논문 2008-45CI-5-1

다양한 버스 중재방식에 따른 플라잉 마스터 버스아키텍처의 TLM 성능분석

(Performance Analysis of TLM in Flying Master Bus Architecture Due To Various Bus Arbitration Policies)

이 국 표*, 윤 영 섭*

(Kook-Pyo Lee and Yung-Sup Yoon)

요 약

일반적인 버스 아키텍처는 공용버스 내에 마스터와 슬레이브, 아비터, 디코더 등으로 구성되어 있다. 특히 여러 마스터들이 동시에 버스사용 권리를 받을 수 없으므로, 아비터가 공용버스와 마스터 사이에서 중재하는 역할을 수행한다. 중재 방식에는 fixed priority 방식, round-robin 방식, TDMA 방식, Lottery 방식 등이 연구되고 있는데, 중재방식에 따라 버스 사용의 효율성이 결정된다. 반면 버스 아키텍처를 수정하여 시스템의 성능을 극대화할 수 있는데, 본 논문에서는 병렬 데이터 통신을 지원하는 플라잉 마스터 버스 아키텍처를 제안하였고, 위에서 언급한 여러 가지 버스 중재 방식에 대하여 일반적인 공용버스와 비교하여 장단점을 분석하였다. TLM(Transaction Level Model)을 이용한 성능검증 결과로부터 버스 중재방식과 무관하게 약 40%의 성능이 향상되었음을 확인하였다. 플라잉 마스터 버스 아키텍처가 좀 더 연구되고 다양한 SoC에 적용되면서 고성능 버스 아키텍처로 자리매김할 것이다.

Abstract

The general bus architecture consists of masters, slaves, arbiter, decoder and so on in shared bus. Specially, as several masters do not concurrently receive the right of bus usage, the arbiter plays an important role in arbitrating between shared bus and masters. Fixed priority, round-robin, TDMA and Lottery methods are developed in general arbitration policies, which lead the efficiency of bus usage in shared bus. On the other hand, the bus architecture can be modified to maximize the system performance. In the paper, we propose the flying master bus architecture that supports the parallel bus communication and analyze its merits and demerits following various arbitration policies that are mentioned above, compared with normal shared bus. From the results of performance verification using TLM(Transaction Level Model), we find that more than 40% of the data communication performance improves, regardless of arbitration policies. As the flying master bus architecture advances its studies and applies various SoCs, it becomes the leading candidate of the high performance bus architecture.

Keywords: bus architecture, performance improvement, flying master, arbitration policy

I. 서 론

현재 우리는 고도로 발달된 문명의 혜택 속에서 살고 있다. 언제 어디에서나 통화를 할 수 있으며, 세계 사람들하고 컴퓨터를 통해 대화를 할 수 있고, 또한 위성을 통해 유럽에서 하는 축구 경기를 시청할 수도 있다. 이 문명의 혜택을 이끄는 결정적인 분야가 바로 전자 통신 분야이다. 전자 통신 분야 중 SoC는 이런 문명의 혜택을 더욱 더 진보시키는데 있어서 결정적인 역할을 수행하고 있다. SoC는 System on a Chip의 약자로서 하나의 칩 안에 모든 시스템을 집적화 시킨 것을 의미한다. SoC 중 ARM 프로세서는 전 세계적으로 70%이상의 시장 점유율을 보이고 있다[1]. 결국 ARM 사에서 제안한 버스 구조인 AMBA(Advanced Microcontroller Bus Architecture) 시스템이 온 칩 통신의 표준이 되고 있다. AMBA 시스템에는 AHB(Advanced High performance Bus)와 ASB(Advanced System Bus) 그

* 정회원, 인하대학교 전자공학과
(Dept. of Electronics Engineering, Inha University)
접수일자: 2008년8월20일, 수정완료일: 2008년9월5일

리고 APB(Advanced Peripheral Bus)가 있는데, 일반적으로 고성능 버스라 함은 AHB를 의미한다. 그렇기 때문에 많은 회사에서 시스템 성능을 높이기 위해 AHB에 대해 많은 연구를 진행하고 있다. 일반적으로 AHB는 하나의 버스에 4가지의 구성요소로 이루어져 있다. 그 구성요소는 마스터, 슬레이브, 아비터, 디코더이며 마스터는 CPU, DMA, DSP 등과 같이 어드레스나 제어 신호를 내보냄으로 "read"나 "write"의 동작을 할 수 있도록 하는 주체를 말하며 슬레이브는 DRAM, SRAM 컨트롤러 등과 같이 주어진 어드레스 공간 내에서 "read"나 "write"를 가능하게 해주는 장치를 말한다. 그리고 아비터는 여러 개의 마스터가 동시간대에 버스를 사용할 수 없기 때문에 이를 중재하는 역할을 수행하며, 중재하는 방식에 따라 시스템의 성능을 향상 시킬 수 있어 별개의 IP의 성능 향상과는 별도로 많은 연구를 하고 있는 부분이다. 마지막으로 디코더는 마스터로부터 나오는 어드레스의 상위 비트를 가지고 적절한 슬레이브를 선택해주는 역할을 수행한다. 아비터의 일반적인 버스 중재 방식에는 fixed priority 방식과 round-robin 방식이 있다. Fixed priority 방식은 각 마스터들이 우선순위를 가지고 있어서 우선순위가 높은 마스터의 데이터 처리량이 많기 때문에 우선순위가 낮은 마스터의 스타베이션(Starvation)이 발생할 수 있다. Round-robin 방식의 경우에는 마스터의 우선순위가 정해지지 않고 골고루 버스 점유권을 주기 때문에 마스터들이 공평하게 버스를 이용할 수 있다. 그러나 중요한 마스터의 데이터 처리를 할 경우에 이를 빠르게 처리할 수 없는 단점을 가지고 있다[2].

일반적인 버스 아키텍처의 경우는 하나의 버스에 여러 개의 마스터, 슬레이브와 아비터, 디코더로 구성되어 있는데, 하나의 버스를 동시간대에 이용할 수 없기 때문에 이를 해결하기 위해 최근에는 버스를 다중으로 두고 그 사이에 브리지를 연결한 다중 버스아키텍처가 구성되어 있다[3]. 그러나 이 아키텍처의 경우는 버스 1에 있는 마스터가 버스 2에 있는 슬레이브를 이용하고자 할 경우 브리지를 통해서 데이터를 처리해야하기 때문에 브리지의 레이턴시에 의한 지연이 발생할 수 있다.

본 논문에서는 일반적인 버스 아키텍처를 보완하여 중요한 마스터의 데이터 처리를 버스를 이용하지 않고 직접적으로 처리하고 다른 마스터들 또한 동시간대에 버스를 이용하여 데이터를 처리할 수 있는 플라잉 마스터 버스 아키텍처를 제안하고, TLM(Transaction Level Model) 방법[4]으로 성능을 분석하고 검증하였다.

Ⅱ. 본 론

1. 플라잉 마스터 버스아키텍처

그림 1(a)는 일반적인 버스 아키텍처를 보여주고 있다. 하나의 공용 버스 내에 마스터 M1과 M2 그리고 아비터 AB, 슬레이브 S1과 S2가 있다. 만약 마스터 M1이 슬레이브 S1에 데이터를 전송하고, 마스터 M2가 슬레이브 S2에 데이터를 동시에 전송하고자 할 경우, 공용 버스에서 동시에 전송을 수행할 수 없으므로 아비터 AB에 의해서 버스 시스템의 중재가 이루어진다. 그림 1(a)에서는 마스터 M1이 전송되고 마스터 M2는 전송을 하지 못하고 있다. 마스터 M1의 데이터 전송이 끝난 후에야 마스터 M2의 데이터 전송을 수행할 수 있다.

그림 1(b)는 본 논문에서 제안하는 플라잉 마스터 버스 아키텍처를 보여주고 있다. 플라잉 마스터 FM에서 슬레이브 S1으로 데이터를 전송할 경우, 버스를 거치지 않고 직접적으로 전송이 되고 또한 동시에 마스터 M1과 슬레이브 S2간에 데이터 전송이 이루어짐을 보이고 있다. 이 경우 그림 1(a)와 비교하여도 데이터를 병렬처리할 수 있기 때문에 시스템의 성능을 높일 수 있는 효율적인 버스 아키텍처라 할 수 있다. 그러나 플라잉 마스터 FM이 슬레이브로 직접 신호를 전달해야 하므로 별도의 플라잉 마스터 래퍼(wrapper)가 필요하고 또한 슬레이브는 플라잉 마스터와 버스로부터 신호를 받게 됨으로 신호를 선택하는 블록인 슬레이브 래퍼를 추가해야 한다. 이럴 경우 발생할 수 있는 디자인 오버헤드나 타이밍 마진은 이전에 발표한 논문[6]에서 증명하

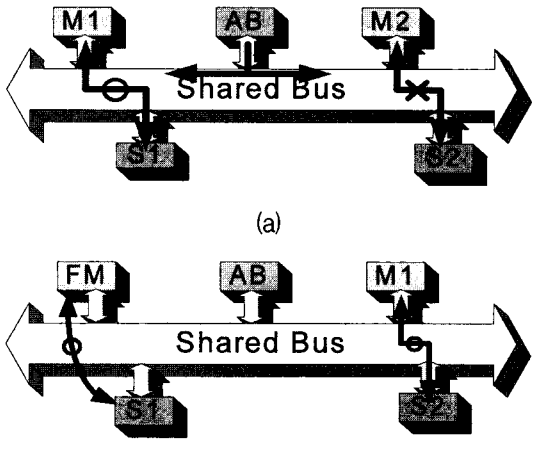


그림 1. (a) 기존의 버스 아키텍처와 (b) 제안한 버스 아키텍처
Fig. 1. (a) Conventional and (b) proposed bus architecture.

였다. 또한 일반적인 버스 아키텍처의 경우 프로세서의 비중이 50% 이상이므로 fixed priority에서 1순위 마스터를 프로세서로 정하면 성능은 좋아지지만, 다른 마스터의 스타베이션이 발생할 수 있는데 비해, 제안한 버스 아키텍처의 경우, 플라잉 마스터를 프로세서로 정하여 버스아키텍처를 구성하면 프로세서의 버스점유율을 높여 결국 성능을 향상시킬 수 있다.

2. 버스아키텍처 모델구성

그림 2(a)는 일반 싱글버스 아키텍처 모델의 상태도이며[5], 그림2(b)는 플라잉 마스터 버스 아키텍처 모델의 상태도이다. 플라잉 마스터의 데이터 트랜잭션은 그림 2(b)의 상태도처럼 동작하게 되며, HoldSt 지연상태가 추가된다.

그림 2(b)의 플라잉 마스터 모델을 살펴보면 다음과 같다. InitSt 상태는 모든 마스터로부터 데이터전송 요청이 발생하지 않아서 기다리는 구간이다. 마스터의 데이터 요청이 발생하면 TransferGenSt 상태로 진입하게 된다. TransferGenSt 상태는 데이터전송에 필요한 어드레스, 데이터, 전송신호 등을 만드는 구간이며, 사이클 cur_cycle이 증가 없이 ArbitrationSt 상태로 진행하게 된다. ArbitrationSt 상태는 여러 마스터의 데이터전송 요청 중에서 우선순위에 따라 마스터를 선택하는 구간이다. Arbitration() 함수에 의해서 해당 마스터를 선택하게 되는데 ArbitrationSt 상태에서 선택된 슬레이브가 다른 마스터에 의해 접근되고 있으면, HoldSt 상태로 진입하게 되고, 선택된 슬레이브에 접근이 없으면 TransferExecSt 상태로 진입한다. HoldSt 상태에서 다른 마스터들이 선택된 슬레이브에 접근하지 않을 때까지 기다린 후, TransferExecSt 상태로 진입하게 된다. TransferExecSt 상태는 데이터 전송이 실제로 이루어지는 구간으로서, 슬레이브 레이턴시가 고려되어 데이터 트랜잭션이 발생한다.

각 마스터별로 전송하고자 하는 데이터는 길이에 따라 싱글 데이터와 버스트 데이터로 나뉘며, 버스트 데이터는 길이에 따라 4, 8, 16 등을 가질 수 있다. 본 연구에서는 랜덤함수를 이용하여 데이터 길이를 무작위로 발생시켜 전송시켰다.

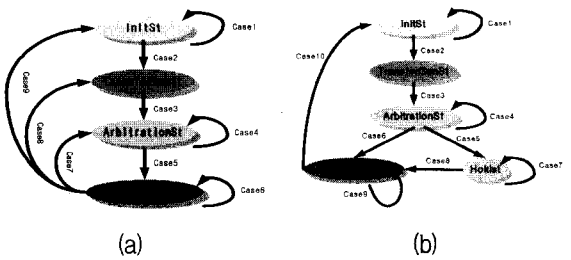


그림 2. 버스 아키텍처 모델의 상태도
Fig. 2. State machine of bus architecture model.

표 1. 플라잉 마스터 버스아키텍처의 알고리즘
Table 1. Algorithm of flying master bus architecture.

```
1:  flying_bus_model()
2:  begin
3:    Cur_Cycle=0
4:    State=InitSt
5:    Master_Signal[flying master]=Idle
6:    Slave_Signal[all slaves]=Idle
7:      if (State=InitSt) then
8:        Idle_Gen(Master_Signal[flying master])
9:        while(Master_Signal.Idle_Cycle[flying master]--) do
10:         Detail_Cycles_Cal(NULL,NULL,InitSt)
11:       end while
12:       Execute_Bus_Model(Master_Signal[flying master],NULL,InitSt)
13:       State=TransferGenSt
14:     end if
15:     else if (State=TransferGenSt) then
16:       Req_Gen(Master_Signal[flying master])
17:       Addr_Gen(Master_Signal[flying master])
18:       Data_Gen(Master_Signal[flying master])
19:       Transfer_Signal_Gen(Master_Signal[flying master])
20:       If(TransferExecSt[granted master]&(addr[granted master]==addr[flying master]))
then
            State=HoldSt
21:       end if
22:       else then
            State=TransterExecSt
23:       end else
24:     end else if
25:     else if (State=Holdst) then
26:      while (TransferSt[granted master]) do
27:         Detail_Cycles_Cal(NULL,NULL,ArbitrationSt)
28:         Execute_Bus_Model(Master_Signal[flying master],NULL,ArbitrationSt)
29:       end while
30:       State=TransferExecSt
31:     end else if
32:     else then  //State is TransferExecSt.
33:      while ((Master_Signal.Data_Size[flying master]+Slave_Signal.
Slave_Latency[selected slave])--) do
34:         Detail_Cycles_Cal(Master_Signal[flying master],Slave_Signal[all slaves].
TransferExecSt)
35:         Execute_Bus_Model(Master_Signal[flying master],Slave_Signal[all slaves].
TransferExecStSt)
36:       end while
37:       Master_Signal.Req[flying master]=False
38:       State=InitSt
39:     end else
40:   end while
41: end flying_bus_model()
```

그림 2와 표1의 알고리즘을 C++ 툴을 이용하여 구현하였으며 데이터 트랜잭션을 중심으로 시뮬레이션을 수행하는 TLM(Transaction Level Model) 방법을 이용하여 결과를 도출하였다. 버스중재방식은 fixed priority 방식, round-robin 방식, TDM 중재방식, Lottery 방식 등을 모두 적용하였으며, 일반적인 버스 아키텍처와 플라잉 마스터 버스 아키텍처의 성능을 분석하고 비교하였다. 슬레이브는 데이터 레이턴시가 길고, 가변적인 SDRAM 컨트롤러로 하였으며, 마스터의 개수를 4개로 두고 1,000,000 사이클 동안 시뮬레이션을 실행하였다. 그리고 슬레이브의 개수는 4개로 하였으며, 마스터는 4개의 슬레이브에 랜덤하게 접근하도록 하였다.

3. 플라잉 마스터 버스아키텍처 성능분석

그림 3과 4는 일반적인 버스와 플라잉 마스터 버스에서의 fixed priority 방식과 round-robin 방식에서의 마스터 점유율을 보여준다. Case1, Case2, Case3, Case4는 데이터 통신의 빈번함에 차이가 있으며, Case1에서 Case4로 갈수록 idle cycle의 길이가 길어진다.

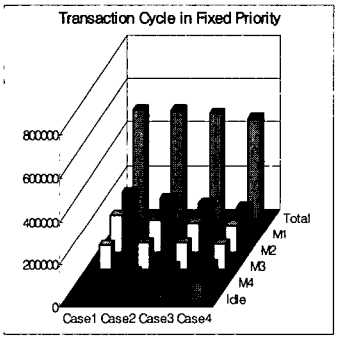
그림 3(a)를 보면, 평균 idle cycle이 짧은 Case1에서는 우선순위에 의한 마스터별 버스 점유율이 큰 차이를 보이고 있으나 평균 idle cycle이 긴 Case4에서는 마스터별 버스 점유율의 차이가 거의 없음을 알 수 있다.

또한 Case1의 경우 우선순위가 높은 마스터 M1과 가장 우선순위가 낮은 마스터 M4의 데이터 처리량의 차이가 크며, 또한 M4의 스타베이션이 발생하였다. 그림 3(b)의 플라잉 마스터 버스의 경우는 Case1에서 Case4로 갈수록 우선순위의 비중이 낮아지는 것은 그림 3(a)와 같다. 그러나 Case1의 경우 우선순위가 높은 마스터 M1의 데이터 처리량이 약 250,000 사이클이고 우선순위가 낮은 마스터 M4의 데이터 처리량이 100,000 사이클로 마스터의 스타베이션이 발생하지 않았음을 알 수 있다.

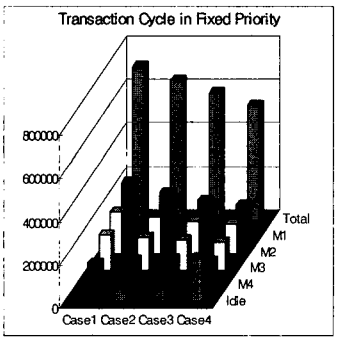
또한 총 데이터 처리량도 약 500,000 사이클에서 700,000 사이클로 200,000 사이클 증가하여 약 40%의 성능 향상을 가져왔다. 이는 데이터의 병렬 처리를 가능하게 할 수 있는 플라잉 마스터 버스 아키텍처의 장점이라 할 수 있다.

그림 4(a)의 일반적인 버스에서의 round-robin 방식의 경우는 Case에 상관없이 모든 마스터들에게 공평하게 버스 점유권이 배분되었다. 그림 4(b)의 round-robin 방식의 경우는 그림 4(a)와는 다르게 마스터 M1의 데이터 처리량이 다른 마스터 M2, M3, M4 보다 약 20,000 사이클에서 50,000 사이클이 많았다. 이는 중요한 마스터를 직접적으로 처리할 수 있는 플라잉 마스터 버스 아키텍처의 장점이다. 총 데이터 처리량 또한 일반적인 버스 아키텍처가 500,000 사이클인 것에 비해 약 200,000 사이클이 많은 700,000 사이클이었으며 이 또한 fixed priority와 비슷한 약 40%의 성능 향상을 보였다.

그림 5와 6은 일반적인 버스 아키텍처와 플라잉 마스



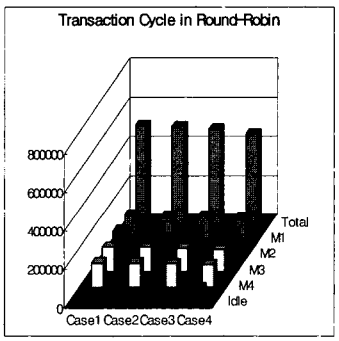
(a)



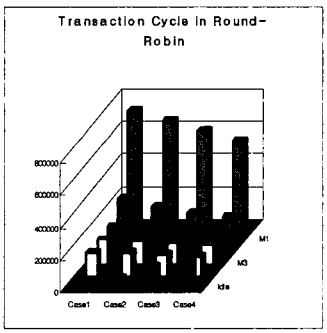
(b)

그림 3. (a) 일반적인 버스와 (b) 플라잉 마스터 버스의 fixed priority 방식에서 데이터 트랜잭션 비교

Fig. 3. Comparison of data transaction in fixed priority of (a) conventional bus architecture and (b) flying master bus architecture.



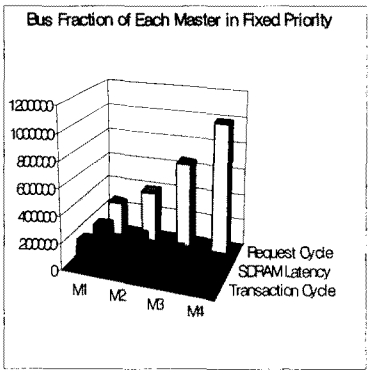
(a)



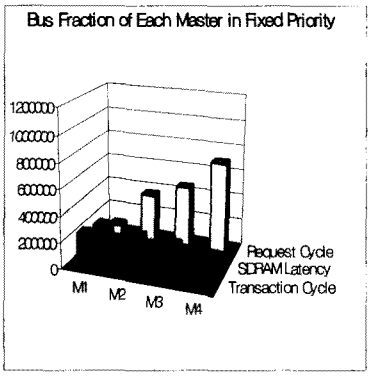
(b)

그림 4. (a) 일반적인 버스와 (b) 플라잉 마스터 버스의 round-robin 방식에서 데이터 트랜잭션 비교

Fig. 4. Comparison of data transaction in round robin of (a) conventional bus architecture and (b) flying master bus architecture.



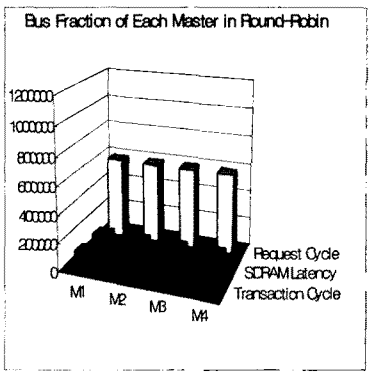
(a)



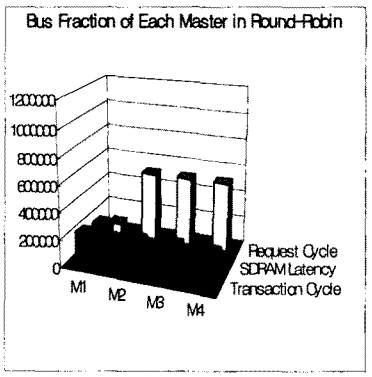
(b)

그림 5. (a) 일반적인 버스와 (b) 플라잉 마스터 버스의 fixed priority 방식에서 마스터별 레이턴시

Fig. 5. The latency of each master in fixed priority of (a) conventional bus architecture and (b) flying master bus architecture.



(a)



(b)

그림 6. (a) 일반적인 버스와 (b) 플라잉 마스터 버스의 round-robin 방식에서 마스터별 레이턴시

Fig. 6. The latency of each master in round-robin of (a) conventional bus architecture and (b) flying master bus architecture.

터 버스 아키텍처의 fixed priority 방식과 round-robin 방식의 각 마스터 당 데이터 처리량, SDRAM 레이턴시, 버스 요청 사이클을 보여주고 있다.

슬레이브는 SDRAM 컨트롤러이며, precharge 사이클, refresh 사이클, 로우 칼럼 액세스 사이클 등이 고려되었다. 그림 5, 6에서 보듯이 SDRAM 레이턴시가 데이터 처리 사이클(transaction cycle)과 유사할 정도로 상대적으로 큰 값을 나타내었다. 그리고 그림 5(a)의 마스터 M4는 버스요청 사이클이 1,000,000 사이클에 근접할 정도로 큰 값을 갖기 때문에 스타베이션이 발생했다고 할 수 있다. 반면에 그림 6(a)의 round-robin 방식의 경우는 스타베이션은 없고 각 마스터의 버스요청 사이클이 약 600,000 사이클로 균등하게 주어졌다.

이에 비해 플라잉 마스터 버스 아키텍처의 경우인 그림 5(b)와 6(b)의 경우를 보자. 먼저 그림 5(b)의 경우, 마스터 M4의 버스요청 사이클이 700,000 사이클로 그림 5(a)의 1,000,000 사이클에 비해 약 43%의 개선을 가져왔다.

이는 그림 5(a)에서 발생한 스타베이션이 플라잉 마스터 버스 아키텍처의 경우에서는 발생하지 않았음을 재확인할 수 있으며, 그림 6(b)의 round-robin 방식의 경우 또한 마스터 M1이 다른 마스터 M2, M3, M4보다 버스 요청 사이클이 적음을 알 수 있다.

이 또한 플라잉 마스터 버스 아키텍처에서 중요한 마스터 M1의 데이터 처리를 직접적으로 하고 나머지를 버스를 이용해서 병렬 처리를 수행했기 때문에 나타난 결과라 할 수 있다.

표 2는 일반적인 버스 아키텍처의 버스 중재방식별 마스터 점유율을 보여주고 있다. 각 마스터별 버스 점유율을 비교하여 보았을 때, fixed priority 방식의 경우는 우선순위가 높은 master1의 버스 점유율이 높음을 알 수 있다. Round-robin 방식의 경우는 모든 마스터에게 골고루 버스 점유권이 배분되었기 때문에 버스 점유율이 비슷함을 알 수 있다. TDMA 방식의 경우, M1에

표 2. 일반 버스아키텍처와 플라잉 마스터 버스아키텍처의 버스중재방식별 데이터 트랜잭션 비교
Table 2. Comparison of data transaction in each arbitration policy of conventional bus architecture and flying master bus architecture.

Master Type	Normal Bus Architecture				Flying Master Bus Architecture			
	Arbitration policy				Arbitration policy			
	Fixed Priority	Round-Robin	2-Level TDMA	2-Level Lottery	Fixed Priority	Round-Robin	2-Level TDMA	2-Level Lottery
Master1	203151	127952	178176	207242	248757	247727	248673	248977
Master2	172536	128142	133308	116352	192386	154424	175559	167940
Master3	108039	126713	110542	92376	162341	152296	141895	155963
Master4	24870	126658	86613	75218	104467	153419	142513	134414
Total	508596	509465	508639	491188	707951	707866	708640	707294

서 마스터 M4의 슬롯(slot) 수를 각각 3,1,1,1로 정하였으며, Lottery 방식의 경우는 마스터 M1에서 마스터 M6의 버스점유 확률을 3/6, 1/6, 1/6, 1/6으로 정하였다.[2] TDMA 방식의 경우는 마스터1이 가장 높고 마스터4의 경우가 가장 낮았다. Lottery 방식의 경우 또한 마스터1의 경우가 가장 높았다.

그러나 전체적인 총 마스터 점유율은 1,000,000 사이클 동안에 약 500,000 사이클 밖에 되지 않았다. 특히 마스터 M2에서 마스터 M4의 슬롯 수 또는 버스 점유 확률이 같았지만 데이터 트랜잭션 사이클이 서로 크게 다름을 알 수 있다. 이는 아비터에 의해 선택된 마스터가 버스 요청을 하지 않을 경우, 2순위 중재(2-level arbitration)에 의해서 다른 마스터가 버스 점유권을 받기 때문이다. 이는 일반적인 버스 아키텍처가 시스템적인 면에서 비효율적임 알 수 있는 부분이다. 플라잉 마스터 버스 아키텍처가 일반 버스에 비하여 모든 중재 방식에서 약 40%의 성능 향상을 보였다. 결국, 플라잉 마스터 버스 아키텍처의 병렬 처리로 인한 결과로서 플라잉 마스터 버스 아키텍처가 단지 fixed priority 방식과 round-robin 방식에만 적용되지 않고, 다른 버스 중재방식에도 효율적임을 알 수 있다.

Ⅲ. 결 론

본 논문에서 우리는 TLM 알고리즘을 구성하고, 새로운 버스 아키텍처 구조인 플라잉 마스터 버스 아키텍처를 제안하고 이를 일반적인 버스 아키텍처와 비교하여 성능을 분석하고 검증하였다. 일반적인 버스 아키텍처의 단점은 동시간대에 여러 개의 마스터가 일을 수행할 수 없다는 점이다. 이를 보완하고자 제안한 플라잉 마스터 버스 아키텍처의 경우, 중요한 마스터의 데이터 처리를 버스를 이용하지 않고 직접적으로 처리할 수 있기 때문에 동시간대에 다른 마스터가 버스를 이용하여 데이터를 처리할 수 있는 데이터의 병렬 처리가 가능하다. 또한 일반적인 버스 중재 방식인 fixed priority 방식과 round-robin 방식에만 적용 가능한 것이 아니라 새로운 중재 방식에도 적용할 수 있는 효율적인 시스템 버스 아키텍처라 할 수 있다. 앞으로의 플라잉 마스터 버스 아키텍처는 일반적인 SoC 버스 아키텍처로 자리매김할 것이며, 더욱 더 진보된 시스템의 성능 향상을 이끌 것으로 여겨지며, 본 논문이 중요한 자료로 역할을 할 것으로 여겨진다.

참 고 문 헌

[1] AMBA TM Specification(AHB) (Rev 2.0), ARM Ltd, May 1999.
[2] K. Lahiri, A. Raghunathan, and G. Lakshminarayana, "The LOTTERYBUS On-Chip Communication Architecture", IEEE Trans. VLSI Systems, vol.14, no.6, 2006.
[3] K. Sekar, K. Lahiri, A. Raghunathan, and S. Dey, "FLEXBUS: A high performance system-on-chip communication architecture with a dynamically configurable topology", in Proc. Design Autom. Conf., pp.571-574, 2005.
[4] K. Lee and Y. Yoon, "Architecture Exploration for Performance Improvement of SoC Chip Based on AMBA System", ICCIT, pp.739-744, 2007.
[5] 이국표, 윤영섭, "하이브리드 버스 중재 방식", 대한전자공학회 논문지, 심사중.
[6] 이국표, 윤영섭, "SoC를 위한 고성능 NAWM 버스 아키텍처", 대한전자공학회 논문지, 제45권, SD편, 제9호 게재예정.

저 자 소 개

이 국 표(정회원)
대한전자공학회 논문지
제45권 SD편 제4호 참조

윤 영 섭(정회원)
대한전자공학회 논문지
제45권 SD편 제4호 참조