논문번호 05-02-14

DMA(Direct Memory Access)을 이용한 SDRAM의 고속 인터페이스

SDRAM Fast Accession By DMA (Direct Memory Access)

金 眞 完*, 趙 鉉 默*★
Jin-Wan Kim*, Hyun-Mook Cho*★

Abstract

In this paper, we present the efficient way of SDRAM accessing through the DMA(Direct Memory Access) when a microprocessor and peripheral blocks are sharing a SDRAM. The microprocessor is able to access a memory through the AMBA which is the system bus provided by ARM Corporation and DMAs are able to access a memory through their own bus. Peripheral block's reading and writing on the SDRAM memory are realized by the intermediate DMA in order to minimize times of access and addressing the memory. While the microprocessor doesn't access to the SDRAM aproaching other registers or occurring a hit signal for fetching program or data, the DMAs may read/write the data in the SDRAM without an interference of the AMBA. This way increases the efficient of the system and performance is more by 16.8%.

요 약

본 논문에서는 마이크로프로세서와 주변블록 사이에서 SDRAM을 사용함에 있어서 DMA(Direct Memory Access)에 의한 효율적인 SDRAM 접근방식을 제시하고 있다. 여기에서 마이크로프로세서는 AMBA 버스를 통해서 SDRAM에 접근을 하고 DMA는 DMA 전용 버스를 통해서 SDRAM에 접근한다. 마이크로프로세서가 SDRAM에 접근하지 않고 다른 레지스터에 접근하거나, 아니면 마이크로프로세서 캐쉬에서 히트(hit)신호가 발생하여 SDRAM에 접근할 필요가 없을 때에 주변 블록에서는 DMA를 통해서 SDRAM에 접근하여 데이타를 읽거나 쓰기 동작을 통해서 SDRAM을 효율적으로 사용할 수 있다. 이 방법은 DMA가 마이크로프로세서의 SDRAM 억세스를 최소한의 방해로 SDRAM을 사용할 수 있다. 이와 같은 방법을 이용함으로써 전체적인 시스템 효율을 높여 약 16.8% 정도의 성능 향상 효과를 가져옴을 확인 할 수 있었다.

Keywords : mircoprocessor, AMBA, Sdram, DMA, high performance

Ⅰ. 서 론

오늘날 멀티미디어를 처리하기 위한 프로세서들의 처리 데이터 량은 갈수록 급증하고 있는 추세이다. 과거 프로세서들이 처리하는 데이터들은 데이터베이스를 기반으로 하는 자료들이 주류를 이루었으나, 최근의 동향은 고속통신을 통한 음성데이터, 화상데이터 등의 실시간 처리뿐만 아니라, 주어진 멀티미디어 데이터들을 실시간 연산 및 압축하는 기능까지도 요구되고 있는 실정이다. 이에 따라 데이터를 저장하는 메모리 매체들의 용량도 같은 비율로 급증하게 되고, 빠르고 효율적인 메모리 액세스에 대한 방법적 고찰이 새로운 아키텍처 개발의 중요한 이슈가 되고 있다. 또한, 현대의 멀티미디어 SOC(System on Chip)는 모바일 기기를 기반으로 한 저 전력 고 효율의 추세로, 한정된 동

* 公州大學校 情報通信工學部
(Division of Info. & Comm. Eng., Kongju Nat. Univ.)
★ 교신 저자 (Correspondence author)
E-mail : hmchov@kongju.ac.kr
接受日:2005年 11月 11日, 修正完了日: 2006年 6月 30日

작 주파수에서 높은 효율의 데이터 처리, 빠르고 효과적인 메모리 액세스는 곧 제품의 경쟁력을 의미한다. 결국, 제한된 휴대용 전력을 토대로 최대의 멀티미디어적 성능을 이루기 위하여 아키텍처적인 저 전력 고효율에 대한 연구 개발이 절실하다. 본 논문에서는 현재 모바일용 저 전력 프로세서로 널리 사용되고 있는 32-bit ARM 프로세서를 내장한 SOC 설계에 있어서 효과적인 SDRAM 액세스 방법을 제안한다. 제안된 방법은 DMA를 Host로 사용하고 AMBA 버스를 ARM과의 공유 버스로 이용하는 기존의 방식으로부터 탈피하여, ARM과 DMA의 전용버스를 각각 따로 두고 SDRAM 컨트롤러에 포함된 Arbiter가 우선순위에 따라 메모리를 엑세스하는 방식을 채택하였다. 본 논문에서 사용한 응용의 예는 2D, 3D를 이용하여 그래픽을 블렌딩, 텍스처링, 스케일링, Z 버퍼링과 같은 렌더링에 의해서 변형, 이동, 회전, 하는 테스크를 수행한다. ARM은 API를 통해서 어플리케이션을 수행하고, 주변 블록인 2D, 3D는 이 어플리케이션의 수행 속도를 높이기 위한 가속기로 사용되어 SDRAM과는 DMA를 통해서 데이터를 주고 받는다[1],[3].

II. 시스템구성

여기서 SDRAM은 클록속도가 마이크로프로세서와 동기화되어 있는 DRAM의 다양한 종류를 모두 일컫는 일반 명칭이다. 클록속도의 동기화는 주어진 시간 내에 프로세서가 수행할 수 있는 명령어 개수를 증가시키는데 도움을 준다. 즉, SDRAM은 훨씬 높은 주파수에서 데이터의 이동이 가능한데, 이로 인해서 3D와 같은 그래픽을 위한 데이터 흐름의 특성이 좋아진다[5]. 따라서 좀 더 복잡한 상태도를 가지며 연속적으로 빠른 속도로 데이터의 이동이 가능한 다중 모드(burst mode)를 지원한다. 따라서, 본 논문에서는 이를 만족하도록 SDRAM 컨트롤러를 설계하였으며 그림 1에 SDRAM 컨트롤러의 블록도를 나타내었다. 그림 1에서와 같이 SDRAM 컨트롤러 블록은 몇 개의 세부블록으로 구성되어있으며 ARM에 의한 SDRAM 접근은 AMBA 타이밍 규격을 따르고 있고, DMA에서 SDRAM 접근시에는 DMA 자신의 간단한 타이밍 규격에 준해서 움직인다. SDRAM 컨트롤 블록을 구성하는 각 세부 블록에 대해서 간략하게 설명을 한다.

2.1 AHB Interface & Address Generator

ARM으로부터 메모리에 접근하고자 할 때 ARM은 리퀘스터를 발생 하며 동시에 관련 인터페이스 신호을 발생하게 되는데, ARM에서 리퀘스터가 발생되면 메모리 컨트롤러는 ARM에 관련된 시퀀스를 시작한다.

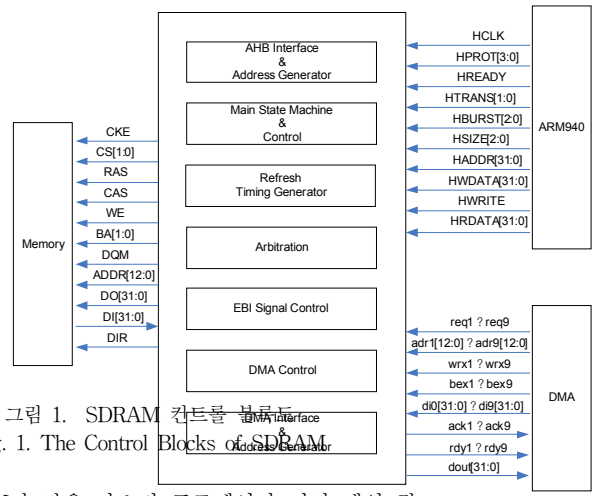

그림 1. SDRAM 컨트롤 블록도
Fig. 1. The Control Blocks of SDRAM

참고로 ARM과 같은 마스터 프로세서가 여러 개인 경우에는 AMBA에 있는 Arbiter에서 우선순위를 정하여 최종 하나의 마스터만이 정해진다[6],[8]. 이것이 다음 2.4절의 ARM 리퀘스터 소스가 된다.

2.2 Refresh Timing Generator

모든 DRAM 디바이스는 데이터의 보존을 위해서 항상 정해진 시간 안에 리프레쉬 기능을 수행해야 한다. 따라서, 리프레쉬 타이머는 적당한 시간에 리프레쉬 리퀘스터를 발생한다.

2.3 DMA Interface & Address Generator

DMA로부터 메모리를 직접 접근하고자 할 때 DMA는 리퀘스터와 관련 신호를 발생하는데, 인터페이스 신호로를 그림 1.에서 볼 수 있다. DMA로부터 리퀘스터가 발생되면 메모리 컨트롤러는 DMA에 관련된 시퀀스를 시작한다. 여기서 DMA의 리퀘스터 소스는 여러 개일 수 있는데 이는 이 블록 안에 있는 Arbiter에서 우선순위를 정하여 최종 하나만이 결정되어 다음 2.4절의 DMA 리퀘스터 소스가 된다.

2.4 Arbitration

ARM, DMA 그리고 Refresh로부터 인가되는 리퀘스터들의 우선 순위를 결정한다. 정해진 우선 순위에 의해서 최종 하나의 리퀘스터만이 선택된다. 여기서 최우선의 순위를 갖는 것은 리프레쉬인데, 이 리퀘스터가 발생되면 현재 실행되고 있는 태스크가 ARM과 관련된 경우에는 그 태스크가 끝난 후에 리프레쉬가 실행되고 DMA와 관련된 태스크인 경우에는 진행되고 있는 태스크를 잠시 중지시키고 리프레쉬를 수행한 후

에 그 중지된 태스크가 이어서 수행된다. 만약 arbitration이 idle 상태에서 리프레쉬 리퀘스터와 다른 리퀘스터가 동시에 발생되는 경우에는 리프레쉬가 먼저 수행된 후에 다른 태스크를 수행한다.

2.5 DMA Control

DMA 컨트롤러는 FIFO를 이용해서 각 DMA에서 입력된 어드레스, 데이터 그리고 기타 신호들을 처리하는 부분이다. 일반적으로 DMA로부터는 어드레스, 데이터, 그리고 신호들이 연속적으로 발생하게 되는데 이들은 일단 FIFO에 저장된 후에 필요에 따라 이들과 관련된 메모리 디바이스의 precharge, 뱅크 액티브, row 액티브가 실행된 후에 데이터 프로세싱이 수행된다.

2.6 EBI Signals Control

EBI(External Bus Interface)는 SDRAM 디바이스에 직접 연결되는 어드레스, 데이터 및 신호들을 컨트롤하는 블록이다. 이 블록에는 읽기/쓰기와 관련된 데이터의 멀티플렉싱 부분이 있고, 메모리 디바이스의 row 액티브 상태, 뱅크 액티브 상태의 정보를 가지고 있다. 필요에 따라서는 다른 뱅크나 row가 선택되도록 Main State Machine에 정보를 제공한다.

2.7 Main State Machine & Control

이 블록은 메모리 프로세싱의 시퀀스를 정의하고 관련된 명령 신호를 발생한다. 그림 2는 SDRAM의 타이밍을 컨트롤하는 스테이터 machine으로 SDRAM 인터페이스를 위한 외부 스트로브 신호를 발생하는 디코더 부분이다. 그래서 RAS, CAS 신호을 발생시켜 메모리 디바이스의 prechage 사이클, 뱅크 선택 사이클, row 액티브 사이클, 읽기 사이클, 쓰기 사이클, 그리고 리프레쉬 사이클 명령을 발생하는 시퀀스를 담당한다[5]. 그림 2의 상태도를 좀 더 구체적으로 설명하면 다음과 같다. 리셋 후에 start=0인 경우에는 'IDLE'에 있다. 이 상태로부터 읽기/쓰기/리프레쉬 사이클이 start로 부터 bnkact와 sref에 따라서 다음 스테이터로 진행된다. SDRAM이 초기화된 후에 모든 row는 비활성화 상태에 있다. 따라서 메모리가 접근되기 전에 디바이스의 row는 컨트롤러에 의해서 액티브 상태로 되어야 한다. 그렇지만 같은 뱅크에서는 오직 한 row만이 액티브 상태로 있을 수 있다. 따라서 4개의 뱅크가 있는 디바이스에서는 각각의 뱅크에서 하나씩 액티브 상태일 수 있으므로 최대 4 rows가 액티브 상태로 있을 수 있다. 만약에 어떤 뱅크에 이미 한 row가 액티브 상태에 있다면, 같은 뱅크에서 다른 row가 액티브되기 위해서는 이미 활성화 상태인 row는 비활성화 상태로 되어야 한다.

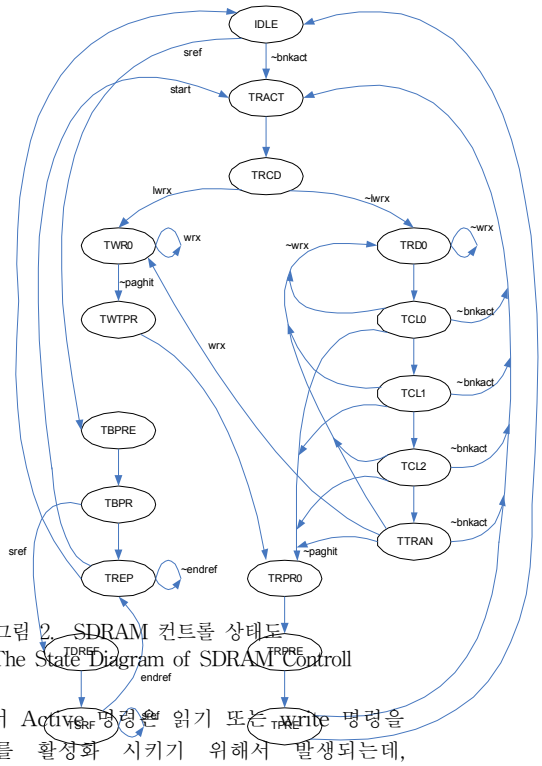

그림 2. SDRAM 컨트롤 상태도
Fig. 2. The State Diagram of SDRAM Controll

본 설계에서 Active 명령 후 읽기 또는 Write 명령을 위해서 row를 활성화 시키기 위해서 발생되는데, tRCD 지연 후에는 읽기/쓰기 명령이 발생될 것이다. 그런데, 읽기/쓰기 클럭 수는 이미 정해져 있고, 메모리 접근은 full 어드레스 범위에서 랜덤하다. 읽기/쓰기는 tRCD 지연을 만족하기 전에 클럭의 rising edge에서 샘플 되어지는데 lwrx에 의해서 결정된다. 만약에 high가 샘플 되면 TRD0, low가 샘플 되면 TWR0의 machine state가 된다. 읽기 경우에는 state machine은 CAS latency를 위해서 TRD0에서 TCL0, TCL1, TCL2로 바뀌게 되고 TCL2에서 SDRAM의 데이터는 버스 마스터로 이동된다. burst mode인 경우에는 wrx에 의해서 TRD0에서 feedback되고 TRD0, TCL0, TCL1에서도 데이터가 버스 마스터로 이동되지만 마지막 데이터는 TCL2에서 버스 마스터로 이동된다. Write 경우에는 state machine이 TWR0로 이동되며 데이터는 TWR0 상태에서 버스 마스터에서 SDRAM으로 이동된다. 읽기에서와 비슷하게, burst mode를 위해서는 wrx에 의해서 TWR0로feedback된다. 그런데, write시에는 write recovery time tWR이 필요하고 이 시간 후에 다른 명령이 올 수 있다. 표 1에서 위에서 설명하고 있는 신호들의 기능을 간단히 요약, 설명하였다.

표 1. 신호 설명
Table 1. The description of signals

신호	설 명
start	arbite의 리퀘스터에 의해 활성화됨
wrx	마스터의 읽기/쓰기 신호
lwrx	wrx의 1 클록 delay 신호
paghit	같은 뱅크에서 이전 활성화된 row와 지금 접근하려는 row가 같은지 여부 판단
bnkact	지금 접근하려는 뱅크가 활성화되어 있는지 판단
endref	리프레쉬가 동작 완료 신호
sref	셀프 리프레쉬 동작 신호

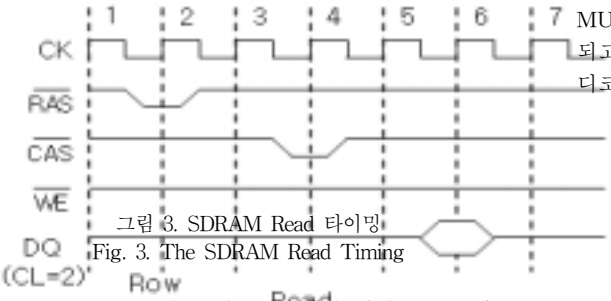

그림 3. SDRAM Read 타이밍
Fig. 3. The SDRAM Read Timing

그림 3은 SDRAM의 읽기를 할 때의 타이밍으로 어떤 뱅크의 Row가 활성화되어 있지 않은 경우에는 Row Actve 명령을 수행 한 후에 읽기 명령을 수행한다. 여기서 CL(Cas Latency)이 '2'인 경우 2클럭 후에 데이터가 출력됨을 보이고 있다. 또한, 그림 4는 SDRAM의 쓰기를 할 때의 타이밍을 보여 주고 있다. 역시 Row를 활성화 시킨 후에 쓰기 명령을 수행한다.

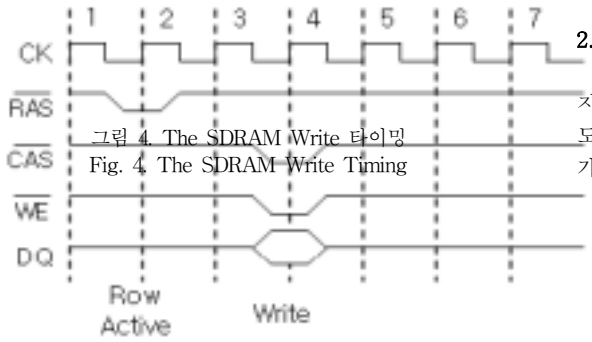

그림 4. The SDRAM Write 타이밍
Fig. 4. The SDRAM Write Timing

2.8 AMBA 버스와 DMA 버스에 의한 효과적인 데이터 control 구조

ABMA 버스의 기본적인 구조를 그림 5에 나타내었다. 각각의 마스터에서 리퀘스터 신호를 Arbiter에 보내면 Arbiter에서는 GRANT와 MASTER을 발생하고 이 신호에 의한 어드레스의 멀티플렉싱 과정과 어드레스의 디코딩에 의하여 slave를 선택하는 select 신호를 발생하게 된다.

그림 5에서 2개의 마스터가 있는 것으로 가정했으며 Mater #1, Master #2에서 REQ1, REQ2가 발생되면 Arbiter에서는 이미 정해진 우선순위에 의해서 둘 중에서 우선순위가 높은 것이 선택되고 관련 GRANT가 액티브 상태로 된다. GRANT를 받은 마스터는 AMBA 버스의 신호를 발생하게 된다. 즉, HADDR, HWDATA, HWRITE등이다. 신호 MASTER는 Master #1, Master #2 중에서 Arbiter에 의해서 선택된 마스터의 신호를 선택하는데 사용된다. 그림 5에서 MUX의 출력 값이 Slave의 어드레스 입력으로 연결되고, 디코더 블록에서는 이 어드레스의 상위 비트를 디코딩하여 Slave를 선택하는 신호를 발생한다.

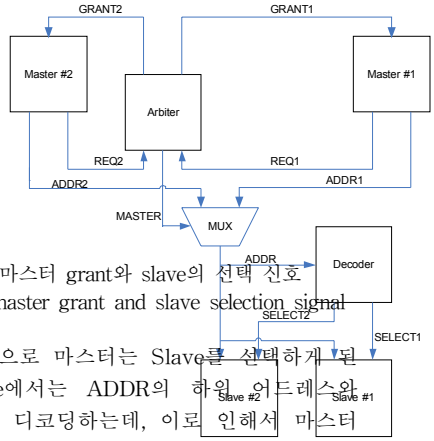

그림 5. 버스 마스터 grant와 slave의 선택 신호
Fig. 5. The Bus master grant and slave selection signal

위와 같은 과정으로 마스터는 Slave를 선택하게 된다. 그러면 Slave에서는 ADDR의 하위 어드레스와 SELECT#를 local 디코딩하는데, 이로 인해서 마스터는 Slave의 레지스터를 읽고 쓰는 것이 가능하다 [2],[4]. 여기에서 DMA는 마스터중의 하나가 될 수 있다.

2.8.1 ARM과 DMA의 AMBA 버스 공유

일반적으로 ARM은 SDRAM을 접근하는 경우도 있지만 RAM이나 일반 파일 레지스터를 접근하는 경우도 많이 있다. 그림 6 에서 SDRAM은 ARM에서 접근 가능한 경우와 DMA에서 접근 가능한 경우를 볼 수

있다. Arbiter와 Decoder가 있어 2개의 마스터 즉, DMA와 ARM에서 동시에 SDRAM을 접근하는 경우에는 Arbiter에서 acknowledge를 발생하여 하나의 마스터에 버스를 사용할 수 있는 권한을 준다. 그리고 Decoder는 slave SDRAM 또는 File 레지스터로 부터의 데이터를 muxing하기 위하기 위해서 사용된다. 만약 DMA의 우선순위 레벨이 높으면 DMA가 AMBA 버스를 점유하고 있기 때문에 ARM의 성능이 낮아진다. 그리고 만약 ARM의 우선순위 레벨이 높으면 ARM이 AMBA 버스를 점유함으로써 DMA의 성능이 떨어진다.

본 논문에서는 DMA가 ARM 보다 우선순위 레벨이 높다고 가정한다. Arbiter와 Decoder는 그림 5 과 같은 구조로 연결되어 있는데, 이 그림에서는 도면의 복잡성을 피하기 위해서 간략하게 그렸다. 그림 6 에서, DMA가 AMBA 버스를 점유하여 SDRAM을 접근하고 있으면 ARM은 물론 우선순위가 낮기 때문에 SDRAM을 접근할 수 없다. 때문에 ARM은 idle 상태로 있으면서 DMA의 SDRAM 접근이 끝날 때까지 기다린다. 더 이상의 DMA 리퀘스터가 없으면 arbitrator에서는 ARM에 acknowledge를 발생한다. 그런데, DMA가 SDRAM을 접근하고 있으면 ARM이 File 레지스터를 접근할 수도 없다. 이미 DMA가 AMBA 버스를 점유하고 있기 때문이다. 따라서 ARM이 AMBA 버스를 점유하기 전까지는 File 레지스터를 접근하지 못하고 idle 상태로 기다려야 한다. 따라서, SDRAM에 대해서 2개의 마스터에 의해서 접근되는 구조에서는 그림 6 와 같은 구조는 비효율적이라는 것을 쉽게 알 수 있다.

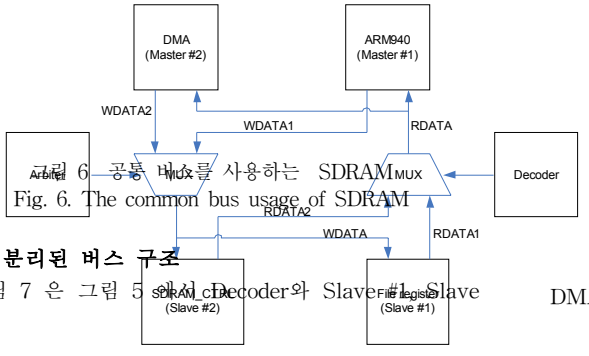

그림 6 공통 버스를 사용하는 SDRAM
Fig. 6. The common bus usage of SDRAM

2.8.2 분리된 버스 구조

그림 7 은 그림 5 에서 Decoder와 Slave #1, Slave #2을 제외한 형태이다. 본 설계는 그림 8에 나타낸바와 같이 ARM 버스와 DMA 버스가 분리되어 있는 형태의 구조를 가지며 SDRAM_CTRL 내부에는 그림 7 와 같은 구조의 Arbiter가 내장되어 있다. 그림 7와 그림 8을 보면, Master #1은 ARM940으로 Master #2는 DMA로 대응된다. 만약, DMA가 DMA 버스를 통해서 SDRAM을 접근하기 위해서 리퀘스터를 발생하고 ARM은 AMBA 버스를 통해서 SDRAM을 접근 하고자 리퀘스터를 발생하면 Arbiter는 이미 정해진 우선순위에 의해서 우선 순위를 정한다. 앞의 그림 1을 참조하면 HWRITE, HWDATA, HRDATA, HADDR 등은 ARM과 연결되어 있고, req#, addr#, wrx#, di#, dout# 등은 DMA에 연결되어 있는 것을 볼 수 있다. 그림 8 을 좀더 구체적으로 설명하면 다음과 같다.

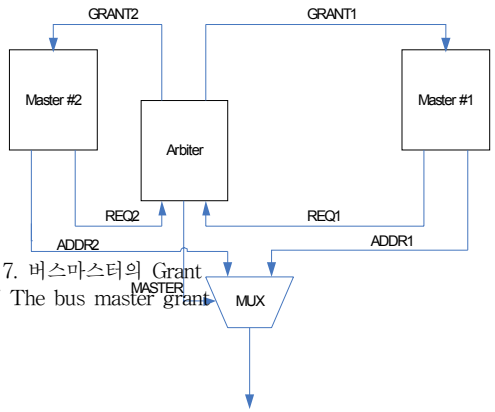

그림 7. 버스마스터의 Grant
Fig. 7 The bus master grant

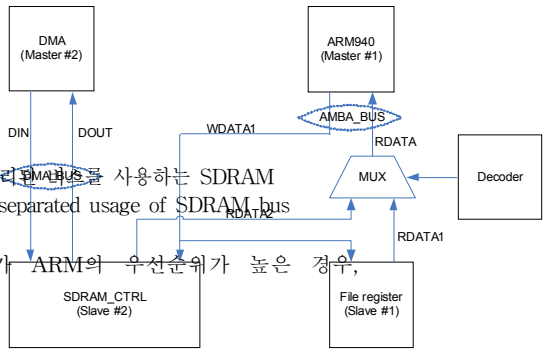

그림 8. 분리된 버스를 사용하는 SDRAM
Fig. 8 The separated usage of SDRAM bus

DMA 우선순위가 ARM의 우선순위가 높은 경우,

DMA에서 SDRAM을 접근하고 있거나, 아니면 DMA와 ARM에서 동시에 리퀘스터를 발생하면 ARM은 DMA의 메모리 접근이 끝날 때까지 기다려야 한다. 이때 SDRAM 컨트롤러는 ARM으로 HREADY active 상태의 신호를 보낸다. 그리고 DMA의 동작이 완료되고 ARM에서 필요로 하는 데이터가 유효할 때까지 ARM은 대기 상태로 있게 된다. 그러나 DMA가 SDRAM을 접근하고 있을지라도 ARM이 File 레지스터를 접근하려 한다면 AMBA 버스를 점유하여 접근이 가능하다.

다음으로 ARM의 우선순위 레벨이 DMA의 우선순위 레벨 보다 높으면 ARM은 DMA에 우선하여 SDRAM의 접근이 가능하고 File 레지스터도 접근 가능하다. 위의 설명에서 어느 경우든지 File 레지스터를 사용하고 있고 ARM의 File 레지스터 의존도가 높을수록 이 구조에 의한 성능은 더욱 높아진다. 일반적으로, ARM은 AMBA 규격에 의해서 동작을 하기 때문에 문제가 되지 않지만 DMA는 AMBA 규격을 만족하지 않기 때문에, ARM과 DMA의 메모리 접근이 바뀌는 시점에 문제가 발생 될 수 있다. 따라서 본 설계에서는 SDRAM의 State Machine과 arbiter에서 AMBA 규격을 만족하도록 하였다.

Ⅳ. 시뮬레이션 결과

본 설계에서는 메인 프로그램과 데이터 메모리로 SDRAM을 사용하고 있는데, SDRAM은 일반적으로 속도가 늦다. 그래서 ARM의 데이터 임시 저장 영역으로 속도가 빠른 SRAM이 있고 주변 블록에는 많은 레지스터들이 있는데 이는 AMBA 버스를 이용한다. 여기에서 ARM은 이런 메모리을 사용하기 위해서 DMA의 영향을 받지 않고 AMBA 버스를 점유하기를 원한다. 이와 같은 이유로 본 설계에서는 버스를 분리시키는 방법을 구현했고 표 2와 표 3에서 버스 분리에 의한 성능 개선 효과가 정리되어 있다.

일반적으로, ARM이 프로그램을 수행하는 동안 SRAM과 레지스터에 접근하는 횟수는 프로그램에 따라 다르겠지만 데이터의 변화량이 많은 응용일수록 많을 것이다. 이는 ARM의 AMBA 버스 점유율이 높아짐을 의미한다. 여기서는, 메모리 접근 효율의 개선 효과를 증명하기 위해서 2D와 3D 이미지 데이터를 모니터에 출력하는 일반적인 멀티미디어 응용을 예로 들었고, 이 경우의 메모리 접근 비율을 측정하여 표 2와 표 3에 정리하였다.

표 2. ARM과 DMA의 SDRAM 접근 비율
Table 2. SDRAM access rate for ARM & DMA

호스트	접근비율 (%)
ARM	30
DMA	70
총	100

표 3. ARM과 DMA 각각의 메모리 접근 비율
Table 3. Memory access rate for ARM & DMA

항목 \ 호스트	ARM(%)	DMA(%)
register	10	
SRAM	30	
SDRAM	60	100
총	100	100

표 2는 ARM과 DMA가 모니터에 디스플레이 될 데이터의 한 프레임 동안 SDRAM에 접근하는 횟수를 백분율로 나타낸 것이며 표 3은 ARM과 DMA가 모니터에 디스플레이 되는 데이터의 한 프레임 동안 각각의 메모리에 접근하는 횟수를 백분율로 나타낸 것이다. 상기의 응용 프로그램에서 AMBA 버스가 호스트 ARM과 호스트 DMA를 위해서 공용으로 사용되는 경우 두 호스트에서 동시에 AMBA 버스를 점유되는 버스 충돌 계수는 평균 0.6 정도 이다. 표 2, 표 3의 경우는 ARM과 DMA 중에 DMA가 priority가 높다고 가정한다. 표 2에서 AMBA버스를 ARM과 DMA가 공통으로 사용한다면 DMA가 버스를 점유하고 있는 동안에 ARM은 hold 상태가 된다. 만약에 ARM의 캐쉬가 사용되지 않는다면 ARM은 프로그램 및 데이터를 항상 SDRAM에서 읽어 와야 할 것이고 충돌 개수는 1이 될 것이다[7]. 그러나 캐쉬가 사용되고 있고 캐쉬 Hit ratio를 약 65%로 충돌 개수 0.6 정도가 되는 것을 볼 수 있었다. 표 2는 이 ratio가 고려된 자료이다. 여기에서 표 2의 DMA 70%에 충돌 계수 0.6을 고려하면 42%의 충돌이 예상된다. 표 3에서는 register와 SRAM을 위해서 총 40%의 비율을 보이고 있어, 위의 계산 42%의 40%는 16.8% 이므로 버스를 분리하여 설계함으로써 ARM의 성능이 16.8% 개선 되었음을 알 수 있다. 그림 9과 그림 10은 본 설계와 관련한 시뮬레이션 결과를 보여주고 있다. 그림 9은 동일 버스인 경우로 DMA가 AMBA를 점유하고 있기 때문에 ARM의 어드레스 HADDR에는 값의 변화 없이 어떤 상태를 계속 유지하다가 DMA가 기능이 끝난 후에 값의 변화를 보이는 시뮬레이션 결과를 나타내고 있다.

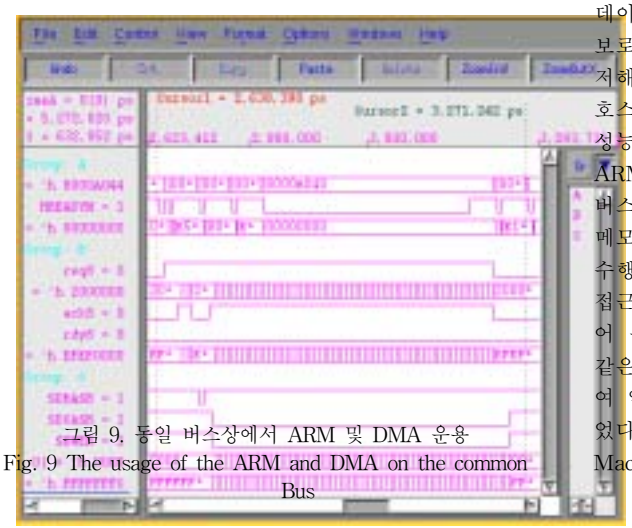

그림 9. 동일 버스상에서 ARM 및 DMA 운용
Fig. 9 The usage of the ARM and DMA on the common Bus

결과에서 보면 DMA가 AMBA 버스를 점유하고 있는 동안 ARM은 idle 상태로 되어 시스템 성능을 저하하는 요인이 되고 있다. 그림 10은 ARM과 DMA의 버스가 분리된 구조를 시뮬레이션 한 결과를 보여 주고 있다. 그림 10에서 DMA의 어드레스 버스인 addr#의 값이 변하고 있는 동안에도 HADDR의 값이 DMA에 영향을 받지 않고 계속 변하고 있다.

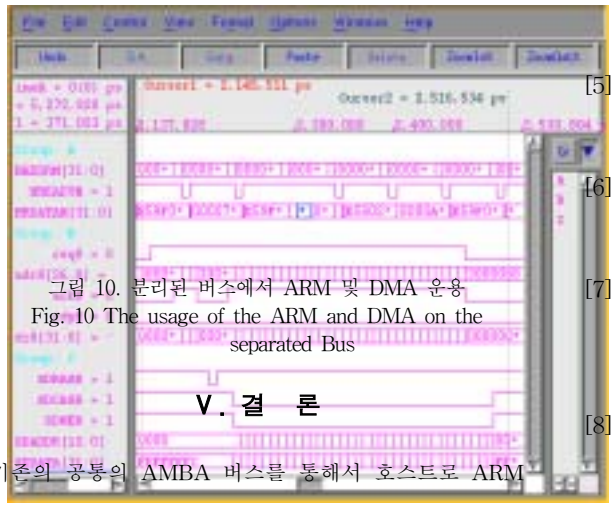

그림 10. 분리된 버스에서 ARM 및 DMA 운용
Fig. 10 The usage of the ARM and DMA on the separated Bus

V. 결 론

기존의 공통의 AMBA 버스를 통해서 호스트로 ARM과 DMA의 동작을 지원하는 상태에서는 버스 상에서 데이터 충돌(conflict) 현상이 발생하여 어느 한쪽이 양보로 칩 동작이 가능한 구조였다. 이는 칩의 성능을 저해하는 주요 원인이 되었다. 따라서, 본 설계에서는 호스트 ARM과 DMA의 상호 데이터 충돌에 의한 칩 성능 저하를 개선하는 방법으로 설계를 진행 했다. 즉 ARM은 AMBA 버스를 사용하고, DMA는 DMA 전용 버스를 사용함으로서 ARM에서 SDRAM이 아닌 다른 메모리에 접근하는 경우에는 DMA의 동작에 관계없이 수행이 가능하다. 두 호스트에서 동시에 SDRAM에 접근하는 경우에는 SDRAM 컨트롤러에 Aribiter을 두어 우선순위를 결정하는 방법으로 설계 되었다. 이와 같은 방법을 이용함으로써 전체적인 시스템 효율을 높여 약 16.8% 정도의 성능 향상 효과를 확인 할 수 있었다. 물론 이러한 기능을 제어하기 위해서 State Machine이 복잡해지는 단점은 부가적으로 발생했다.

참 고 문 헌

[1] Flynn,D. *"Enabling reusable on-chip designs"*, Publication Date: July-Aug. 1997

[2] Young woo Kim and Kyoung Park *"Based multiprocessor system"*, Electron. & Telecommunication. Res. Inst., Daejeon, South Korea *This paper appears in:*

[3] System-on-Chip, 2003. Proceedings. International Symposium on Publication Date: 19-21 Nov. 2003 On page(s): 41 42

[4] Using formal techniques to debug the AMBA system-on-chip bus protocol Roychoudhury, A.; Mitra, T.; Karri, S.R.; Publication Date: 2003

[5] D. Bursky, *"Combo RISC CPU and DRAM Solves Data Bandwidth Issues"*, Electronic Design, Mar.4, 1996.

[6] Dave Jaggar. *"ARM Architecture and Systems"*, IEEE Micro, vol. 17, no. 4, pp. 9-11, July/August, 1997.

[7] R. Cucchiara, M. Piccardi, and A. Prati. Exploiting cache in multimedia. In International Conference on Computing and Systems, Florance, Italy, 1999. IEEE.

[8] Friederich Mombers and Daniel Mlynek. A multithreaded multimedia processor merging on-chip multiprocessors and distributed vector

pipelines. In Proceedings of the International Symposium on Circuits and Systems ISCAS 1999, pages 287-290. IEEE, 1999.

저 자 소 개

김진완 (정회원)

1990년 : 고려대학교 전기공학과졸업(공학사)
1990년 : 하이닉스 주식회사
2002년~현재 디게이트 주식회사
2004년~현재 공주대학교 대학원 전기전자정보공학과 석사과정
<주관심분야>
SoC 설계, 멀티미디어 시스템

조현묵 (정회원)

1989년 : 고려대학교 전자공학과 졸업(공학사)
1991년 : 고려대학교 일반대학원 (공학석사)
1995년 : 고려대학교 대학원 전자공학과 졸업(공학박사)
1995년~ 현재 공주대학교 정보통신공학부 교수
<주관심분야>
SoC 설계, 멀티미디어 시스템 설계 등