멀티미디어 SoC 플랫폼의 효율적인 통신을 위한 크로스바 스위치 온칩 버스 설계

허정범*, 임미선*, 류광기*
*한밭대학교 정보통신전문대학원 정보통신공학과
e-mail:skyhjb@nate.com

A Crossbar Switch On-chip Bus Design for Efficient Communication of a Multimedia SoC Platform

Jungbum Heo*, Misun Lim* and Kwangki Ryoo*
*Graduate School of Information and Communication,
Hanbat National University

요 약

최근 EDA 툴의 기술적인 향상과 반도체 공정의 발달로 IC 설계자들은 RISC 프로세서, DSP 프로세서, 메모리 등 많은 IP가 하나로 집적되는 SoC구조가 가능해졌다. 하지만 기존에 사용되는 대부분의 SoC는 공유버스 구조를 가지고 있어, 병목현상이 발생하는 문제점을 가진다. 이러한 문제점은 SoC 내부의 IP들이 많을수록 SoC 플랫폼의 전체 성능이 저하되어, CPU 자체의 속도보다는 효율적인 통신에 의해 성능이 좌우된다. 본 논문에서는 공유버스의 단점인 병목현상을 줄이고 성능을 향상시키기 위하여 크로스바 스위치버스 구조를 제안한다. OpenRISC 프로세서, VGA/LCD 제어기, AC97 제어기, 디버그 인터페이스, 메모리 인터페이스로 구성되는 SoC 플랫폼의 WISHBONE 온칩 공유버스 구조와 크로스바 스위치 버스 구조의 성능을 비교한 결과, 기존의 공유버스보다 26.58%의 성능이 향상됨을 확인하였다.

1. 서론

SoC(System-on-a-Chip)기술은 휴대폰을 비롯한 PDA, 디지털 카메라, DVD 플레이어, 디지털 TV 등 다양한 기능을 갖는 제품들로 상용화되고 있으며, 해를 거듭할수록 칩의 집적도가 기하학적으로 높아지고, SoC에 요구되는 기능들이 증가하여 SoC 안에는 많은 수의 IP를 내장하게 되었다. 이에 따라 SoC 설계 복잡도는 증가하여, SoC 설계 시 더 많은 시간과 노력이 요구되었다. 따라서 설계자는 짧은 TTM(Time-To-Market)에 설계요구를 만족시키기 위해 IP를 재사용하는 플랫폼을 이용한 SoC 설계 기법을 널리 사용하고 있다.

기존에 사용되는 SoC 플랫폼은 대부분 공유버스 구조 기반으로 설계되어있다. 공유버스 기반의 SoC 플랫폼은 내부의 많은 IP들이 하나의 버스를 공유하여 통신하기 때문에, 하나의 통신이 종료할 때까지 대기해야하는 문제점이 발생한다. 이러한 문제점은 통신을 하고자 하는 IP의 개수가 많을수록 병목현상이 증가하여 전체 시스템의 성능을 저하시키게 되어, CPU 자체의 속도보다는 효율적인 통신이 그 성능을 좌우하게 된다. 본 논문에서는 공유버스 구조의 문제점인 병목현상을 해결하고, 효율적인 통신을 통해 SoC플랫폼의 성능을 향상시키기 위하여 크로스바 스위치 온칩 버스 구조를 제안한다.

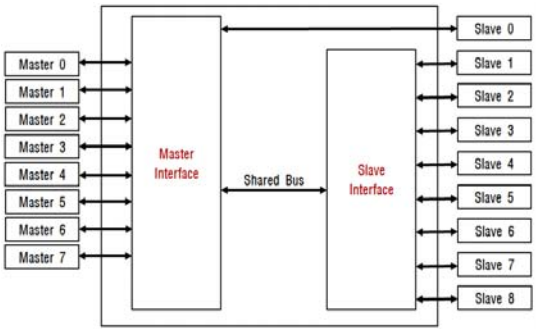
본 논문의 구성은 다음과 같다. 2장에서는 기존의 공유버스 구조 및 문제점을 설명하고, 3장에서는 제안 된 크로스바 스위치 온칩 버스의 특징 및 구조를 설명하며, 4장에서는 SoC플랫폼을 이용한 통합 설계

및 검증 결과를 설명한다. 마지막으로 5장에서는 본 연구의 결론을 도출한다.

2. 기존의 공유버스 구조

기존의 공유 버스 구조는 최대 8개의 마스터와 9개의 슬레이브를 연결할 수 있으며, 크게 마스터 인터페이스, 슬레이브 인터페이스로 구성된다.

그림 1은 공유버스의 전체적인 블록도이다. 마스터 인터페이스는 8개의 마스터 중 하나의 마스터를 선택하여 공유버스를 통해 슬레이브 인터페이스와 통신하도록 제어한다. 마스터 인터페이스는 마스터 모듈들로부터 버스를 사용하겠다는 요청신호를 받으면, 우선순위 제어 로직에 의해 요청신호를 보낸 마스터들 중 가장 높은 우선순위를 갖는 마스터가 버스의 소유권을 점유한다. 우선순위는 마스터 0, 1, 2, 3, 4, 5, 6, 7 순서로 고정된다. 슬레이브 인터페이스는 공유버스를 통해 전달받은 주소를 디코딩하여 8개의 슬레이브 중 하나의 슬레이브를 선택하여 공유버스와 통신하도록 제어한다.



[그림 1] 공유버스 전체 블록도

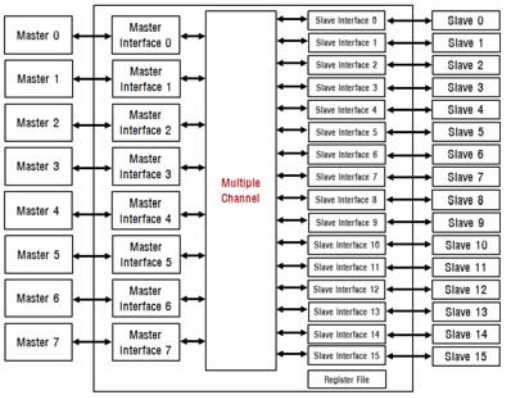
이 공유버스는 단순한 구조와 프로토콜로 구성되지만, 동등한 우선순위레벨을 갖는 마스터들이 존재하지 않는다. 따라서 우선순위가 높은 하나의 마스터가 버스를 독점하는 문제점이 있다. 또 다른 문제점은 하나의 공유버스를 통하여 다수의 마스터와 슬레이브가 통신한다는 점이다. 하나의 마스터와 슬레이브가 버스를 점유하게 되면, 다른 마스터가 이미 버스를 점유한 마스터와 슬레이브 간의 통신이 끝날 때까지 기다려야하는 병목현상이 일어난다.

3. 다중 채널 버스구조

다중 채널 버스는 기존의 공유버스 구조의 문제점인 병목현상을 해결하고, 균형 있고 효율적인 통신을 지원한다.

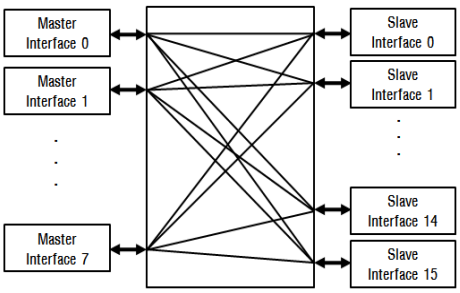
그림 2는 다중 채널 버스의 내부구조로 최대 8개의 마스터와 16개의 슬레이브를 연결할 수 있어. 기존의 버스 구조보다 뛰어난 확장성을 가지며, 마스터 인터페이스, 슬레이브 인터페이스, 레지스터 파일로 구성된다.

마스터 인터페이스는 마스터 모듈로부터 통신 요청신호를 받으면, 주소를 디코딩하여 16개의 통신채널 중 하나를 선택하여, 통신하고자 하는 슬레이브 인터페이스에 요청신호를 보낸다. 슬레이브 인터페이스는 슬레이브 모듈과 통신하기를 원하는 각 마스터 모듈들의 우선순위를 검사하여 우선순위가 가장 높은 마스터 모듈과 통신을 연결해준다. 레지스터 파일은 16개의 16비트 레지스터를 가지며, 각 레지스터는 슬레이브 인터페이스와 매핑되어 마스터 모듈 8개의 우선순위 정보를 제공한다.



[그림 2] 다중채널 버스 전체 블록도

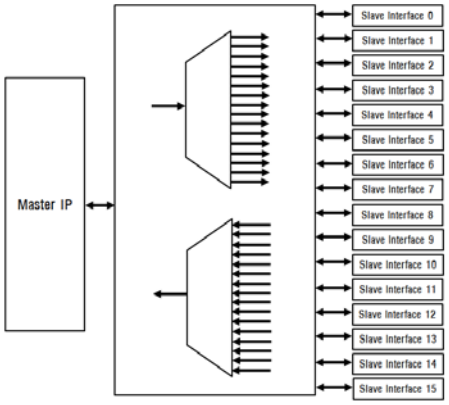
그림 3은 다중 채널 버스의 내부 구조이다. 다중 채널 버스는 WISHBONE 스펙의 크로스바 스위치 연결 구조이다. 각 마스터는 16개의 슬레이브에 접근이 가능한 16개의 통신채널을 확보하여, 다른 마스터가 사용하고 있지 않은 슬레이브와 통신이 가능하도록 하여 병렬통신을 지원한다. 따라서 공유버스 구조의 각 마스터들이 하나의 통신채널을 공유하여 사용함으로써 발생되는 병목현상의 문제점을 해결한다.



[그림 3] 다중채널 내부 구조

3.1 마스터 인터페이스

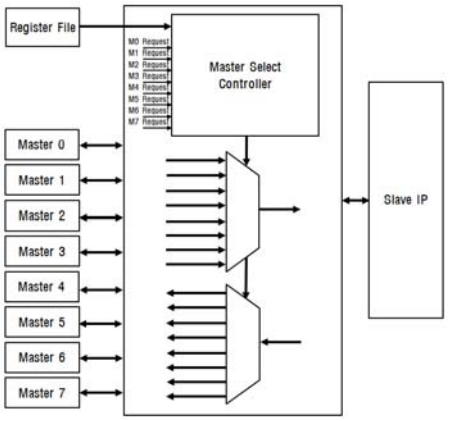
그림 4는 마스터 인터페이스의 내부 구조이다. 마스터 인터페이스는 마스터로부터 통신 요청 신호를 받으면, 주소를 판별해 16개의 연결된 통신채널중 하나를 선택하여 마스터가 원하는 슬레이브 인터페이스에 통신 요청 신호를 보내는 역할을 하며, 연결이 이루어진 이후에는 연결된 마스터와 슬레이브간에 통신이 이루어지도록 지원한다.



[그림 4] 마스터 인터페이스 블록도

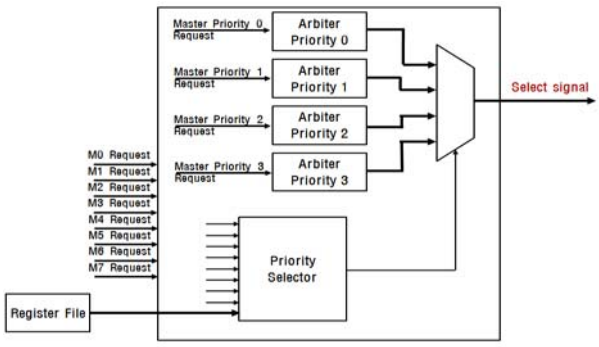
3.2 슬레이브 인터페이스

그림 5는 슬레이브 인터페이스를 나타낸다. 슬레이브 인터페이스는 마스터선택제어기에 의해 통신 요청이 들어온 마스터 중 하나를 선택하여 슬레이브와 통신이 이루어지도록 지원한다.



[그림 5] 슬레이브 인터페이스 블록도

그림 6은 마스터선택제어기의 구조로 4개의 아비터와 우선순위선택기로 구성된다. 우선순위선택기는 슬레이브 인터페이스에서 마스터들의 요청신호를 받으면 레지스터 파일로부터 각 마스터들의 우선순위 정보를 제공받아 요청신호를 발생한 마스터들 중에 가장 높은 우선순위를 갖는 마스터부터 순서대로 슬레이브와 통신이 가능하도록 연결한다. 각각의 아비터는 라운드-로빈 형식으로 동작하여 하나의 마스터가 버스를 오래 독점하는 것을 방지한다.



[그림 6] 마스터선택제어기의 블록도

표 1은 마스터 0번의 우선순위 값을 3, 마스터 1, 2 , 3, 4번의 우선순위 값을 1로 설정하고, 슬레이브 인터페이스 0번에 5개의 마스터가 계속해서 요청신호를 보냈을 때, 슬레이브 0번과 통신하는 순서를 나타낸다. '↓'는 통신이 이루어진 것을 표시한 기호이다.

공유 버스의 경우 두 개의 마스터가 계속해서 버스를 점유하는 반면, 다중 채널 버스는 우선순위가 높은 마스터 0번을 제외한 나머지 마스터들이 균일하게 슬레이브 0번과 통신한다. 즉, 기존의 공유버스는 우선순위가 8개로 나뉘어 우선순위가 높은 마스터가 계속해서 버스를 점유하는 반면, 다중 채널 버스는 우선순위 값을 4단계로 나누고 같은 우선순위를 갖은 마스터는 라운드-로빈 방식으로 버스 사용권한을 주어, 하나의 마스터가 버스를 슬레이브를 독점하는 것을 방지하여 효율적인 통신이 이루어지도록 지원한다.

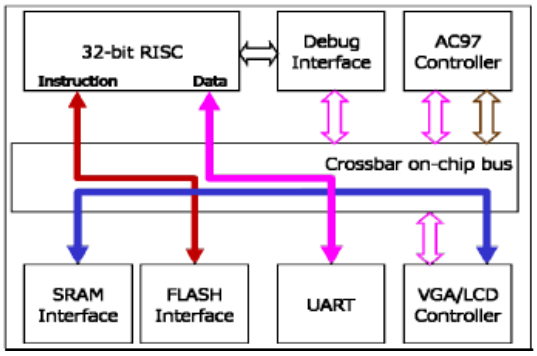
[표 1] 5개의 마스터가 슬레이브 0번에 계속해서 접근할 경우의 처리순서
(마스터0의 우선순위: 3, 마스터 1, 2 ,3, 4의 우선순위: 1)

Shard bus					Multichannel bus				
m0	m1	m2	m3	m4	m0	m1	m2	m3	m4
↓					↓				
	↓					↓			
↓					↓				
	↓						↓		
↓					↓				
	↓							↓	
↓					↓				
	↓								↓

4. 실험결과

크로스바 스위치 버스의 검증을 위해 32비트 OpenRISC 프로세서, 디버그 인터페이스, VGA/LCD

제어기, AC97 제어기, UART, 메모리 인터페이스로 구성되어 있는 WISHBONE 프로토콜기반의 멀티미디어 SoC 플랫폼을 사용한다.



[그림 7] 병렬통신이 이루어지고 있는 멀티미디어 SoC 플랫폼의 블록도

그림 7은 이미지-디스플레이 테스트 프로그램이 멀티미디어 SoC 플랫폼에서 병렬통신을 하는 모습이다. 프로세서가 플래시 메모리로부터 명령을 읽어가는 것과 동시에 UART에 출력할 메시지 데이터를 전송하고, VGA 컨트롤러는 SRAM으로 부터 이미지 데이터를 읽어간다.

표 2는 이미지-디스플레이 테스트 프로그램을 이용하여 크로스바 온칩 버스를 내장한 SoC 플랫폼의 성능을 확인한 결과이며, 실험 결과 기존의 공유버스에 비해 성능이 26.58%가 향상됨을 검증하였다. 크로스바 온칩 버스를 내장한 SoC 플랫폼은 Xilinx XC4VLX8FPGA 디바이스에 구현한 결과 최대 48.66MHz 주파수에서 동작한다.

[표 2] 테스트 프로그램을 적용한 시뮬레이션결과

	공유버스 (ns)	크로스바 버스 (ns)	차이 (ns)	효율 (%)
테스트 프로그램	19,556	14,357	5,199	26.58

5. 결론

본 논문에서는 대부분의 SoC에 사용되는 공유버스의 단점인 병목현상을 줄이고 성능을 향상시키기 위하여 크로스바 스위치버스 구조를 제안한다. 크로스바 스위치버스 구조는 WISHBONE 프로토콜을 사용하며, 최대 8개의 마스터와 16개의 슬레이브의 인터페이스를 지원한다. 다중 채널을 보유함으로써 병렬통신을 지원하여 공유버스의 문제점인 병목현상을 해결한다. 제안된 크로스바 스위치 버스를 검증하기 위해 멀티미디어 SoC 플랫폼을 사용하였다. 제안된 크로스바 스위치 버스를 사용한 SoC 플랫폼의 실험결과에 따르면 공유버스 구조를 가진 SoC 플랫폼보다 26.58%의 성능이 향상됨을 확인하였다.

참고문헌

[1] Zhihui Xiong, Sikun Li, Jihua Chen and Dawei Wang, "A Platform-based SoC Hardware/ Software Co-Design Environment", The 8th International Conference on Computer Supported Cooperative Work in Design, vol.2, pp 443-448, May 2004 .

[2] Sanghun Lee, Chanho Lee, "A High Performance SoC On-chip-bus with Multiple Channels and Routing Processes", 2006 IFIP International Conference on Very Large Scale Integration, pp 86-91, Oct. 2006.

[3] Jacob Gorban, UART IP Core Specification, Rev. 0.6 August 11, 2002 .

[4] Damjan Lampret, OpenRISC1200 IP Core Specification Rev. 0.7, September 6, 2001.

[5] Rudolf Usselmann, AC97 Controller IP Core Specification Revision 1.2, September 19, 2002.

[6] Richard Herveille, VGA/LCD core specification, Rev 2.0, March 20, 2003.

[7] Igor Mohor, SoC Debug Interface, Rev. 3.0 April 14, 2004.