST와 EXTEST 명령어는 선택된 스캔 체인에 값을 인가하는 기능을 수행하고 IDCODE와 BYPASS는 각각 IDCODE 레지스터와 BYPASS 레지스터를 TDI와 TDO사이에 위치하게 한다. RESTART 명령어는 프로세서를 디버그 모드에서 탈출하게 하는 명령이다.

Ⅲ. 제안한 TAP 블록

제안한 TAP 블록은 디버깅 기능을 수행하는데 필요한 사이클을 감소하기 위하여 TAP 제어기 상태 제거 및 JTAG 명령어 추가 구현 등 2가지의 방법을 사용하였다.

1. TAP 제어기 상태 제거

TAP 제어기는 DR 제어 상태와 IR 제어 상태로 나뉜다. 분석을 통하여 RISC 프로세서의 디버거를 제어함에 있어서 필요로 하지 않는 상태가 있다는 것을 확인하였다. 예를 들면 IR 제어 상태를 이용한 JTAG 명령어 디코딩 과정의 경우 Exit1-IR, Pause-IR, Exit2-IR 등과 같은 상태는 특별한 기능을 제공하지 않는다. 이런 불필요한 상태를 제거함에 따라 명령어 인가에 필요한 사이클 소모가 줄어 됨을 확인하였다.

본 절에서는 TAP 제어기 상태를 제거하면서 RISC 프로세서의 디버깅 기능을 100% 지원하는 변형된 TAP 제어기를 소개한다. 불필요한 상태 확인을 위하여 DR와 IR 제어 방법을 나뉘어 분석하였다.

가. DR 제어 분석

DR 제어는 TAP 블록 내부의 DR 레지스터와 타깃 프로세서에 연결되어 있는 스캔 체인을 제어하는 것이다. Pause-DR 상태는 특별히 수행하는 기능이 없고 다만 구현하는 스캔 체인의 bit수가 스캔 체인과 연결되

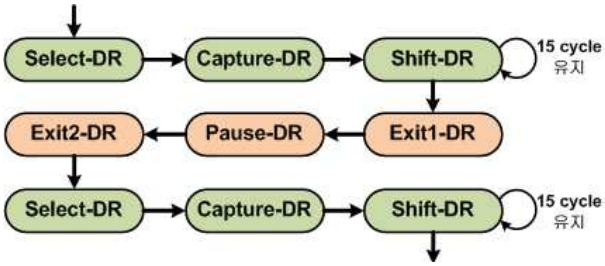


그림 5. Pause-dr를 사용하는 상태 천이도
Fig. 5. State transfer diagram of using pause-dr.

어 있는 버스의 data width보다 작을 때 사용되는 것이다. 32bit RISC 프로세서를 예로 들면 다음과 같다. 32bit 프로세서의 버스의 data width는 32이다. 이 경우, 버스에 액세스(access)하는 스캔 체인의 bit수를 32로 하면 Shift-DR 상태를 31 사이클 유지하여 원하는 데이터를 한 번에 버스에 인가하거나 버스에서 추출할 수가 있다. 하지만 스캔 체인의 bit수를 32보다 작은 16으로 하면, Shift-DR 상태를 15사이클 유지하여 데이터 교환하는 작업을 두 번 수행하여야 완성할 수 있으며 그 과정은 그림 5와 같다.

위의 과정에서 보여주는 바와 같이 두 번의 shift 작업사이에 Exit1-DR, Pause-DR, Exit2-DR 상태를 거쳐야 한다. 다시 말하여 RISC 프로세서의 디버거에서 스캔 체인의 bit수를 버스의 data width로 구현하면 Pause-DR 상태는 사용하지 않아도 된다.

Exit-DR 상태는 Exit1-DR와 Exit2-DR로 2개가 있는데 이는 Shift-DR, Pause-DR, Update-DR 상태간의 천이를 최소 사이클로 가능하게 하기 위한 것이다. 그런데 Pause-DR를 제거함으로 인해 Exit-DR 상태는 하나로만 가능하기에 하나는 제거할 수가 있다. 즉 DR 제어를 위한 7개의 상태가 Pause-DR, Exit1-DR 상태를 제거함으로써 5개로 줄일 수가 있다.

나. IR 제어 분석

IR 제어는 IR에 JTAG 명령어를 인가하는 동작만 수행하고 스캔 체인에 연결되어 있는 IR 레지스터의 값을 추출하여 TDO로 출력하는 동작은 필요로 하지 않는다. 그러므로 레지스터의 값을 스캔 체인으로 로드하는 Capture-IR 상태는 제거하여도 된다. 그리고 IR 제어도 DR 제어와 마찬가지로 Pause-IR 상태를 제거하여도 IR 제어의 기능을 그대로 구현할 수 있다. 즉 IR 제어 상태에서 Capture-IR, Pause-IR, Exit1-IR는 필요하지 않는 상태이다.

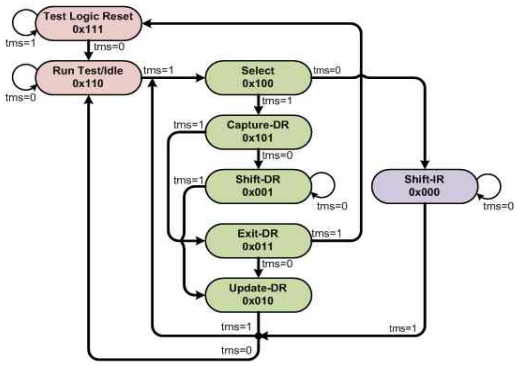


그림 6. 제안한 TAP 제어기 상태도
Fig. 6. State diagram of proposed TAP controller.

위의 분석 결과를 토대로 하여 재구성한 TAP 제어기의 상태 천이는 그림 6과 같다. IR 제어에는 Shift-IR를 제외한 나머지 상태는 모두 제거하였다. 제안한 TAP 제어기 상태는 디버깅 기능을 수행함에 있어서 기존의 TAP 제어기와 100% 호환된다.

제안한 TAP 제어기는 불필요한 상태 제거로 인하여 기존의 TAP 제어기와 비교하였을 때 두 가지 장점이 있다. 첫째로 DR, IR 제어에 필요한 사이클 수가 감소되었다. 상태 제거로 인하여 DR, IR 제어에 필요한 상태 천이가 감소되면서 사이클 수도 감소된다. 두 번째는 게이트 카운트가 감소되었다. 기존의 TAP 제어기는 16개의 상태였지만 제안한 제어기는 8개의 상태로 줄어들면서 게이트 카운트가 많이 줄어들었다.

2. JTAG 명령어 추가

OCD가 제공하는 디버깅 기능 중 스캔 체인을 변경하는 기능의 사용 빈도가 매우 높다. 즉 스캔 체인 변경에 소요되는 사이클을 줄이면 디버깅 성능 향상에 많은 도움이 된다. TAP 제어기의 IR 제어 부분을 간략하여 스캔 체인 변경이 기존보다 많이 빨라졌다. 하지만 더 적은 사이클의 소모로 스캔 체인 변경을 꾀하기 위하여 본 논문에서는 스캔 체인을 변경하는 방법에 관하여 연구하였다.

기존의 스캔 체인 변경하는 방법은 IR에 SCAN_N JTAG 명령어를 삽입하고 스캔 체인 선택 DR 레지스터에 원하는 스캔 체인 번호를 삽입하는 것으로 수행된다. 즉 스캔 체인을 변경하려면 JTAG 명령어 디코딩 과정과 데이터이동과정을 모두 수행하여야 된다. 만약 데이터이동과정을 거치지 않고 JTAG 명령어 디코딩과정만으로 스캔 체인 변경이 가능하면 사이클 수가 많이

감소될 것이다.

Core-A의 OCD에 구현된 JTAG 명령어는 6개로 IR를 3bit로 지정하면 구현할 수 있으며 2개의 명령어를 추가로 구현할 수 있는 여유가 있다. 또한 Core-A의 스캔 체인 개수는 2개로, 만약 스캔 체인을 선택하는 동작을 JTAG 명령어로 구현하면 IR 레지스터 bit수가 증가하지 않는 전제하에서 스캔 체인을 변경하는 데 소모되는 사이클을 줄일 수가 있다. 즉 새로운 방법은 기존의 JTAG 명령어 디코딩과정과 데이터이동과정을 통하여 스캔 체인을 변경하는 방법에서 JTAG 명령어 디코딩과정으로만 가능한 방법이기에 수행 사이클이 획기적으로 감소될 수 있다.

Ⅳ. 구현 및 성능 비교

본 논문에서 제안한 변형된 JTAG은 TAP 제어기의 상태 감소와 스캔 체인을 선택하는 JTAG 명령어의 추가로 구현된다. 이렇게 변형함으로써 RISC 프로세서의 디버거가 타깃을 디버깅 할 때 수행 사이클이 줄어들고 TAP 블록의 게이트 카운트 감소되는 결과를 꾀하였다.

1. 구현

제안한 TAP 제어기의 상태는 기존의 16개의 상태에서 8개 상태로 줄어들었다. 즉 상태 천이를 위한 FSM 구현은 기존의 4bit register가 아닌 3bit register로 가능하다. 그리고 DR, IR 제어 상태의 감소로 인하여 해당 레지스터 제어 신호도 감소된다.

구현한 TAP 제어기의 내부 구조는 아래의 그림 7과 같으며 왼쪽 파트는 기존의 TAP 제어기이고 오른쪽이 변형된 TAP 제어기이다. 변형된 IR 제어는 Shift-IR 상태 하나로만 구현 가능하기에 3bit의 IR를 TAP 제어기는 shift-ir 신호로 동작시킨다. 즉 기존의 것과 비교하여 보면 IR가 4bit에서 3bit로 감소되고 제어신호도 7

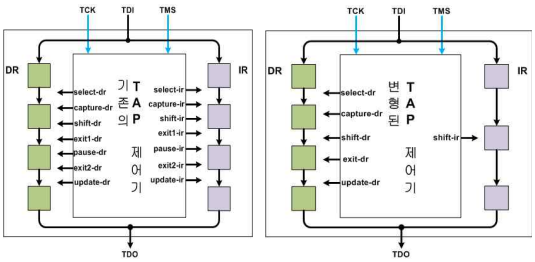


그림 7. TAP 제어기 내부 구조
Fig. 7. Internal structure of TAP controller.

표 2. 변경된 JTAG 명령어
Table 2. Modified JTAG instruction.

JTAG 명령	IR Value	Register
INTEST	3'b000	스캔 체인 0/1
EXTEST	3'b001	스캔 체인 0/1
IDCODE	3'b010	ID register
BYPASS	3'b011	BYPASS register
SCAN_N	3'b100	스캔 체인 선택 register
RESTART	3'b101	BYPASS register
SCAN0	3'b110	스캔 체인 0 선택
SCAN1	3'b111	스캔 체인 1 선택

개에서 Shift-IR 하나만 구현하였다. DR 제어도 마찬가지로 7개의 제어신호에서 Pause-DR, Exit1-DR 신호를 제외한 나머지 5개 신호를 구현하였다. 결과적으로 제안한 TAP 제어기는 기존의 제어기보다 게이트 카운트가 감소되고 상태간의 천이가 빠르기 때문에 수행 사이클도 따라서 감소된다.

기존의 JTAG 명령어에 스캔 체인을 변경하는 명령어 SCAN0, SCAN1을 추가한 JTAG 명령어는 아래의 표 2와 같다. SCAN0은 스캔 체인 0을 선택하고 SCAN1은 스캔 체인 1을 선택하는 JTAG 명령어이다. Core-A의 OCD는 2개의 스캔 체인만 가지고 있기에 SCAN_N 명령어가 없어도 SCAN0, SCAN1 명령어로 가능하지만 나중에 스캔 체인 확장 가능성을 위하여 SCAN_N 명령어를 그대로 남겨두었다. 추후에 스캔 체인이 추가되면, 그 스캔 체인의 사용 빈도에 따라 JTAG 명령어를 추가할지 아니면 기존의 스캔 체인 변경 방법을 채택할지를 결정하면 된다.

2. 성능 비교

참고문헌 [7], [8]에서 구현한 PIDM방식 TAP은 아

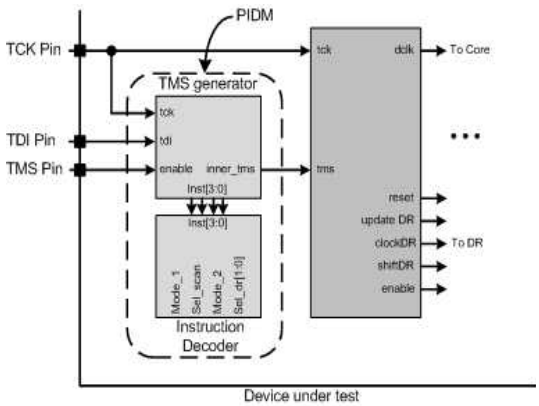


그림 8. PIDM 블록도
Fig. 8. PIDM block diagram.

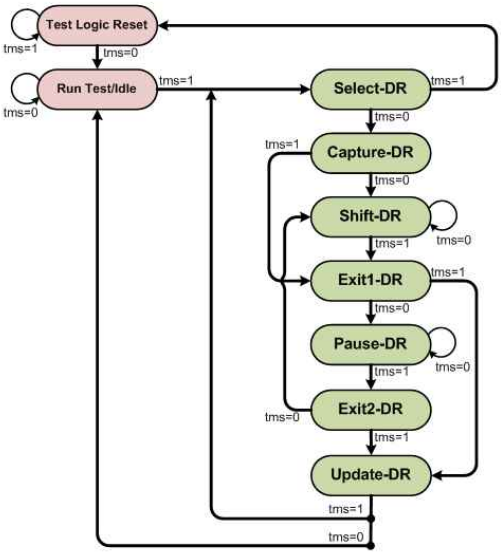


그림 9. PIDM TAP 제어기 상태도
Fig. 9. State diagram of PIDM TAP controller.

래의 그림 8과 같이 PIDM블록과 TAP 제어기로 구성되어 있으며 PIDM방식에 관련된 최근 연구이다.

PIDM방식은 디버깅 기능 수행 사이클을 감소하기 위하여 IR 제어에 필요한 신호를 모두 제거하였으며 아래의 그림 9와 같은 TAP 상태천이도에 의하여 동작한다. PIDM방식은 IR 제어 신호가 없기 때문에 Test Logic Reset 상태에서 IR를 입력하고 이를 위하여 4bit 카운터가 추가로 내장된다.

제안한 TAP 블록을 구현하고 기존의 TAP 블록 및 PIDM방식[7~8]과 성능비교를 진행하였다. 세 가지 TAP 블록을 Core-A의 OCD에 각각 적용하여 디버거가 디버깅 기능을 수행하는데 소요되는 사이클을 서로 비교하여 성능을 평가하였다. 사이클은 디버깅 기능 수행에 소요되는 것이기 때문에 TCK 한 주기를 단위로 정하였다.

아래에 스캔 체인 0이 선택된 상황에서 스캔 체인 1을 선택하는 동작에 필요한 사이클 즉 스캔 체인 변경에 소요되는 사이클을 카운팅(counting)하는 방법을 예로 하여 자세히 소개한다. TAP 제어기 상태에서 Run Test/Idle 상태가 TAP의 초기 상태이기 때문에 사이클 카운팅은 Run Test/Idle 상태에서 출발하여 다음 동작의 시작점인 Select-DR 상태로 돌아오는데 소요되는 사이클을 말한다.

가. 기존의 TAP 블록

기존의 TAP 블록을 이용하여 스캔 체인을 변경하는

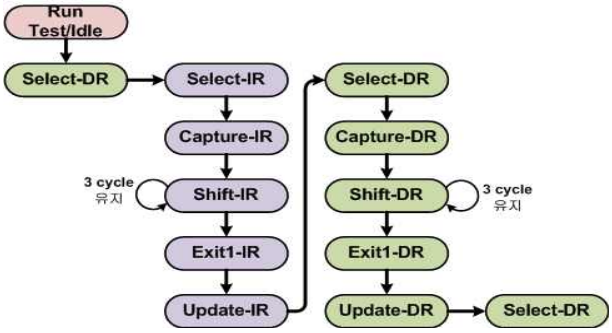


그림 10. 스캔체인변경의 상태천이도(기존의 TAP블록)
Fig. 10. State transfer diagram of changing scanchain (original TAP block).

방법은 IR에 SCAN_N(4'b0010) JTAG 명령어를 삽입하고 스캔 체인 선택 DR 레지스터에 원하는 스캔 체인 번호를 삽입하는 것으로 수행되며 그 과정은 아래의 그림 10과 같다. 상태 천이 과정을 보면 Run Test/Idle 상태에서 시작하여 Select-DR 상태사이에 IR 제어와 DR 제어를 통과하게 된다. 그리고 Shift-IR 상태가 3 사이클 유지될 때 SCAN_N 명령인 4'b0010이 IR에 인가하게 되고 Shift-DR가 3 사이클 유지될 때 스캔 체인 1을 나타내는 4'b0001이 스캔 체인 선택 레지스터에 저장된다. 임의의 한 상태에서 다른 상태로 천이하는 과정에 TCK 1주기가 소요된다. 즉 기존의 TAP 블록으로 스캔 체인 변경하는 동작을 제어하면 18 사이클 소요된다.

나. PIDM방식 TAP 블록

PIDM 방식은 TAP 제어기 상태가 Test Logic Reset 상태일 때에 JTAG 명령어를 4bit 카운터를 이용하여 IR에 인가하는 방법을 사용한다. 스캔 체인 변경의 자세한 과정은 아래의 그림 11과 같으며 8 사이클에 동작이 수행된다. 이 방식에서는 Test Logic Reset 상태가 4 사이클 유지될 때 스캔 체인 1을 선택하는 SC1(4'b0011) JTAG 명령어가 IR로 로드되어 스캔 체

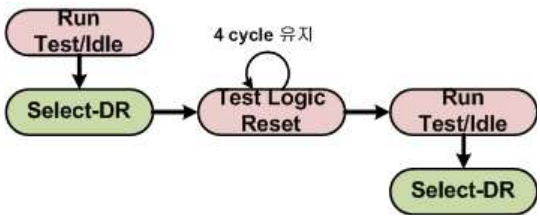


그림 11. 스캔체인 변경의 상태천이도(PIDM TAP블록)
Fig. 11. State transfer diagram of changing scanchain (PIDM TAP block).

인 1이 선택되게 된다.

다. 제안한 TAP 블록

본 논문에서 제안한 TAP 블록으로 OCD를 제어하면 아래의 그림 12에서 보이는 상태 천이에 따라 동작하게 된다. 변형된 JTAG은 스캔 체인을 선택하는 동작을 명령어로 구현하였기에 스캔 체인 1을 선택하는 SCAN1(3'b111) JTAG 명령어를 IR로 인가하면 된다. 즉 이 과정은 5 사이클에 완성된다.

위에서 소개한 방법으로 여러 가지 디버깅 기본 기능 수행에 필요한 구체적인 사이클 비교는 아래의 표 3과 같다. 스캔 체인 바꾸는 데 필요한 사이클은 기존의 TAP은 18 사이클, PIDM 방식은 8 사이클, 제안한 방식은 5 사이클로, 제안한 TAP 블록을 기존의 방식과 비교하면 72.2%, PIDM 방식과 비교하면 37.5% 감소되었다. 이는 IR 제어를 위한 상태를 하나만 사용하기 때문에 보다 많은 사이클 감소가 이루어졌다. 제안한 방식을 이용하였을 경우, IDCODE 읽기는 40 사이클로 기존의 방식 및 PIDM 방식과 비교하면 각각 약 9.1%, 9.1%가 감소되고 RISC 프로세서 내부에 디버깅 명령어를 삽입하는 과정은 40 사이클로 각각 25.9%, 9.1%가 감소된다. OCD가 제공하는 디버깅 기능은 BP (Break-Point)/WP(Watch-Point) 지정 및 감지, 싱글 스텝, 레지스터/메모리 제어 및 디버그 모드 진입, 탈출 등이 있다. 이런 디버깅 기능들은 아래의 표 3에 열거한 기본 기능의 조합으로 구현 가능하다. 예를 들면 레지스터 읽기는 스캔 체인 0을 선택하고 코어에 data 쓰기 기능을 5번 수행하는 것으로 구현이 된다. 그리고 복잡한 디버깅 기능 중 하나인 싱글 스텝은 코어 상태 확인, 전체 레지스터 복원, BP 설정, 디버그 모드 탈출 및 재진입, 코어 상태 재확인 및 레지스터 백업 등 과정으로 구현된다. 이렇게 복잡한 디버깅 기능도 제안한 JTAG으로 구현하면 4367 사이클이 소요되며 기존의 JTAG 및 PIDM 방식보다 더 적은 사이클이 소모되는 것을 확인할 수 있다. 즉 변형된 JTAG은 OCD의 디버

그림 12. 스캔체인 변경의 상태천이도(제안한 TAP블록)
Fig. 12. State transfer diagram of changing scan-chain (proposed TAP block).

표 3. 디버깅 기능 수행에 필요한 사이클
Table 3. Cycle for executing debugging functions.

디버깅 기능	기존 TAP블록	PIDM TAP블록	제안한 TAP블록
스캔 체인 변경	18 cycle	8 cycle	5 cycle 72.2%/37.5%
IDCODE read	44 cycle	44 cycle	40 cycle 9.1%/9.1%
Core status check	50 cycle	30 cycle	23cycle 54%/23.3%
Debug Reg. write	60 cycle	50 cycle	46 cycle 23.3%/8%
Debug Reg. read	64 cycle	54 cycle	49 cycle 23.4%/9.3%
Data write to core	54 cycle	44 cycle	40 cycle 25.9%/9.1%
Data read from core	87 cycle	77 cycle	73 cycle 16.1%/5.2%
Breakpoint setting	446 cycle	426 cycle	408 cycle 8.5%/4.2%
Core Reg. write	303 cycle	269 cycle	248 cycle 18.2%/7.8%
Core Reg. read	307 cycle	273 cycle	251 cycle 18.2%/8.1%
Memory write	544 cycle	488 cycle	453 cycle 16.7%/7.2%
Memory read	781 cycle	691 cycle	635 cycle 18.7%/8.1%
Soft reset halt	1106 cycle	914 cycle	822 cycle 25.7%/10.1%
Resume	2648 cycle	2416 cycle	2281 cycle 13.9%/5.6%
Single step	4915 cycle	4579 cycle	4367 cycle 11.1%/4.6%

깅 기능을 수행함에 있어서 기존의 JTAG과 비교하면 약 8.5~72.2% 성능향상을 보이고 PIDM 방식과 비교하면 약 4.2~37.5% 성능향상을 보인다.

설계된 TAP 블록은 디버깅 기능 수행 사이클 방면에서의 성능향상 뿐만 아니라 추가적으로 기존의 TAP과 비교하였을 때 게이트 카운트도 많이 감소되는 것을 확인할 수 있다. 아래의 표 4는 TSMC 0.13μm CMOS 셀 라이브러리(cell library)로 합성하여 얻은 게이트 카운트 비교 결과이다. 변형된 TAP 블록은 기존과 비교하여 약 31.8% 게이트 카운트가 줄어들었다. 여기서 TAP 블록은 스캔 체인을 제외한 나머지 JTAG 블록을 가리킨다. 감소된 게이트 카운트 200은 디버거를 포함하는 프로세서 사이즈에 비해 극히 미미하지만 이것은 디버깅 수행 사이클이 감소하면서 게이트 카운트가 증

표 4. 게이트 카운트
Table 4. Gate count.

TSMC 0.13μm	기존 TAP 블록	제안한 TAP 블록	비고
Gate count	648	442	31.8%

가된 것이 아니라 부가적으로 게이트가 감소되었기 때문에 좋은 결과이다.

본 논문에서 제안한 TAP 블록은 PIDM 방식과 비교하여서도 사이클 소모가 적을 뿐만 아니라 게이트 카운트도 적은 것을 확인할 수 있으며 이는 구체적인 게이트 카운트로 직관적으로 비교할 수는 없지만 유추할 수는 있다. PIDM 방식은 TAP 제어기 상태가 9개로 FSM 구현에 4bit의 레지스터가 필요하다. 그리고 IR 제어 상태를 모두 제거함으로써 JTAG 명령어 인가를 위한 카운터 등 추가회로가 필요하기에 결과적으로 본 논문에서 구현한 방식보다 게이트 카운트가 더 많아지게 된다.

V. 검 증

설계된 JTAG은 OCD에 적용하여야 검증을 진행할 수가 있다. 그러므로 제안한 변형된 JTAG을 검증하기 위해 본 논문에서는 참고문헌 [1]에서 사용된 Core-A에 내장되는 OCD를 검증한 방법 및 과정을 그대로 사용하였다. 참고문헌 [1]에서 사용한 방법은 아래의 그림 13에서 보이는 바와 같이 세 단계의 검증 과정 즉 행위수준 시뮬레이션 (functional simulation)을 통한 검증, PLI(Program Language Interface)를 이용한 SW 디버거 연동 검증 및 FPGA 레벨에서의 SW 디버거 연동 검증을 진행하였다.

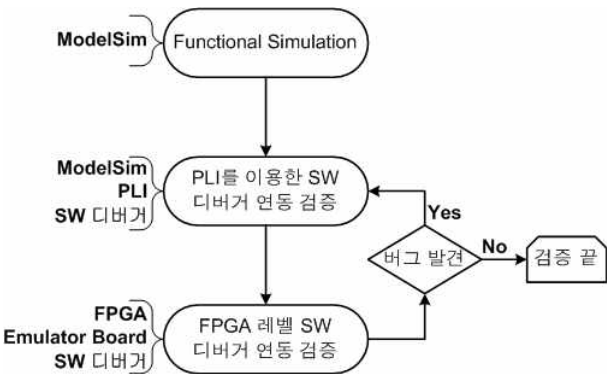


그림 13. OCD 검증 과정
Fig. 13. Procedure of OCD verification.

제안한 JTAG은 Core-A의 OCD에 내장되어 Emulator Board를 통하여 컴퓨터에서 수행되는 SW 디버거와 연동하여 동작하여야 하기에 FPGA 레벨 검증에서 오류가 발생한다 하더라도 원인 분석이 매우 어렵다. 때문에 검증 과정에 PLI를 이용한 SW 디버거 연동 검증을 추가하였다. 검증에 사용된 SW 디버거는 기존의 JTAG을 포함하는 OCD를 제어하는 디버거가 아닌 JTAG 신호 생성 부분이 수정된 SW 디버거이다. JTAG 신호 생성 부분은 변형된 JTAG 제어기의 상태에 따라 새로 구성된 것이다.

1. 행위수준 시뮬레이션 기능 검증

행위수준 시뮬레이션 검증은 프로세서의 단일 명령어, 조합 명령어 및 간단한 응용 알고리즘을 실행시키면서 진행하였다. 이 과정 중 변형된 JTAG을 Core-A의 OCD에 내장하였을 때, 기존의 JTAG과 비교하여 같은 기능을 지원하는지를 확인하였다. 구체적으로 JTAG 블록이 스캔 체인을 제어하는지, OCE에 BP/WP가 지정되는지, 그리고 레지스터 읽기/쓰기, 메모리 읽기/쓰기 등 모든 디버깅 기능을 지원하는지를 검증하였다.

행위수준 시뮬레이션 검증의 하나의 예로 OCE BP 레지스터 셋에 BP를 지정하는 디버깅 기능 검증 과정을 아래의 그림 14에서 보여주고 있다. 먼저 C 언어를 이용하여 변형된 JTAG 제어과정을 모델링하고 텍스트 벡터(Test Vector)를 생성하였다. 텍스트 벡터는 디버깅 기능에 따라 생성되는 TDI, TCK, TMS 3가지 신호

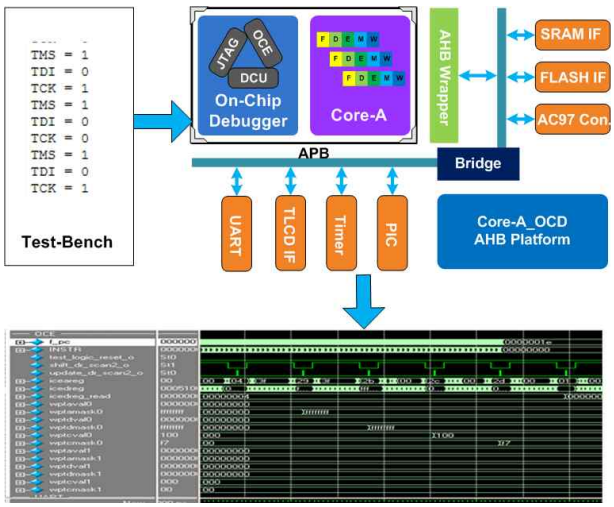


그림 14. 행위수준 시뮬레이션 검증
Fig. 14. Verification by functional simulation.

로 구성되고 이 신호를 테스트 벤치(Test-bench)를 통하여 DUT(Design Under Test)에 인가하면 그림 14에서의 시뮬레이션 파형이 생성된다. 파형을 보면서 원하는 디버깅 기능이 정상적으로 동작하는지 확인하면서 모든 디버깅 기능을 검증하였다.

2. PLI를 이용한 SW 디버거 연동 검증

PLI를 통한 SW 디버거와 ModelSim 시뮬레이터 연동 검증도 진행하였다. PLI를 이용한 검증 환경은 아래의 그림 15와 같다. SW 디버거에서 PLI 연동을 위한 원격 디버깅 RDA를 선택하고 Core-A 테스트 벤치와 소켓 통신을 진행하면서 모델심과 연동 및 통신을 진행한다. 이렇게 연동되면 SW 디버거의 디버깅 명령어가 RDA_PLI를 통하여 테스트 벤치에 전달하게 되고 다시 TDI, TMS, TCK 포트를 통하여 OCD를 내장한 Core-A에 전달된다. OCD는 위 3개 신호를 받아들여 해당 디버깅 기능을 수행하고 Core-A를 제어한다. 그리고 디버깅 기능에 따라 생성된 디버깅 결과는 다시 TDO를 통해 테스트 벤치, PLI를 거쳐 최종적으로 SW 디버거에 전달된다. SW 디버거는 디버깅 결과를 Eclipse GUI를 통하여 사용자에게 보여준다. 만약 SW 디버거에서 보여준 디버깅 결과가 사용자가 원하는 결과와 일치하지 않으면 시뮬레이터 파형을 통하여 원인 분석할 수 있다.

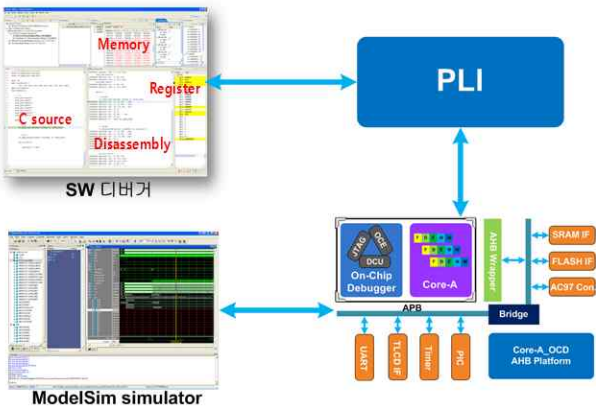


그림 15. PLI를 이용한 검증 환경
Fig. 15. Verification environment by PLI.

3. FPGA레벨 SW 디버거 연동 검증

행위수준 시뮬레이션 검증과 PLI를 이용한 검증은 모두 gate delay를 고려하지 않고 기능만 검증하게 된다. 그리고 실시간 검증이 아니기 때문에 추가적으로

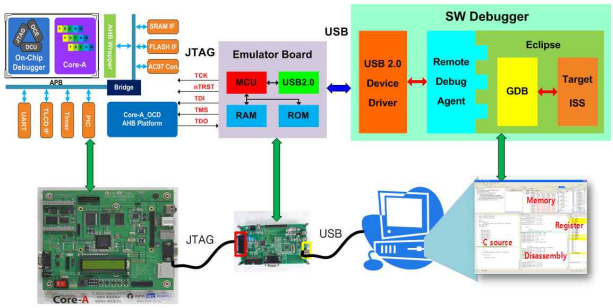


그림 16. 디버깅 검증 환경
Fig. 16. Verification environment for debugging.

FPGA 레벨에서의 검증이 필요하다.

설계된 JTAG을 아래 그림 16의 타깃 시스템과 같이 Core-A SoC에 내장되는 OCD에 적용하였다. 그리고 Core-A SoC를 Xilinx FPGA에 다운로드(download)하여 그림 16과 같은 디버그 시스템을 구축하여 FPGA 레벨에서의 SW 디버거 연동 검증을 진행하였다.

타깃 시스템은 설계된 OCD를 내장한 Core-A SoC 플랫폼이다. 타깃 시스템과 Emulator Board를 JTAG cable로 연결하고 Emulator Board와 호스트 컴퓨터는 USB cable로 연결한다. 그리고 컴퓨터에는 GDB[9] 기반의 Core-A SW 디버거를 실행시키면 전반적인 디버그 시스템이 구축된다.

FPGA 레벨에서의 SW 디버거 연동 검증에서 OCD에 내장된 JTAG의 오동작을 발견하면 PLI 연동 검증에서 최대한 FPGA와 비슷한 환경을 조성하여 원인을 찾고 수정하여 다시 FPGA 검증을 진행하였다. FPGA 검증에서는 ADPCM, MP3 decoder 등과 같은 복잡한 알고리즘을 실시간으로 실행시키면서 SW 디버거가 지원하는 디버깅 기능을 모두 검증하였다. 이렇게 FPGA 검증과 PLI 검증을 반복 수행하면서 변형된 JTAG이 RISC 프로세서의 디버깅 기능을 수행하는 것을 검증하고 신뢰성을 확인하였다.

Ⅵ. 결 론

기존의 TAP 블록은 RISC 프로세서의 디버거를 제어함에 있어서 많은 불필요한 사이클이 존재한다. 이는 DR 제어 및 IR 제어할 때 불필요한 TAP 상태 천이가 있기 때문이다. 본 논문에서는 이런 불필요한 상태를 제거함으로써 RISC 프로세서의 디버거에 적합한 변형된 JTAG을 설계하고 검증하였다.

제안한 JTAG을 구현하고 32bit RISC 프로세서인

Core-A에 적용하여 검증 작업을 진행하였다. 변형된 JTAG을 이용하여 RISC 프로세서의 디버거를 제어하는 기능은 기존의 JTAG과 비교하였을 경우 100% 호환된다. 즉 변형된 JTAG은 기존의 JTAG과 같은 기능을 지원하는 전제하에서 디버깅 기능 수행에 필요한 사이클 수가 8.5~72.2% 감소되는 장점을 갖고 있으며 추가적으로 게이트 카운트도 31.8% 감소된다. 그리고 PIDM 방식과 비교하여도 약 4.2~37.5% 성능향상을 보인다.

제안한 JTAG은 ARM 시리즈 프로세서, OpenRISC 등과 같이 이미 검증되고 널리 사용되고 있는, JTAG을 기반으로 하는 RISC 프로세서의 디버거에 모두 적용 가능하지만 기존에 설계된 HW, SW 파트를 수정해야 해야 하고 또다시 검증해야 하는 단점이 있기 때문에 이런 프로세서에 적용하는 것은 바람직하지 않다. 본 논문에서 제안한 JTAG을 적용할 타깃은 JTAG을 기반으로, 새롭게 설계되는 RISC 프로세서들이다. 디버거가 내장되는 프로세서를 설계함에 있어서 제안한 JTAG을 적용하려면 HW, SW 2가지 측면으로 볼 수가 있다. HW 측면에서는 본 논문에서 기술한 방법대로 제안한 JTAG을 설계하면 기존의 JTAG 설계와 비교할 시 추가 어려움이 없다. SW디버거는 프로세서마다 서로 다르기 때문에 새로운 프로세서를 위하여 SW는 새로 설계되어야한다. 즉 SW 측면에서 제안한 JTAG 제어를 위한 파트는 TAP 상태 천이 제어와 추가된 JTAG 명령어만 구현하면 되기에 별로 어려움이 없을 것이다. 그리고 제안한 JTAG을 새로 설계되는 프로세서에 적용한 결과는 기존의 JTAG 표준 혹은 PIDM 방식으로 설계할 때보다 더 좋은 성능향상 결과를 보일 것을 확신한다.

본 논문에서는 제안한 JTAG을 설계하여 검증 타깃을 Core-A로 선택하였다. Core-A의 OCD에 제안한 JTAG을 내장하고 Core-A SoC 플랫폼을 구현하여 GDB 기반의 Core-A SW 디버거와 연동하여 FPGA 레벨까지의 3단계 검증을 진행하였다. 변형된 JTAG은 RISC 프로세서의 디버거를 제어함에 있어서의 충분한 기능 및 신뢰성을 확인하였다. 제안한 JTAG을 새로 설계되는 RISC 프로세서의 디버거에 적용하면 기존의 JTAG보다 디버깅 기능 수행 사이클을 많이 감소할 뿐만 아니라 게이트 카운트도 감소하게 된다.

참 고 문 헌

[1] 허경철, 박형배, 정승표, 박주성, “Core-A를 위한 효율적인 On-Chip Debugger 설계 및 검증”, 전자공학회논문지, 제47권 SD편, 제4호, 322-333쪽, 2010년 4월.

[2] G.R. Alves and J.M. Martins Ferreira, “From Design-for-Test to Design-for-Debug-and-Test: Analysis of Requirements and Limitations for 1149.1,” Proc. 17th IEEE VLSI Test Symp. (VTS99), IEEE CS Press, Los Alamitos, California., pp.473-480, 1999.

[3] IEEE Std. 1149.1a-1993, “Test Access Port and Boundary-Scan Architecture”, IEEE, Piscataway, N.J., 1993.

[4] www.core-a.net

[5] 박형배, 지정훈, 허경철, 우균, 박주성, “GNU 디버거를 이용한 온칩 디버깅 시스템 설계”, 전자공학회논문지, 제46권 SD편, 제1호, 24-38쪽, 2009년 1월.

[6] Yue-li Hu, Ke-xin Zhang, “Design of On-Chip Debug Module based on MCU”, High Density packaging and Microsystem Integration, 2007. pp. 1-4, June 2007.

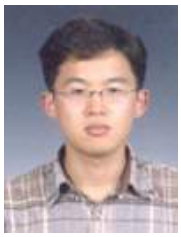
[7] 윤연상, 김승열, 권순열, 박진섭, 김용대, 유영갑, “JTAG 기반 테스트의 성능향상을 위한 PIDM (Preceding Instruction Decoding Module)”, 전자공학회논문지, 제41권 SD편, 제8호, 697-704쪽, 2004년 8월..

[8] 윤연상, 류광현, 김용대, 한선경, 유영갑, “JTAG 기반 SoC의 개선된 온 칩 디버깅 유닛 설계”, 한국통신학회논문지, 제30권, 제3A호, 226-232쪽, 2005년 3월.

[9] Richard Stallman, Roland Pesch, Stan Shebs, “GDB User Manual: Debugging With GDB(The GNU Source-Level Debugger)”, GDB version 6.4. Technical report, Free Software Foundation, Cambridge, MA.

저 자 소 개

허 경 철(학생회원)
2005년 중국 연변과학기술대학 전자통신학과 학사 졸업.
2008년 부산대학교 전자공학과 석사 졸업.
2011년 현재 부산대학교 전자전기공학과 박사과정.
<주관심분야 : 프로세서 설계, 디버거 설계, SoC 설계, 멀티 프로세서 플랫폼 설계, 오디오 알고리즘 구현>



정 승 표(학생회원)
2007년 부산대학교 전자공학과 학사 졸업.
2009년 부산대학교 전자공학과 석사 졸업.
2011년 부산대학교 전자공학과 박사 과정.
<주관심분야 : 신호처리, SoC 플랫폼 설계>

박 형 배(정회원)
2004년 동서대학교 전자공학과 학사 졸업.
2006년 부산대학교 전자공학과 석사 졸업.
2011년 현재 부산대학교 전자전기공학과 박사과정.
<주관심분야 : 프로세서 설계, 디버거 설계, 디지털 시스템 설계, SoC 설계>

박 주 성(평생회원)-교신저자
1976년 부산대학교 전자공학과 학사 졸업.
1978년 한국과학기술원(KAIST) 전자공학과 석사 졸업.
1989년 University of Florida 전자공학과 박사 졸업.
1998년～2007년 부산대학교 IDEC 센터장
1991년～현재 부산대학교 전자공학과 교수
2008년～2009년 부산대학교 공과대학 학장
<주관심분야 : DSP 설계, ASIC 설계, 반도체 소자 모델링, 음성/사운드 신호처리 및 구현, SoC 설계>