



(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2010-0056652
(43) 공개일자 2010년05월28일

(51) Int. Cl.

G06F 13/36 (2006.01) G06F 13/14 (2006.01)

(21) 출원번호 10-2008-0115555

(22) 출원일자 2008년11월20일

심사청구일자 2008년11월20일

(71) 출원인

인하대학교 산학협력단

인천 남구 용현동 253 인하대학교

(72) 발명자

윤영섭

서울 서초구 잠원동 한신아파트 8차 308동 801호

이국표

인천광역시 남동구 만수3동 벽산아파트 109동 2103호

(74) 대리인

황광연, 강동호, 오정환

전체 청구항 수 : 총 5 항

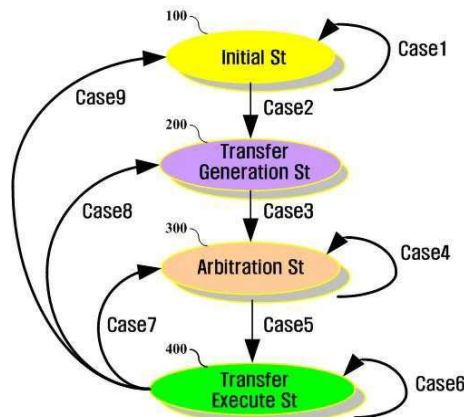
(54) 버스 시스템 성능 분석 방법

(57) 요약

본 발명은 버스 시스템에 관한 것으로, 특히 데이터 송수신의 동작에 기초한 분석 기법을 통해, 성능 분석 시간을 줄일 수 있는 버스 시스템 성능 분석 방법에 관한 것이다.

본 발명의 실시 예들에 따른 버스 시스템 성능 분석 방법은 적어도 하나의 버스를 포함하는 버스 시스템의 성능 분석 방법에 있어서, 버스 시스템에서 수행되는 데이터 송수신의 동작에 기초하여 복수의 상태를 정의하는 단계; 및 상기 복수의 상태에 기초한 데이터의 송수신 과정에서 발생하는 복수의 케이스의 조건을 정의하는 단계를 포함하고; 상기 복수의 상태 및 복수의 케이스의 조건에 관련된 사항들을 함수로 생성하고, 상기 함수를 알고리즘으로 구현하여 이루어지는 것을 특징으로 한다.

대표도 - 도5



특허청구의 범위

청구항 1

적어도 하나의 버스를 포함하는 버스 시스템의 성능 분석 방법에 있어서,

버스 시스템에서 수행되는 데이터 송수신의 동작에 기초하여 상기 버스에서 수행되는 복수의 상태를 정의하는 단계; 및

상기 복수의 상태에 기초한 데이터의 송수신 과정에서 발생하는 복수의 케이스에 따른 조건을 정의하는 단계를 포함하고;

상기 복수의 상태 및 복수의 케이스의 조건에 관련된 사항들을 함수로 생성하고, 상기 함수를 알고리즘으로 구현하여 이루어지는 것을 특징으로 버스 시스템 성능 분석 방법.

청구항 2

제 1 항에 있어서, 상기 복수의 상태는

마스터를 통한 버스 이용 요청이 발생하지 않아 버스의 동작이 없는 상태를 의미하는 초기 상태;

복수의 마스터 중 적어도 하나의 마스터에서 버스를 사용하고자 하는 경우의 상태를 의미하는 전송 준비 상태;

미리 설정된 버스 중재 방법을 이용하여 복수의 마스터 중에서 하나의 마스터에게 버스 이용권한을 할당하는 상태를 의미하는 버스 중재 상태; 및

실제로 데이터의 송수신 처리가 이루어지는 상태를 의미하는 전송 수행 상태를 포함하는 것을 특징으로 버스 시스템 성능 분석 방법.

청구항 3

제 2 항에 있어서, 상기 버스 시스템이 복수의 버스를 포함하는 경우,

상기 복수의 상태는 서로 다른 버스 간의 데이터 송수신을 제어하는 브리지의 동작 상태를 의미하는 브리지 상태를 더 포함하고;

상기 브리지 상태는 상기 버스 중재 상태로부터 진입되어, 마스터로부터 데이터의 전송이 이루어지는 슬레이브가 접속된 버스의 이용 가능 여부를 확인하는 제 1 브리지 상태; 및

상기 전송 수행 상태로부터 진입되어, 상기 마스터로부터 상기 슬레이브가 접속된 버스로 데이터의 전송이 수행되는 제 2 브리지 상태를 포함하여 이루어지는 것을 특징으로 하는 버스 시스템 성능 분석 방법.

청구항 4

제 1 항에 있어서,

적어도 하나의 버스에서 수행되는 데이터 송수신과 관련된 함수 및 상기 적어도 하나의 버스에서 수행되는 중재 방법이 정의된 함수를 알고리즘으로 구현하여 이루어지는 것을 특징으로 하는 버스 시스템 성능 분석 방법.

청구항 5

제 1 항에 있어서, 상기 복수의 케이스는,

모든 마스터의 아이들 사이클이 0 보다 큰 경우에 수행되는 제 1 케이스;

상기 모든 마스터 중에서 어느 하나의 마스터의 아이들 사이클이 0인 경우에 수행되는 제 2 케이스;

상기 버스의 동작 수행을 상기 버스 중재 상태로 진입시키는 제 3 케이스;

상기 버스의 동작 수행을 버스 중재 사이클 기간 동안 버스 중재 상태를 유지시키도록 수행되는 제 4 케이스;

상기 버스 중재 사이클 기간이 완료되면 상기 버스의 동작 수행을 상기 전송 수행 상태로 진입시키는 제 5 케이스;

상기 버스를 이용하여 마스터와 슬레이브 간에 데이터의 송수신이 이루어지는 기간을 정의하는 제 6 케이스;

복수의 마스터 중에서 어느 하나의 마스터에서 버스 이용의 요청이 있는 경우에 상기 버스의 동작 수행을 상기 버스 중재 상태로 진입시키는 제 7 케이스;

모든 마스터들로부터 버스 이용 요청이 없고, 복수의 마스터들 중에서 어느 하나의 마스터의 아이들 사이클이 0 인 경우에 상기 버스의 동작 수행을 상기 전송 준비 상태로 진입시키는 제 8 케이스; 및

모든 마스터들의 아이들 사이클이 0이 아닌 경우에 상기 버스의 동작 수행을 상기 초기 상태로 진입시키는 제 9 케이스를 포함하고;

상기 제 1 내지 제 9 케이스 각각의 조건과 관련된 사항들을 함수로 정의하고, 상기 함수를 알고리즘으로 구현 하여 이루어지는 것을 특징으로 하는 버스 시스템 성능 분석 방법.

명세서

발명의 상세한 설명

기술 분야

[0001] 본 발명은 버스 시스템에 관한 것으로, 특히 데이터 송수신의 동작에 기초한 분석 기법을 통해, 버스 시스템의 성능 분석 시간을 줄일 수 있는 버스 시스템 성능 분석 방법에 관한 것이다.

배경 기술

[0002] 반도체 제조 공정의 발달에 힘입어 서로 다른 기능을 수행하는 여러 컴포넌트들을 하나의 칩(Chip)에 집적시키는 SoC(System on Chip) 기술이 개발되어 회로설계에 널리 이용되고 있다. 이러한, SoC(System on Chip)는 시스템 설계와 구현 시 칩 크기와 설계비용을 점감시킴과 아울러, 구동 시 저전력 소모와 실시간 처리 능력 및 시스템의 유연성 면에서 향상된 성능을 제공할 수 있다.

[0003] SoC의 구성요소는 복수의 마스터(Master)와 중재기(Arbitrer), 복수의 슬레이브(Slave) 및 마스터와 슬레이브 간의 데이터 송신을 위한 버스(Shared Bus)로 이루어져 있다. 일반적으로 마스터는 프로세서(CPU), DMA(Direct Memory Access), DSP(digital signal processor)를 의미하며, 슬레이브는 SDRAM, SRAM과 같은 메모리와, USB(Universal Serial Bus) 및 UART(Universal Asynchronous Receiver Transmitter)와 같은 입력/출력 수단을 의미한다.

[0004] 하나의 칩에 여러 가지의 기능이 집적화된 SoC칩의 경우, 하나의 버스에 마스터, 중재기, 슬레이브 등이 연결된 버스 시스템이 필요하다. 다수의 마스터들이 하나의 버스를 이용하여 데이터를 송수신하고자 하는 경우, 중재기로부터 버스 이용 허가 신호를 받은 마스터들이 순서대로 버스를 이용하여 데이터를 슬레이브(메모리)로 송신하거나, 슬레이브로부터 데이터를 로딩하게 된다.

[0005] 다수의 마스터들이 하나의 버스를 동시에 이용하고자 하는 경우, 중재기에 의해 버스 이용 허가 신호를 받은 마스터들이 순서대로 버스를 이용하여 데이터를 슬레이브로 전송하게 된다. 이를 위해, 중재기는 다수의 마스터들이 동시에 버스 이용을 요청할 때 우선 순위를 결정하게 되는데, 중재기의 우선 순위 결정 방식에 따라 칩의 성능이 달라지게 된다.

[0006] 도 1은 일반적인 단일 버스 시스템을 나타내는 도면이다. 도 1에서는 종래의 여러 버스 시스템들 중에서 AMBA(Advanced Micro-controller Bus Architecture) 시스템을 일 예로 하였고, 버스 시스템을 구성하는 복수의 마스터들 및 슬레이브들 중에서 각각 2개만 도시하였다.

[0007] 도 1을 참조하면, 일반적인 단일(single) 버스 시스템(10)은 하나의 버스(30), 복수의 마스터(20, 22), 복수의 슬레이브(40, 42) 및 버스(30)를 이용을 중재하는 중재기(50)를 포함한다.

[0008] 복수의 마스터(20, 22) 각각은 복수의 슬레이브(40, 42) 각각으로 접속하고자 하는 경우 즉, 마스터들(20, 22)이 슬레이브들(40, 42)로 데이터를 송신하거나, 슬레이브들(40, 42)로부터 데이터를 로딩하고자 하는 경우, 중재기(50)에게 버스(30)의 이용을 요청하게 된다. 이때, 중재기(50)는 미리 정해진 우선 순위 판단 방법에 따라 마스터들(20, 22) 각각의 버스(30) 이용의 우선권을 결정하며, 중재기(50)로부터 버스(30)의 이용 승인을 획득한 마스터가 버스(30)를 통해 슬레이브들(40, 42)로 접속하게 된다.

- [0009] 예를 들어, 제 1 마스터(20)가 제 1 슬레이브(40)로의 접속을 요청하고, 동시에 제 2 마스터(32)가 제 2 슬레이브(42)로의 접속을 요청한 경우, 중재기(50)는 미리 정해진 우선 순위 판단 방법을 이용하여 버스(30) 이용의 우선권을 결정하게 된다. 일반적으로 CPU, ARM 프로세서와 같이 많은 연산을 수행하고 신속성을 요하는 마스터에 버스(30) 이용의 우선권을 부여한다. 여기서, 중재기(50)에 적용되는 우선 순위 판단 방법은 고정된 우선 순위(Fixed Priority) 판단 방법과 라운드 로빈 우선 순위(Round Robin Priority) 판단 방법이 가장 널리 이용된다.
- [0010] 도 2는 종래 기술에 따른 RTL(Register Transfer Level) 방식을 이용한 버스 시스템 성능 분석 방법을 나타내는 도면이다. 도 2에서는 복수의 마스터와 슬레이브 중에서 각각 1개만을 일 예로 도시하였다.
- [0011] 도 1에 도시된 버스 시스템의 성능을 분석하기 위하여 종래 기술에서는 도 2에 도시된 바와 같은 RTL(Register Transfer Level, 이하 "RTL"이라 함) 방식을 사용하고 있다.
- [0012] 이러한, RTL 방식을 이용한 버스 시스템 성능 분석 방법은 디지털 회로 설계 툴을 이용하여 실제 로직들을 설계 및 구현한 후, 버스 시스템의 성능을 분석하는 방법이다. 이러한 RTL 방식은 실제 로직들을 설계한 후, 복수의 마스터, 복수의 슬레이브 및 버스에 클럭(CLK, 60)을 입력하고, 상기 클럭(60)에 따라 실제 데이터의 송수신을 수행하여 버스 시스템의 성능을 분석하게 된다.
- [0013] 이러한, RTL 방식을 이용한 버스 시스템의 성능 분석 방법은 성능 분석의 대상이 되는 버스 시스템과 동일하게 실제 로직을 설계하여야 하는데, 마스터와 슬레이브의 수가 증가되면 이에 비례하여 로직 내의 라인이 증가하게 되므로 버스 시스템의 설계가 어렵고, 설계 및 구현에 많은 시간이 소요되게 된다. RTL 방식은 버스 시스템의 성능 분석을 위해 버스 시스템의 실제 로직에 대응되는 RTL 코드를 설계하여 성능 분석을 위한 시뮬레이션을 수행하게 되므로 성능 분석에 많은 시간이 소요되는 문제점이 있다.
- [0014] 특히, 도 3에 도시된 바와 같이, 복수의 버스 사이에 브리지를 형성하여 복수의 마스터와 복수의 슬레이브 간에 데이터를 송수신하는 다중(multi) 버스 시스템의 경우에는 로직 내의 라인이 더욱 증가하게 되고, 복수의 버스 시스템 각각에 대하여 성능 분석을 수행하여 전체 다중 버스 시스템의 성능을 분석하여야 하므로 성능 분석에 많은 시간이 소요되는 문제점이 있다.
- [0015] 또한, 다중 버스 시스템의 경우, 복수의 버스 간에 데이터를 송수신하는 경우에 브리지에서 대기 시간(latency) 발생할 수 있는데, 빠른 시간 내에 높은 정확도를 가지고 브리지에서 어느 정도의 대기 시간이 발생하여 버스 시스템의 성능에 영향을 미치는지를 평가할 수 있는 방법이 없다. 즉, 다중 버스 시스템의 경우, 종래 기술의 이용하여 성능 분석을 수행하게 되면 성능 분석의 정확도 저하 및 성능 분석에 많은 시간이 소요되는 문제점이 있다.
- [0016] 상술한, 단일 버스 시스템 및 다중 버스 시스템은 사용자의 요구에 따라 다양한 형태로 구성의 변경 및 데이터 전송 성능의 개선이 이루어지고 있다. 이에 따라, 개발이 이루어지는 버스 시스템의 성능 분석을 보다 정확하고 신속하게 처리하고자 하는 요구가 증대되고 있으나, 이를 지원할 수 있는 버스 시스템의 성능 분석 방법이 제시되지 못하고 있다.

발명의 내용

해결 하고자하는 과제

- [0017] 본 발명은 상술한 문제점을 해결하기 위한 것으로서, 단일 버스 시스템 및 다중 버스 시스템의 성능을 분석함에 있어, 성능 분석의 정확도를 향상시킬 수 있는 버스 시스템 성능 분석 방법을 제공하는 것을 기술적 과제로 한다.
- [0018] 본 발명은 상술한 문제점을 해결하기 위한 것으로서, 단일 버스 시스템 및 다중 버스 시스템의 성능을 분석함에 있어, 성능 분석 시간을 줄일 수 있는 버스 시스템 성능 분석 방법을 제공하는 것을 기술적 과제로 한다.

과제 해결수단

- [0019] 상술한 목적을 달성하기 위한 본 발명의 실시 예에 따른 버스 시스템 성능 분석 방법은 적어도 하나의 버스를 포함하는 버스 시스템의 성능 분석 방법에 있어서, 버스 시스템에서 수행되는 데이터 송수신의 동작에 기초하여 복수의 상태를 정의하는 단계; 및 상기 복수의 상태에 기초한 데이터의 송수신 과정에서 발생하는 복수의 케이스의 조건을 정의하는 단계를 포함하고; 상기 복수의 상태 및 복수의 케이스의 조건에 관련된 사항들을 함수로

생성하고, 상기 함수를 알고리즘으로 구현하여 이루어지는 것을 특징으로 한다.

- [0020] 상술한 목적을 달성하기 위한 본 발명의 실시 예에 따른 버스 시스템 성능 분석 방법에서 상기 복수의 상태는, 마스터를 통한 버스 이용 요청이 발생하지 않아 버스의 동작이 없는 상태를 의미하는 초기 상태; 복수의 마스터 중 적어도 하나의 마스터에서 버스를 사용하고자 하는 경우의 상태를 의미하는 전송 준비 상태; 미리 설정된 버스 중재 방법을 이용하여 복수의 마스터 중에서 하나의 마스터에게 버스 이용권한을 할당하는 상태를 의미하는 버스 중재 상태; 및 실제로 데이터의 송수신 처리(transaction)가 이루어지는 상태를 의미하는 전송 수행 상태를 포함하는 것을 특징으로 한다.
- [0021] 상술한 목적을 달성하기 위한 본 발명의 실시 예에 따른 버스 시스템 성능 분석 방법은, 상기 버스 시스템이 복수의 버스를 포함하는 경우, 상기 복수의 상태는 서로 다른 버스 간의 데이터 송수신을 제어하는 브리지의 동작 상태를 의미하는 브리지 상태를 더 포함하고; 상기 브리지 상태는 상기 버스 중재 상태로부터 진입되어, 마스터로부터 데이터의 전송이 이루어지는 슬레이브가 접속된 버스의 이용 가능 여부를 확인하는 제 1 브리지 상태; 및 상기 전송 수행 상태로부터 진입되어, 상기 마스터로부터 상기 슬레이브가 접속된 버스로 데이터의 전송이 수행되는 제 2 브리지 상태를 포함하여 이루어지는 것을 특징으로 한다.
- [0022] 상술한 목적을 달성하기 위한 본 발명의 실시 예에 따른 버스 시스템 성능 분석 방법은 적어도 하나의 버스에서 수행되는 데이터 송수신과 관련된 함수 및 상기 적어도 하나의 버스에서 수행되는 중재방법이 정의된 함수를 알고리즘으로 구현하여 이루어지는 것을 특징으로 한다.
- [0023] 상술한 목적을 달성하기 위한 본 발명의 실시 예에 따른 버스 시스템 성능 분석 방법에서, 상기 복수의 케이스는, 모든 마스터의 아이들 사이클이 0 보다 큰 경우에 수행되는 제 1 케이스; 상기 모든 마스터 중에서 어느 하나의 마스터의 아이들 사이클이 0인 경우에 수행되는 제 2 케이스; 상기 버스의 동작 수행을 상기 버스 중재 상태로 진입시키는 제 3 케이스; 상기 버스의 동작 수행을 버스 중재 사이클 기간 동안 버스 중재 상태를 유지시키도록 수행되는 제 4 케이스; 상기 버스 중재 사이클 기간이 완료되면 상기 버스의 동작 수행을 상기 전송 수행 상태로 진입 시키는 제 5 케이스; 상기 버스를 이용하여 마스터와 슬레이브 간에 데이터의 송수신이 이루어지는 기간을 정의하는 제 6 케이스; 복수의 마스터 중에서 어느 하나의 마스터에서 버스 이용의 요청이 있는 경우에 상기 버스의 동작 수행을 상기 버스 중재 상태로 진입시키는 제 7 케이스; 모든 마스터들로부터 버스 이용 요청이 없고, 복수의 마스터들 중에서 어느 하나의 마스터의 아이들 사이클이 0인 경우에 상기 버스의 동작 수행을 상기 전송 준비 상태로 진입시키는 제 8 케이스; 및 모든 마스터들의 아이들 사이클이 0이 아닌 경우에 상기 버스의 동작 수행을 상기 초기 상태로 진입시키는 제 9 케이스를 포함하고; 상기 제 1 내지 제 9 케이스 각각의 조건과 관련된 사항들을 함수로 정의하고, 상기 함수를 알고리즘으로 구현하여 이루어지는 것을 특징으로 한다.

효 과

- [0024] 상술한 바와 같이, 실시 예에 따른 본 발명은 단일 버스 시스템 및 다중 버스 시스템의 성능 분석의 정확도를 향상시킬 수 있다.
- [0025] 실시 예에 따른 본 발명은 단일 버스 시스템 및 다중 버스 시스템의 성능 분석 시간을 줄일 수 있다.
- [0026] 실시 예에 따른 본 발명은 단일 버스 시스템 및 다중 버스 시스템의 성능 분석의 정확성 향상 및 성능 분석 시간을 줄여, SoC 개발에 소요되는 시간을 줄임으로써 사용자의 요구에 빠르게 대응할 수 있다.

발명의 실시를 위한 구체적인 내용

- [0027] 본 발명은 버스 시스템에 관한 것으로, 특히 데이터 송수신의 동작에 기초한 분석 기법을 통해, 성능 분석 시간을 줄일 수 있는 버스 시스템 성능 분석 방법에 관한 것이다.
- [0028] 이하, 첨부된 도면을 참조하여 본 발명의 바람직한 실시 예들에 대하여 설명하기로 한다.
- [0029] 도면을 참조한 설명에 앞서, 본 발명의 실시 예들에 따른 버스 시스템 성능 분석 방법은 단일 버스 시스템 및 다중 버스 시스템에 모두 적용이 가능하며 이하, 설명에서는 먼저, 단일 버스 시스템에서의 성능 분석 방법에 대하여 설명하고 이어서, 다중 버스 시스템에서의 성능 분석 방법에 대하여 설명하기로 한다.
- [0030] 일반적인 단일 버스 시스템은 도 1에 도시된 바와 같이, 하나의 버스(30)에 접속된 복수의 마스터(20, 22)와, 복수의 슬레이브(40, 42)와, 상기 마스터의 버스 이용을 중재하는 중재기(50)와, 디코더(미도시)로 구성되어 있

다.

- [0031] 이러한 단일 버스 시스템은 일반적으로 도 4에 도시된 바와 같은 절차를 통해 마스터와 슬레이브 간에 데이터의 송수신이 이루어지게 된다.
- [0032] 도 4를 참조하면, 먼저, 복수의 마스터(20, 22) 중에서 어느 하나의 마스터 예를 들면, 제 1 마스터(20)가 버스(30)를 이용하여 복수의 슬레이브(40, 42) 중에서 어느 하나의 슬레이브 예를 들면, 제 2 슬레이브(42)로 접속하고자 하는 경우에 제 1 마스터(20)는 중재기(50)로 버스(30) 이용을 요청하게 된다(a).
- [0033] 중재기(50)는 제 1 마스터(20)로부터 버스(30) 이용 요청이 입력되면, 버스(30)의 이용 가능 여부 및 제 1 마스터가 접속하고자 하는 슬레이브로의 접속 가능여부를 확인한다. 이후, 중재기(50)는 제 1 마스터(20)가 버스(40)를 이용하여 제 2 슬레이브로(42)의 접속이 가능한지의 여부를 Grant 신호로 알려주게 된다(b).
- [0034] 중재기(50)로부터 버스(30) 이용 허가를 획득한 제 1 마스터(20)는 버스(30)를 통해 제 2 슬레이브(42)로 접속하여 데이터를 전송 또는 로딩하게 된다(c). 이때, 제 2 슬레이브(42)는 제 1 마스터(20)로부터 요청된 처리를 수행할 수 없는 경우, 제 1 마스터(20)로 대기 신호를 전송하여 앞서 수행중인 동작이 완료될 때까지 제 1 마스터(20)로부터 요청 수행을 대기시킬 수 있다(d).
- [0035] 제 1 마스터(20)와 제 2 슬레이브(42) 간에 데이터 송수신이 완료되면, 제 2 슬레이브(42)는 중재기(50)로 제 1 마스터(20)의 요청 수행이 완료되었음을 통보하게 된다(e). 상술한, (a) 내지 (e)의 절차들이 완료되면 버스 시스템은 다른 마스터가 데이터를 송수신 할 수 있는 초기 상태로 돌아가게 된다.
- [0036] 도 5는 본 발명의 제 1 실시 예에 따른 버스 시스템 성능 분석 방법을 나타내는 도면이다.
- [0037] 도 5를 참조하면, 버스 시스템은 상술한 바와 같은 절차를 통해 마스터와 슬레이브 간에 데이터의 송수신이 이루어지게 되는데, 버스 시스템에서 수행되는 업무 처리를 데이터 송수신의 동작에 기초하여 분석하면 다음과 같이, 초기 상태(initial state, 100), 전송 준비 상태(transfer generation state, 200), 버스 중재 상태(arbitration state, 300), 전송 수행 상태(transfer execute state, 400)의 4개의 상태로 분류할 수 있다.
- [0038] 먼저, 초기 상태(100)는 마스터를 통한 버스 이용 요청이 발생하지 않아 버스의 동작이 없는 상태를 의미한다. 즉, 복수의 마스터 중 하나의 마스터가 슬레이브로의 접속이 가능한 상태를 의미한다. 이러한, 초기 상태(100) 이후, 전송 준비 상태(200)로 진입하게 된다.
- [0039] 전송 준비 상태(200)는 복수의 마스터 중 적어도 하나의 마스터에서 버스를 사용하고자 하는 경우의 상태를 의미한다. 이러한, 전송 준비 상태(200)에서는 버스 이용 요청과 관련된 신호(HBUSREQ, HADDR)와, 데이터 송수신과 관련된 신호들(HADATA, HBURST, HWRITE, HSIZE, HPROT, HTRANS, HLOCK, HSEL_S 등)이 발생되게 된다. 이러한, 전송 준비 상태(200) 이후, 버스 중재 상태(300)로 진입하게 된다.
- [0040] 버스 중재 상태(300)는 고정된 우선 순위 판단 방법, 라운드 로빈 우선 순위 판단 방법, TDMA 방법, 로터리(lottery) 방법과 같은 여러 가지 버스 중재 방법을 이용하여 복수의 마스터 중에서 하나의 마스터에게 버스 이용권(Grant 신호)을 할당하는 상태를 의미한다. 이러한, 버스 중재 상태(300) 이후, 전송 수행 상태(400)로 진입하게 된다.
- [0041] 전송 수행 상태(400)는 실제로 데이터의 송수신 처리(transaction)가 이루어지는 상태로, 버스 중재 상태(300)에서 버스 이용권을 할당 받은 마스터가 슬레이브에 접속하여 데이터를 전송하거나, 슬레이브로부터 데이터를 로딩하게 된다. 전송 수행 상태(400)는 마스터에서 요청된 데이터의 송수신이 완료될 때까지 유지된다.
- [0042] 이러한, 전송 수행 상태(400) 이후, 다른 마스터에서 버스 이용의 요청이 있어, 상기 다른 마스터가 대기 중인 경우에는 버스 중재 상태(300)로 진입하게 된다. 또한, 다른 마스터가 버스를 이용하려고 하는 경우에는 전송 준비 상태(200)로 진입하게 된다. 그리고, 버스 시스템 내의 모든 마스터가 버스를 이용하지 않는 경우에는 초기 상태(100)로 진입하게 된다.
- [0043] 상술한 바와 같이, 버스 시스템에서 수행되는 업무 처리를 데이터 송수신의 동작에 기초하여 분석된 4개의 상태(100, 200, 300, 400)에 따라 데이터의 송수신이 수행되는 과정을 살펴보면, 총 9가지의 케이스(Case1 내지 Case9)가 존재하게 된다.
- [0044] 이러한, 버스 시스템에서 수행되는 데이터 송수신 과정의 9가지의 케이스(Case1 내지 Case9)와 각 케이스를 정의하는 조건은 다음의 표 1에 나타내었다.

표 1

Case	조 건
Case1	If [idle_cycle (all master)] > 0
Case2	If [idle_cycle (some master)] = 0
Case3	Always
Case4	During arbitration cycle
Case5	After arbitration cycle
Case6	During transaction cycle
Case7	If [Request (some masters)] = True
Case8	If [Request (all masters)] = False & [idle_cycle (some masters)] = 0
Case9	If [idle_cycle (all masters)] != 0

[0045]

[0046]

상기, 도 5 및 표 1을 참조하면, 초기 상태(100)에서는 마스터를 통한 버스 이용 요청이 발생하지 않아 버스의 동작이 없는 경우에 마스터의 버스 이용 요청을 대기하는 Case1과, 마스터로부터 버스 이용 요청이 이루어져 전송 준비 상태(200)로 진입하는 Case2가 있다.

[0047]

여기서, Case1은 모든 마스터의 아이들 사이클(cycle)이 0 보다 큰 경우 ([idle_cycle (all master)] > 0)에 해당하는 것으로, 모든 마스터의 아이들 사이클이 0 보다 크다는 것은 모든 마스터들에게서 버스 이용 요청이 발생되지 않음을 의미한다. 이러한 경우에는 초기 상태(100)를 유지하게 된다.

[0048]

Case2는 모든 마스터 중에서 어느 하나의 마스터의 아이들 사이클이 0인 경우 ([idle_cycle (some master)] = 0)에 해당하는 것으로, 상기 어느 하나의 마스터의 아이들 사이클이 0이라는 것은 마스터에게서 버스 이용 요청이 발생되었음을 의미한다. 이러한 경우에는 초기 상태(100)에서 전송 준비 상태(200)로 진입하게 된다.

[0049]

전송 준비 상태(200)에서는 버스 이용 요청이 이루어진 마스터에 대하여 상기 버스 이용 요청과 관련된 신호 및 상기 데이터 송수신과 관련된 신호들이 발생된다. 따라서, 특별한 조건 없이 Case3이 적용되어, 버스 중재 상태(300)로 진입하게 된다.

[0050]

버스 중재 상태(300)에서는 Case4와, Case5가 존재하게 되는데, Case4는 버스에서 발생하는 버스 중재 사이클(arbitration_cycle)을 정의하는 것으로, 버스 중재 사이클 기간(During arbitration cycle) 동안 버스 중재 상태(300)를 유지한 후, 버스 중재 사이클 기간이 완료되면 Case5를 진행하여 전송 수행 상태(400)로 진입하게 된다.

[0051]

전송 수행 상태(400)에서는 Case6 내지 Case9가 존재하게 되는데, Case6은 버스를 이용하여 마스터와 슬레이브 간에 데이터의 송수신이 실제로 이루어지는 기간(During transaction cycle)을 정의하는 것으로, 마스터 및 슬레이브의 대기 시간(latency)이 고려되어 데이터의 송수신 처리(transaction)가 이루어진다.

[0052]

Case7은 복수의 마스터들 중에서 어느 하나의 마스터에서 버스 이용의 요청이 있어 ([Request (some masters)] = True), 상기 어느 하나의 마스터가 대기 중인 경우에는 해당하는 것으로, 이러한 경우에는 버스 중재 상태(300)로 진입하게 된다.

[0053]

Case8은 버스 시스템 내의 모든 마스터들에게서 버스 이용 요청이 없고([Request (all masters)] = False), 복수의 마스터들 중에서 어느 하나의 마스터의 아이들 사이클이 0인([idle_cycle (some masters)] = 0) 경우에 해당하는 것으로, 복수의 마스터들 중에서 어느 하나의 마스터가 버스를 이용하려고 하는 경우에는 전송 준비 상태(200)로 진입하게 된다.

[0054]

Case9는 버스 시스템 내의 모든 마스터들의 아이들 사이클이 0이 아닌 경우([idle_cycle (all masters)] != 0) 즉, 상기 아이들 사이클이 0보다 큰 경우에 해당하는 것으로, 버스 시스템 내의 모든 마스터가 버스를 이용하지

않으면 초기 상태(100)로 진입하게 된다.

- [0055] 도 6은 본 발명의 제 1 실시 예에 따른 버스 시스템 성능 분석 방법의 알고리즘을 나타내는 도면이다.
- [0056] 도 6에 도시된 바와 같이, 버스 시스템에서 수행되는 업무 처리를 데이터 송수신의 동작에 기초하여 분석된 4가지 상태(100, 200, 300, 400)에 따른 수행과, 데이터 송수신 과정의 9가지의 케이스(Case1 내지 Case9)의 조건을 함수로 정의하고, 이를 TLM(Transaction Level Model) 방식을 통해 알고리즘으로 구현하면 RTL 방식을 이용한 종래 기술의 버스 시스템 성능 분석 방법과 같이, 실제 로직을 설계 및 구현하지 않고도 버스 시스템의 성능 분석을 수행할 수 있다.
- [0057] 도 6에 도시된, 알고리즘에서 "InitSt"는 초기 상태(100)를 의미하며, "TransferGenSt"는 전송 준비 상태(200)를 의미하며, "ArbitrationSt"는 버스 중재 상태(300)를 의미하며, "TransferExecSt"는 전송 수행 상태(400)를 의미한다. 또한, "Cur_Cycle"는 사이클의 카운트 값을 의미하는 것으로, 설정된 최대 값까지 증가하거나, 0까지 감소하게 된다.
- [0058] "Detail_Cycles_Cal(Master, Slave, State)"은 버스 트랜잭션 사이클, 버스 요청 사이클, 아이들 사이클 등 데이터의 송수신이 소요되는 사이클의 계산이 정의된 함수이다. "Execute_Bus_Model(Master, Slave, State)"은 각각의 상태에 따른 버스의 동작 수행을 정의하는 함수로 데이터의 송수신을 수행한다.
- [0059] 이러한, TLM 방식을 적용한 본 발명의 제 1 실시 예에 따른 버스 시스템 성능 분석 방법은 실제 로직을 설계 및 구현하지 않고 버스 시스템의 성능 분석을 수행할 수 있다. 본 발명의 제 1 실시 예에 따른 버스 시스템 성능 분석 방법은 버스 시스템에서 수행되는 업무 처리를 데이터 송수신의 동작에 기초하여 분석된 4가지 상태(100, 200, 300, 400)에 따른 수행과, 데이터 송수신 과정의 9가지 케이스(Case1 내지 Case9)의 조건을 함수로 정의한 후, 상기 사항들을 알고리즘으로 구현하여 버스 시스템의 성능 분석의 정확도 향상 및 성능 분석 시간을 줄일 수 있다.
- [0060] 단일 버스 시스템은 동일 시간에 하나의 마스터 만이 데이터를 송수신 할 수 있으므로, 기본적으로 데이터의 병렬 전송이 불가능하다. 이러한, 단일 버스 시스템의 문제점을 개선하고자, 도 3에 도시된 바와 같이, 복수의 버스 및 브리지를 구비하여 동일 시간에 데이터를 병렬로 송수신 할 수 있는 다중 버스 시스템이 제안되고 있다.
- [0061] 도 3에 도시된 다중 버스 시스템에서, 버스는 고속 데이터 송신이 가능한 AHB(Advanced High-performance Bus)를 하나의 예로 하였으며, AHB 외에도 ASB(Advanced System Bus), APB(Advanced peripheral Bus)가 적용될 수 있다.
- [0062] 다중 버스 시스템의 데이터 송수신은 기본적으로 단일 버스 시스템과 동일하며, 서로 다른 버스에 접속되어 있는 마스터와 슬레이브 간에 데이터의 송수신은 제 1 버스와 제 2 버스를 연결하는 브리지를 통해 이루어지므로, 이러한 다중 버스 시스템의 성능 분석을 위해서는 브리지에서 수행되는 동작들이 고려되어야 한다.
- [0063] 이어서, 도 7은 본 발명의 제 2 실시 예에 따른 버스 시스템 성능 분석 방법을 나타내는 도면이고, 도 8 내지 도 9b는 본 발명의 제 2 실시 예에 따른 버스 시스템 성능 분석 방법의 알고리즘을 나타내는 도면이다.
- [0064] 이하, 도 7 내지 도 9b를 참조하여 본 발명의 제 2 실시 예에 따른 버스 시스템 성능 분석 방법에 관하여 설명하기로 한다.
- [0065] 도 7에 도시된 버스 시스템 성능 분석 방법은 복수의 버스를 포함하는 다중 버스 시스템의 성능 분석을 위한 것으로, 브리지에서 수행되는 동작을 고려한 제 1 및 제 2 브리지 상태(500a, 500b)를 제외하고는 상기 도 5를 참조하여 상술한 본 발명의 제 1 실시 예에 따른 버스 시스템 성능 분석 방법과 동일하다. 이에 따라, 제 1 및 제 2 브리지 상태(500a, 500b)에 대하여 상세히 설명하고, 다른 상태 및 케이스들에 대한 설명은 간략하기로 한다.
- [0066] 버스1(Share Bus1)과 버스2(Share Bus2)는 개별적인 데이터 처리가 이루어지며, 각 버스에서 수행되는 데이터의 송수신에 관한 사항 및 성능 분석 방법은 도 5를 참조하여 설명한 본 발명의 제 1 실시 예와 동일하다.
- [0067] 버스1(Share Bus1)과 버스2(Share Bus2) 각각에서 수행되는 업무 처리를 데이터 송수신의 동작에 기초하여 분석하면 상술한 바와 같이, 초기 상태(100a, 100b), 전송 준비 상태(200a, 200b), 버스 중재 상태(300a, 300b) 및 전송 수행 상태(400a, 400b)를 가진다.
- [0068] 다중 버스 시스템에서의 성능 분석을 위해서는 브리지의 동작 특성을 반영하여야 한다. 이를 위해, 본 발명의 제 2 실시 예에 따른 버스 시스템 성능 분석 방법에서는 브리지에서 수행되는 동작을 제 1 브리지 상태(500a) 및 제 2 브리지 상태(500b)로 분류하고, 각 상태에 따른 조건을 정의하였다. 그리고, 각 상태에 따른 조건을 알

고리즘으로 구현하여 다중 버스 시스템의 성능 분석이 가능토록 하였다.

- [0069] 버스1(Share Bus1)에 접속된 마스터들 중 어느 하나의 마스터가 버스2(Share Bus2)에 접속된 슬레이브들 중 어느 하나의 슬레이브로 접속하기 위해서는 브리지를 경유하게 된다. 브리지는 서로 다른 버스 간에 데이터의 송수신이 이루어질 수 있도록 인터페이스를 지원하며, 버스1(Share Bus1)과 버스2(Share Bus2)에서 수행되는 동작에 따라 서로 다른 버스 간의 접속을 제어하게 된다.
- [0070] 버스1(Share Bus1)과 버스2(Share Bus2) 각각은 도 8에 도시된 바와 같이, 최종 사이클(Final_Cycle)이 진행될 때까지 bus1_execute(Bus1_Arbitration_Type), bus2_execute(Bus2_Arbitration_Type) 함수를 반복하여 수행하게 된다.
- [0071] 도 7 내지 도 9b에서는 다중 버스 시스템이 2개의 버스를 포함하는 것으로 기재하고 있으나, 이는 하나의 실시예이며 3개 이상의 버스를 포함하여 다중 버스 시스템을 구현할 수 있다. 예를 들어 4개의 버스를 포함하는 다중 버스 시스템의 성능 분석이 요구되는 경우에는 도 8에 도시된 알고리즘에서 bus3_execute 및 bus4_execute 함수를 추가하고, Multiple_bus_model 함수에 Bus3_arbitration_Type 및 Bus4_arbitration_Type을 추가하여, 추가된 버스의 중재방법을 정의할 수 있다. 이를 통해, 4개의 버스를 포함하는 다중 버스 시스템의 성능을 분석할 수 있다.
- [0072] 도 9a 및 도 9b에서는 다중 버스 시스템을 구성하는 복수의 버스 중에서 버스1에 대한 알고리즘을 나타내고 있으며, 버스2, 버스3 및 버스4의 경우에도 상기 버스1의 알고리즘과 동일하게 구현할 수 있다.
- [0073] 초기 상태(100a, 100b)에서는 모든 마스터로부터 데이터 전송 요청이 발생되지 않아 버스를 이용 요청을 대기하는 구간이며, 이때 사이클(Cur_Cycle)은 계속 증가하게 된다.
- [0074] 임의의 사이클에서 어느 하나의 마스터로부터 버스를 이용한 데이터의 송수신 요청이 발생되면 전송 준비 상태(200a, 200b)로 진입하게 된다. 이때, Detail_Cycles_Cal(Master, Slave, State) 함수에 정의된 사항들에 따라서 버스 트랜잭션 사이클, 버스 요청 사이클, 아이들 사이클 등 데이터 송수신을 위해 소요되는 사이클이 계산되게 된다.
- [0075] 버스1의 경우를 살펴보면, 제 1 전송 준비 상태(200a, Transfer Generation St1)에서 데이터전송에 필요한 어드레스, 데이터, 전송신호 등이 생성되며, 이때, 사이클(Cur_Cycle)은 증가되지 않는다. 이후, 제 1 전송 준비 상태(200a)에 해당하는 케이스를 만족하면 제 1 버스 중재 상태(300a, Arbitration St1)로 진입하게 된다.
- [0076] 제 1 버스 중재 상태(300a)에서는 복수의 마스터 중 어느 하나의 마스터 또는 여러 마스터의 데이터전송 요청 중에서 우선순위에 따라 마스터를 선택하게 된다. 이때, Arbitration(request_master, arbitration_type) 함수에 정의된 사항에 따라 복수의 마스터 중에서 버스를 이용하여 데이터를 송수신할 마스터를 선택하게 된다.
- [0077] 이후, 버스중재 사이클(ARBITRATION_CYCLE) 만큼 시간 지연 후, 데이터 전송이 이루어지는데, 만약 접속하고자 하는 슬레이브가 다른 버스에 접속된 슬레이브에 접속하고자 하면 브리지를 경유해야 한다. 예를 들면, 도 3에 도시된, 버스1에 접속된 마스터1이 버스2에 접속된 슬레이브3에 접속하여 데이터를 송수신하고자 하면 브리지를 경유하게 된다. 이때에는 제 1 버스 중재 상태(300a)에서 제 1 브리지 상태(500a, Bridge St1)로 진입하게 된다.
- [0078] 이후, 브리지는 마스터1이 버스2를 이용할 수 있는지 확인하고, 슬레이브3에 접속이 가능한지를 확인하게 된다. 확인결과, 적어도 하나의 다른 마스터가 버스2를 사용하고 있지 않아 버스2의 이용이 가능하고, 슬레이브3에서 다른 동작의 수행이 없으면 마스터1은 슬레이브3에 접속한 후, 데이터의 송수신을 수행할 수 있게 된다. 이를 위해, 제 1 버스 중재 상태(300a)에서 제 1 전송 수행 상태(400a, Transfer Execute St1)로 진입하게 된다.
- [0079] 여기서, 제 1 전송 수행 상태(400a)는 데이터 전송이 실제로 이루어지는 구간으로서, 슬레이브3 대기시간(latency) 고려되어 데이터의 송수신이 수행된다.
- [0080] 제 1 전송 수행 상태(400a)에서 마스터1이 버스1을 이용하여 데이터를 전송하면, 제 2 브리지 상태(500b, Bridge St2)로 진입하여, 버스1을 경유하여 전송되는 데이터를 버스2로 전송하여 마스터1로부터의 데이터를 슬레이브3으로 전송하게 된다.
- [0081] 상술에서는 버스1에 접속된 마스터1에서 버스2에 접속된 슬레이브3으로 데이터를 전송하는 경우를 하나의 예로서 설명하였다. 다른 실시 예로서, 상기 슬레이브3에서 마스터1로 데이터가 전송되는 경우에는 제 2 버스 중재 상태(300b), 제 2 전송 수행 상태(400b), 제 1 브리지 상태(500a, Bridge St1), 제 2 브리지 상태(500b,

Bridge St2) 순으로 진행하여 데이터의 송수신이 이루어지게 된다. 이때, 각 상태의 정의 및 케이스들의 조건은 상술한 사항들과 동일하다.

[0082] 도 10은 본 발명의 제 1 및 제 2 실시 예에 따른 버스 시스템 성능 분석 방법의 시뮬레이션 결과를 나타내는 것으로, 알고리즘의 구현은 마이크로 소프트(社)의 비주얼(Visual) C++을 이용하였다.

[0083] 도 10에 도시된 바와 같이, 본 발명의 실시 예들에 따른 버스 시스템 성능 분석 방법은 버스 시스템에서 수행되는 업무 처리를 데이터 송수신의 동작에 기초하여 분석된 4가지 상태에 따른 수행과, 데이터 송수신 과정의 9가지의 케이스(Case1 내지 Case9)의 조건을 함수로 정의하고, 이를 TLM(Transaction Level Model) 방식을 통해 알고리즘으로 구현함으로써, 실제 로직을 설계 및 구현하지 않고도 버스 시스템의 성능 분석을 수행할 수 있다.

[0084] 상술한, 본 발명의 실시 예에 따른 버스 시스템 성능 분석 방법의 시뮬레이션 결과를 종래 기술과 대비하여 다음의 표 2에 나타내었다.

표 2

성능 분석 방법	동작 실행 횟수 [cycle/sec]	비고
Real chip clock	133M/sec	·
본 발명 TLM(Transaction Level Model)	18K/sec	종래기술 대비 160배 속도 향상
종래기술 Verilog RTL simulation	112/sec	

[0085]

[0086] 상기 표 2의 결과는 900Mhz CPU, 3Gbyte Memory 사양의 Sun Blade 2000 워크스테이션을 이용하여 버스 시스템 성능 분석 방법의 시뮬레이션을 수행하여 얻은 것으로, 133Mhz의 클럭으로 총 10,000,000번의 사이클을 동작시켜 1초당 처리가 이루어진 사이클의 횟수를 비교하여 나타내었다.

[0087] 본 발명과 종래 기술의 버스 시스템 성능 분석 시간을 비교하였을 때, 상기 표 2에 나타난 바와 같이, 종래 기술에 따른 버스 시스템 성능 분석 방법은 1초당 112 사이클을 수행한 반면, 본 발명의 실시 예에 따른 버스 시스템 성능 분석 방법은 1초당 18,000 사이클을 수행하여, 종래 기술대비 버스 시스템의 성능 분석 시간을 단축시킬 수 있음을 확인할 수 있다.

[0088] 이러한, TLM 방식을 적용한 본 발명의 실시 예들에 따른 버스 시스템 성능 분석 방법은 실제 로직을 설계 및 구현하지 않고 버스 시스템의 성능 분석을 수행함으로써, 버스 시스템의 성능 분석의 정확도 향상 및 성능 분석 시간을 줄일 수 있다. 또한, 빠르게 변화하는 사용자의 SoC 개발 요구 사항에 신속히 대응할 수 있다.

[0089] 상술한 본 발명의 실시 예들에 따른 버스 시스템 성능 분석 방법은 다양한 컴퓨터 수단을 통하여 수행될 수 있는 프로그램 명령 형태로 구현되어 컴퓨터로 판독 가능한 기록 매체에 기록될 수 있다. 이때, 컴퓨터로 판독 가능한 기록매체는 프로그램 명령, 데이터 파일, 데이터 구조 등을 단독으로 또는 조합하여 포함할 수 있다. 한편, 기록매체에 기록되는 프로그램 명령은 본 발명을 위하여 특별히 설계되고 구성된 것들이거나 컴퓨터 소프트웨어 당업자에게 공지되어 사용 가능한 것일 수도 있다.

[0090] 컴퓨터로 판독 가능한 기록매체에는 하드 디스크, 플로피 디스크 및 자기 테이프와 같은 자기 매체(Magnetic Media), CD-ROM, DVD와 같은 광 기록 매체(Optical Media), 플롭티컬 디스크(Floptical Disk)와 같은 자기-광 매체(Magneto-Optical Media), 및 롬(ROM), 램(RAM), 플래시 메모리 등과 같은 프로그램 명령을 저장하고 수행하도록 특별히 구성된 하드웨어 장치가 포함된다. 한편, 이러한 기록매체는 프로그램 명령, 데이터 구조 등을 지정하는 신호를 전송하는 반송파를 포함하는 광 또는 금속선, 도파관 등의 전송 매체일 수도 있다.

[0091] 또한, 프로그램 명령에는 컴파일러에 의해 만들어지는 것과 같은 기계어 코드뿐만 아니라 인터프리터 등을 사용해서 컴퓨터에 의해서 실행될 수 있는 고급 언어 코드를 포함한다. 상술한 하드웨어 장치는 본 발명의 동작을 수행하기 위해 하나 이상의 소프트웨어 모듈로서 작동하도록 구성될 수 있으며, 그 역도 마찬가지이다.

[0092] 한편, 본 발명이 속하는 기술분야의 당업자는 상술한 본 발명이 그 기술적 사상이나 필수적 특징을 변경하지 않고서 다른 구체적인 형태로 실시될 수 있다는 것을 이해할 수 있음은 당 업자에게 자명하다.

[0093] 그러므로, 이상에서 기술한 실시예들은 모든 면에서 예시적인 것이며 한정적인 것이 아닌 것으로 이해해야만 한

다. 본 발명의 범위는 상기 상세한 설명보다는 후술하는 특허청구범위에 의하여 나타내어지며, 특허청구범위의 의미 및 범위 그리고 그 등가 개념으로부터 도출되는 모든 변경 또는 변형된 형태가 본 발명의 범위에 포함되는 것으로 해석되어야 한다.

도면의 간단한 설명

도 1은 일반적인 단일 버스 시스템을 나타내는 도면.

도 2는 종래 기술에 따른 RTL(Register Transfer Level) 방식을 이용한 버스 시스템 성능 분석 방법을 나타내는 도면.

도 3은 일반적인 다중 버스 시스템을 나타내는 도면.

도 4는 단일 버스 시스템의 데이터 전송 방법을 나타내는 도면.

도 5는 본 발명의 제 1 실시 예에 따른 버스 시스템 성능 분석 방법을 나타내는 도면.

도 6은 본 발명의 제 1 실시 예에 따른 버스 시스템 성능 분석 방법의 알고리즘을 나타내는 도면.

도 7은 본 발명의 제 2 실시 예에 따른 버스 시스템 성능 분석 방법을 나타내는 도면.

도 8 내지 도 9b는 본 발명의 제 2 실시 예에 따른 버스 시스템 성능 분석 방법의 알고리즘을 나타내는 도면.

도 10은 본 발명의 제 1 및 제 2 실시 예에 따른 버스 시스템 성능 분석 방법의 시뮬레이션 결과를 나타내는 도면.

<도면 주요 부분에 대한 부호의 설명>

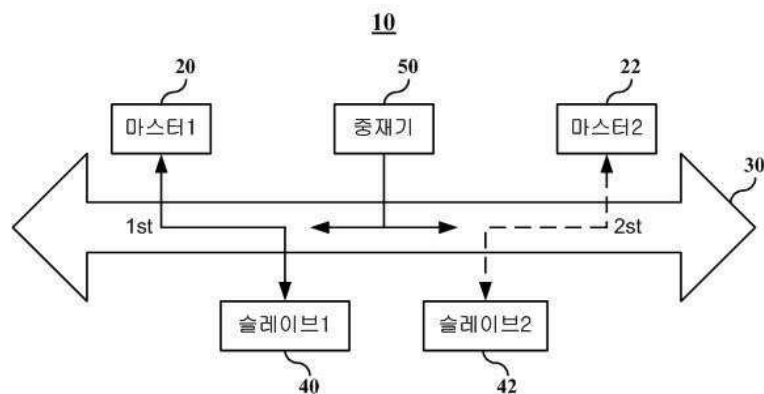
100: 초기 상태 200: 전송 준비 상태

300: 버스 중재 상태 400: 전송 수행 상태

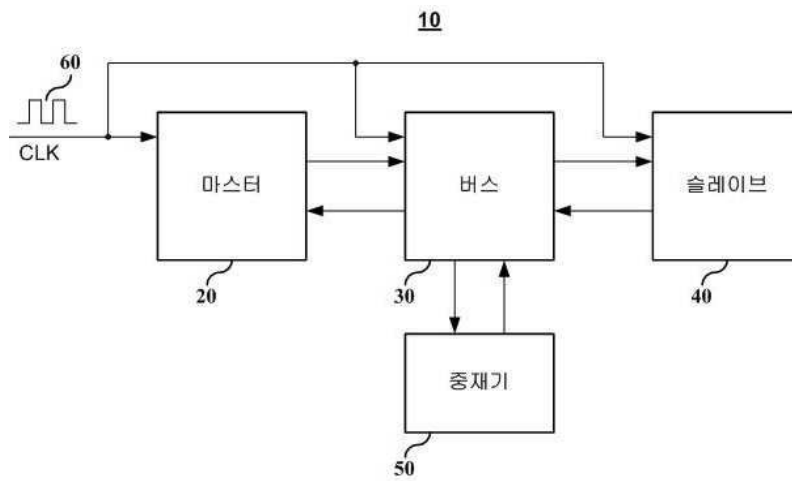
500a, 500b: 브리지 상태

도면

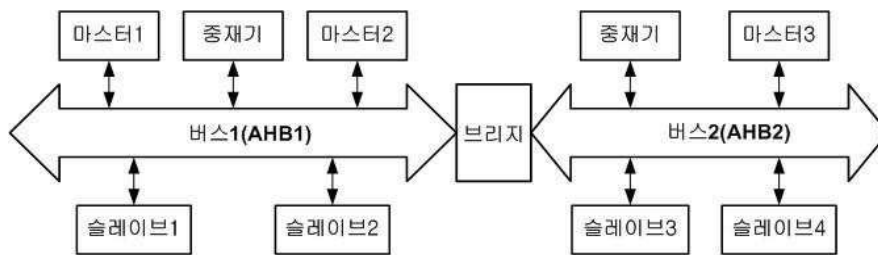
도면1



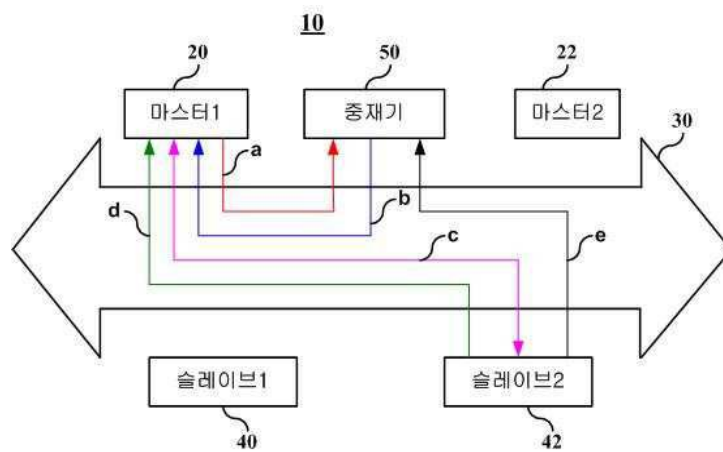
도면2



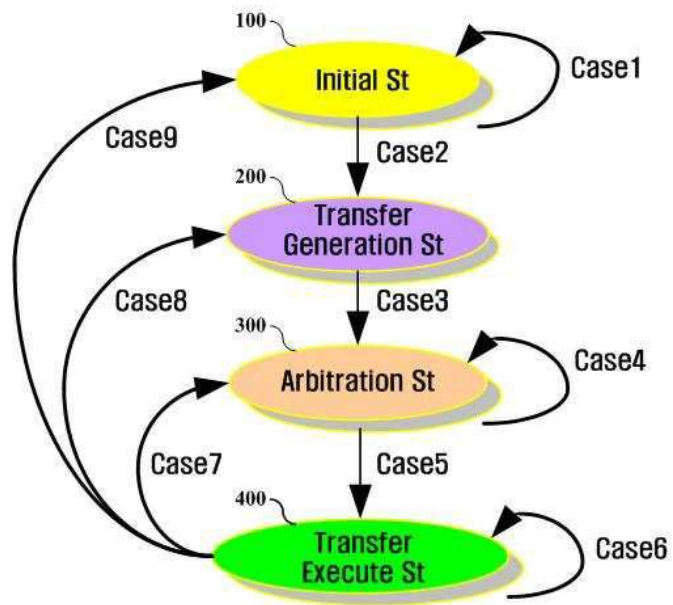
도면3



도면4



도면5



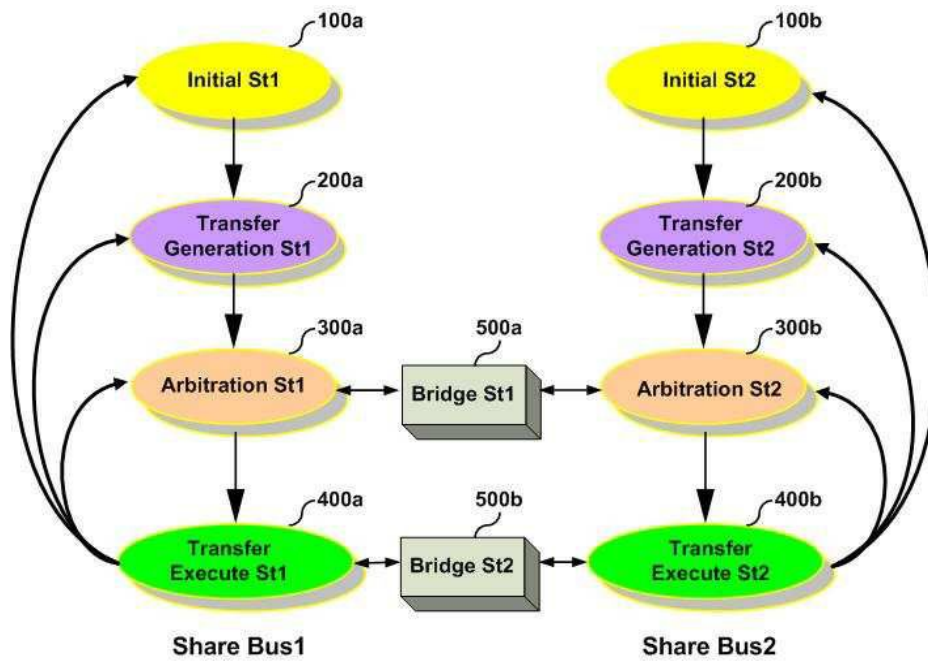
도면6

```

1: single_bus_model(Arbitration_Type)
2: begin
3:   Cur_Cycle=0
4:   State=InitSt
5:   Master_Signal[all masters]=Idle
6:   Slave_Signal[all slaves]=Idle
7:   while (Cur_Cycle<Final_Cycle) do
8:     if (State==InitSt) then
9:       Idle_Gen(Master_Signal[selected masters])
10:      while(Master_Signal.Idle_Cycle
          [all masters]--) do
11:        Cur_Cycle++
12:        Detail_Cycles_Cal(NULL,NULL,InitSt)
13:      end while
14:      Execute_Bus_Model(Master_Signal[all
          masters],NULL,InitSt)
15:      State=ReqSt
16:      end if
17:      else if (State==TransferGenSt) then
18:        Req_Gen(Master_Signal[selected masters])
19:        Addr_Gen(Master_Signal[selected masters])
20:        Data_Gen(Master_Signal[selected masters])
21:        Transfer_Signal_Gen(Master_Signal[selected
          masters])
22:        State=ArbitrationSt
23:      end else if
24:        else if (State==ArbitrationSt) then
25:          Arbitration(Req[selected masters],
            Arbitration_Type)
26:          while (ARBITRATION_CYCLE--) do
27:            Cur_Cycle++
28:            Detail_Cycles_Cal(NULL,NULL,ArbitrationSt)
29:            Execute_Bus_Model(Master_Signal[all masters],
            NULL,ArbitrationSt)
30:          end while
31:          State=TransferExecSt
32:        end else if
33:          else then //State is TransferExecSt.
34:            while ((Master_Signal.Data_Size[selected master]
              +Slave_Signal.Slave_Latency[selected slave])--) do
35:              Cur_Cycle++
36:              Detail_Cycles_Cal(Master_Signal[all masters].Slave_
              Signal[all slaves],TransferExecSt)
37:              Execute_Bus_Model(Master_Signal[all masters].Slave_
              Signal[all slaves],TransferExecStSt)
38:            end while
39:            Master_Signal.Req[selected master]=False
40:            if (Master_Signal.Req[some masters]==True)
              State=ArbitrationSt
41:            else if (Master_Signal.Idle_Cycle[some masters]==0)
              State=TransferGenSt
42:            else State=InitSt
43:            end else
44:            end while
45:            end single_bus_model(Arbitration_Type)

```

도면7



도면8

```

1: multiple_bus_model(Bus1_Arbitration_
    Type,Bus2_Arbitration_Type)
2: begin
3:   Cur_Cycle=0
4:   State=InitSt
5:   Master_Signal[all masters]=Idle
6:   Slave_Signal[all slaves]=Idle
7:   while (Cur_Cycle<Final_Cycle) do
8:     bus1_execute(Bus1_Arbitration_Type)
9:     bus2_execute(Bus2_Arbitration_Type)
10:  .....
10: end while
11: end multiple_bus_model(Bus1_Arbitration_
    Type,Bus2_Arbitration_Type)

```

도면9a

```

1: bus1_execute(Bus1_Arbitration_Type)
2: begin
3: // -----
4: // ----- State is InitSt1. -----
5: // -----
6: if (State==InitSt1) then
7:   Idle_Gen(Master_Signal[selected masters])
8:   if (Master_Signal.Idle_Cycle[all masters]--) then
9:     Cur_Cycle++
10:    Detail_Cycles_Cal(NULL,NULL,InitSt)
11:  end if
12: else then
13:   Execute_Bus_Model(Master_Signal[all masters],
14:     NULL,InitSt)
15:   State=TransferGenSt1
16: end else
17: // -----
18: // ----- State is TransferGenSt1. -----
19: // -----

20: else if (State==TransferGenSt1) then
21:   Req_Gen(Master_Signal[selected masters])
22:   Addr_Gen(Master_Signal[selected masters])
23:   Data_Gen(Master_Signal[selected masters])
24:   Transfer_Signal_Gen(Master_Signal
25:     [selected masters])
26:   State=ArbitrationSt1
27: end else if
28: // ----- State is ArbitrationSt1. -----
29: // -----
30: else if (State==ArbitrationSt1) then
31:   Arbitration(Req[selected masters],
32:     Arbitration_Type)
33:   if (ARBITRATION_CYCLE!=0) then
34:     Cur_Cycle++
35:     Detail_Cycles_Cal(NULL,NULL,ArbitrationSt1)
36:     Execute_Bus_Model(Master_Signal[all masters],
37:       NULL,ArbitrationSt1)
38:   end if

```

도면9b

```

38: else if (bus1.transaction_bus==1) then
39:   if (TransferExecSt2 & (bus2.transaction_
      bus==1)) then
40:     Cur_Cycle++
41:     Detail_Cycles_Cal(NULL,NULL,ArbitrationSt1)
42:     Execute_Bus_Model(Master_Signal[all masters],
      NULL,ArbitrationSt1)
43:   end if
44: else then
45:   State=TransferExecSt1
46: end else
47: end else if //else if (bus1.transaction_
      bus==1) then
48: else if (bus1.transaction_bus==2) then
49:   if (TransferExecSt2 & (bus2.transaction_
      bus==2)) then //State is BridgeSt1.
50:     Cur_Cycle++
51:     Detail_Cycles_Cal(NULL,NULL,ArbitrationSt1)
52:     Execute_Bus_Model(Master_Signal[all masters],
      NULL,ArbitrationSt1)
53:   end if
54: else then
55:   State=TransferExecSt1
56: end if

57: end else if //else if (bus1.transaction_
      bus==2) then
58: end else if //else if (State==ArbitrationSt) then
59: // -----
60: // ----- State is TransferExecSt1. -----
61: // -----
62: else if (State==TransferExecSt1) then
63:   if ((Master_Signal.Data_Size[selected master]
      +Slave_Signal.
64:     Slave_Latency[selected slave])-->) then
65:     Cur_Cycle++
66:     Detail_Cycles_Cal(Master_Signal[all masters],
      Slave_Signal[all slaves],TransferExecSt1)
67:     Execute_Bus_Model(Master_Signal[all masters],
      Slave_Signal[all slaves],TransferExecSt1)
68:   end if
69:   Master_Signal.Req[selected master]=False
70:   if (Master_Signal.Req[some masters]==True)
      State=ArbitrationSt1
71: else if (Master_Signal.Idle_Cycle[some masters]
      ==0) State=TransferGenSt1
72: else State=InitSt1
73: end else
74: end bus_execute(Bus1_Arbitration_Type)

```

도면10

```

void arm_wait_gen      (int Master);
void ethernet_wait_gen (int Master);
FILE *output;
// =====
//                      MAIN MODULE
// =====
int main(int argc, char* argv[])
{
    output=fopen("C:\\\\report.txt", "wt");

    //initialize();

    // =====
    // TEST SCENARIO
    // =====
    // Normal Case (Fixed Priority)
    fprintf(output, "*****FIXED_PRIORITY*****");
    test_scenario(FIXED_PRIORITY,RANDOM_CASE,NULL,NULL);
    //test_scenario(FIXED_PRIORITY,ETHERNET_CASE,INCR4,WRITE

    //fprintf(output, "*****FIXED_PRIORITY_WITH_WAIT*****");
    //test_scenario(FIXED_PRIORITY_WITH_WAIT,RANDOM_CASE,
    //test_scenario(FIXED_PRIORITY_WITH_WAIT,ETHERNET_CASE,INCR4,WRITE

    fprintf(output, "*****ROUND_ROBIN*****");
    test_scenario(ROUND_ROBIN,RANDOM_CASE,NULL,NULL);
    //test_scenario(ROUND_ROBIN,ETHERNET_CASE,INCR4,WRITE

    fprintf(output, "*****FIXED_ROUND*****");
    test_scenario(FIXED_ROUND,RANDOM_CASE,NULL,NULL);
    //test_scenario(FIXED_ROUND,ETHERNET_CASE,INCR4,WRITE

    fprintf(output, "*****TDMA41111*****");
    test_scenario(ROUND_ROBIN1,RANDOM_CASE,NULL,NULL);
    //test_scenario(ROUND_ROBIN1,ETHERNET_CASE,INCR4,WRITE

/*
    fprintf(output, "*****TDMA3111*****");
    test_scenario(ROUND_ROBIN2,RANDOM_CASE,NULL,NULL);
    //test_scenario(ROUND_ROBIN2,ETHERNET_CASE,INCR4,WRITE
}
*****FIXED_PRIORITY*****
Total Idle Cycle: 275, cycle: 1000015
Master0 Total Transaction Cycle: 148510, cycle: 1000015
Master1 Total Transaction Cycle: 135878, cycle: 1000015
Master2 Total Transaction Cycle: 110545, cycle: 1000015
Master3 Total Transaction Cycle: 75041, cycle: 1000015
Master4 Total Transaction Cycle: 31170, cycle: 1000015
Master5 Total Transaction Cycle: 5941, cycle: 1000015
Master0 Total Write Transaction Cycle: 74286, cycle: 1000015
Master1 Total Write Transaction Cycle: 67390, cycle: 1000015
Master2 Total Write Transaction Cycle: 55123, cycle: 1000015
Master3 Total Write Transaction Cycle: 37971, cycle: 1000015
Master4 Total Write Transaction Cycle: 15502, cycle: 1000015
Master5 Total Write Transaction Cycle: 2904, cycle: 1000015
Master0 Total Read Transaction Cycle: 75874, cycle: 1000015
Master1 Total Read Transaction Cycle: 68488, cycle: 1000015
Master2 Total Read Transaction Cycle: 55422, cycle: 1000015
Master3 Total Read Transaction Cycle: 37070, cycle: 1000015
Master4 Total Read Transaction Cycle: 15668, cycle: 1000015
Master5 Total Read Transaction Cycle: 3037, cycle: 1000015
Master0 Total Single Transaction Cycle: 14678, cycle: 1000015
Master1 Total Single Transaction Cycle: 13074, cycle: 1000015
Master2 Total Single Transaction Cycle: 10817, cycle: 1000015
Master3 Total Single Transaction Cycle: 7205, cycle: 1000015
Master4 Total Single Transaction Cycle: 3026, cycle: 1000015
Master5 Total Single Transaction Cycle: 633, cycle: 1000015
Master0 Total Burst Transaction Cycle: 133832, cycle: 1000015
Master1 Total Burst Transaction Cycle: 122804, cycle: 1000015
Master2 Total Burst Transaction Cycle: 99728, cycle: 1000015
Master3 Total Burst Transaction Cycle: 67836, cycle: 1000015
Master4 Total Burst Transaction Cycle: 28144, cycle: 1000015
Master5 Total Burst Transaction Cycle: 5308, cycle: 1000015
Master0 Total Single Transaction Number: 14678, cycle: 1000015
Master1 Total Single Transaction Number: 13074, cycle: 1000015
Master2 Total Single Transaction Number: 10817, cycle: 1000015
Master3 Total Single Transaction Number: 7205, cycle: 1000015
Master4 Total Single Transaction Number: 3026, cycle: 1000015
Master5 Total Single Transaction Number: 633, cycle: 1000015
Master0 Total Burst Transaction Number: 14392, cycle: 1000015
Master1 Total Burst Transaction Number: 13105, cycle: 1000015

```