AMBA 시스템을 통한 주변장치 테스트

김웅*, 정갑천**, 박성모
전남대학교 컴퓨터공학과
전화 : 062-530-0798 / 핸드폰 : 017-617-5261

Peripheral Device Test with AMBA System

Woong Kim, Gab-Cheon Jung, Seong-Mo Park
* Dept. of Computer Eng., Chonnam National Univ.
** Dept. of Electronics Eng., Chonnam National Univ.
E-mail : gobear@hananet.net

Abstract

Recently, AMBA(Advanced Microcontroller Bus Architecture) is used as common system bus at embedded system. In this paper, we described test method of peripheral device which is connected to AMBA according to the bus interface defined by AMBA protocol.
We implemented one of the APB(Advanced Peripheral Bus) peripheral module, GPIO(General Purpose Input/Output), and tested its functionality as it is connected to the AMBA system.

I. 서론

Embedded 시스템에서 버스는 중요한 역할을 담당한다. Macrocell화한 CPU와 모듈화한 IP들을 연결 하여주는 버스 아키텍처의 역할은 시스템의 성능과 효율에 상당한 영향을 미친다. 최근 임베디드 분야에서 가장 많이 이용되고 있는 CPU는 ARM이고, 여기에서 제안한 버스구조가 AMBA(Advanced Microcontroller Bus Architecture)이다.[1] ARM 사의 공개된 버스인 AMBA 는 멀티마스터, 32비트 등으로 SoC설계에 적합하게 구성되어 있다.[2]

본 논문은 한국과학재단지정 전남대학교 고품질 전기전자 부품 및 시스템 연구센터의 연구비 지원에 의해 연구되었음

AMBA는 AHB(Advanced High-performance Bus), ASB(Advanced System Bus), APB(Advanced Peripheral Bus)로 구성되어 있다. AHB는 파이프라인 동작, burst 전달, 단일 클럭 에지 동작 및 non-tristate 구현을 지원함으로 고성능 시스템에 적합한 버스구조이며, APB는 전력 소모를 최소화하고 인터페이스의 복잡도를 줄이기 위해 최적화된 버스 구조이다.[3]

본 논문은 최근 각광받고 있는 임베디드 시스템에서 많이 사용되는 시스템 버스인 AMBA의 프로토콜에 규정된 버스 인터페이스에 따라 시스템에 연결될 주변장치가 AMBA에 통합되어지기 위해 필요한 주변장치 테스트 방법에 대해 기술하였다. APB 주변장치 모듈인 GPIO(General Purpose Input/Output)를 설계하고, AMBA 시스템과 연결하여 테스트하였으며, 이를 통해 SoC을 구현하는데 용이하게 사용되어질 수 있음을 보였다.

II. GPIO 설계

AMBA 시스템에 연결하여 테스트할 주변장치로서 GPIO를 설계하였다. 설계한 GPIO는 AMBA의 APB(Advanced Peripheral Bus) 주변장치인데, AMBA의 슬레이브 모듈로서 동작하며, 2개의 8비트의 프로그래머블한 입력/출력 16bit들로 구성되어 있다.[4] 다중 결합에 의해 16, 24, 32, 40 bits 등의 다양한 폭의 포트들을 만들 수 있으며, 인터럽트 인터페이스는 인터럽트 소스로 들어오는 많은 핀들을 설정할 수 있다. 시스템

리셋시에 GPIO 라인들의 디폴트는 입력이다. 데이터 입력, 데이터 출력, 패드 당 출력 enable을 사용하는 입력과 출력 패드 셀들과 인터페이스 한다. GPIO의 구성은 그림 1 에서처럼 APB 인터페이스를 담당하는 GpioApbif 와 하드웨어 컨트롤 및 입출력 멀티플렉서 역할을 하는 GpioAfm, 그리고, 인터럽트 감지 로직인 GpioInt 부분으로 구성되어진다.

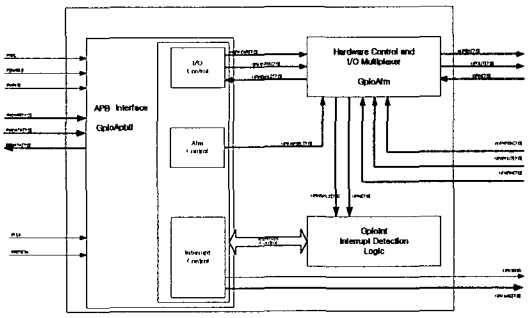

그림 1. GPIO 블럭도

표 1 은 설계한 GPIO 의 레지스터를 나타내고 있다. 레지스터 중에서 프로그램 가능한 레지스터는 다음과 같다.

(1) Data direction register

8비트, 각 핀을 입력과 출력으로 설정

(2) Data register

8비트이며, 입력으로 배정된 GPIO 라인상에서 입력 값을 읽는데 이용한다. 또한 출력으로 배정될 때 그 값을 프로그램하기 위해 사용된다.

표 1. GPIO 레지스터

Address	Name	Description
GPIO Base + 0x000- 0x3FC	GPIODATA	GPIO Data Register
GPIO Base+0x400	GPIODIR	GPIO Data Direction Register
GPIO Base+0x404	GPIOIS	GPIO Interrupt Sense Register
GPIO Base+0x408	GPIOIBE	GPIO Interrupt Both Edges Register
GPIO Base+0x40C	GPIOIEV	GPIO Interrupt Event Register
GPIO Base+0x410	GPIOIE	GPIO Interrupt Enable
GPIO Base+0x414	GPIORIS	GPIO Raw Interrupt Status
GPIO Base+0x418	GPIOMIS	GPIO Masked Interrupt Status
GPIO Base+0x41C	GPIOIC	GPIO Interrupt Clear
GPIO Base+0x420	GPIOAFSEL	GPIO Hardware Control Select
GPIO Base+0x600	GPIOITCR	Integration test control register
GPIO Base+0x604	GPIOITIP1	Integration test input read/set register
GPIO Base+0x608	GPIOITIP2	Integration test input read/set register
GPIO Base+0x60C	GPIOITOP1	Integration test output set register
GPIO Base+0x610	GPIOITOP2	Integration test output set register
GPIO Base+0x614	GPIOITOP3	Integration test output set register

(3) Interrupt control registers

GPIO는 인터럽트를 발생시킬 수 있다. 따라서 외부 GPIO 라인에 7개의 인터럽트 레지스터의 대응되는 비트를 통해 인터럽트를 독립적으로 트리거할 수 있다.

(4) Mode control select register

GPIO는 APB를 통해 소프트웨어적으로 제어될 수 있고, 하드웨어 컨트롤 인터페이스를 통해 하드웨어적으로 제어 가능하다. 각 GPIO 라인의 모드는 이 모드 제어 선택 레지스터에 의해서 선택된다.

GPIO를 VHDL로 설계하여 synopsys툴을 사용하여 합성한 모습은 그림 2 와 같다.

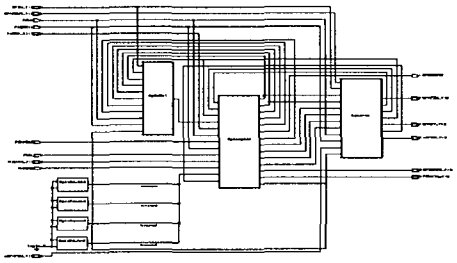

그림 2. GPIO 합성그림

설계한 GPIO가 테스트 벡터 환경을 통해 AMBA 버스 프로토콜에 맞게 동작하는지를 확인하기 위해 가상 AMBA 시스템을 구성하였다.

Ⅲ. 가상AMBA 시스템

시스템 버스인 AMBA의 프로토콜에 맞춰 설계한 것으로써, APB 인터페이스, Arbiter, Decoder, 멀티플렉서, Reset Controller, Static Memory 인터페이스, AMBA Test Interface Controller 로 구성되어 진다. 가상 AMBA 시스템에 구성되어진 각각의 블록들은 다음과 같다.

3.1 APB 인터페이스

AHB 버스와 APB 버스를 인터페이스 해주는 브릿지 역할을 하는 블록으로서, AHB transfer를 APB transfer로 변환시켜준다.

3.2 Arbiter(AHB master bus arbiter)

AHB 시스템 아비터로서, arbitration 계획에 따라 하나의 버스 마스터를 승인하고 버스의 소유권을 요청하는 버스 중재 역할을 한다. Arbitration 계획은 간단한 우선순위 인코더에 의해 우선순위가 가장 높은 마스터의 요청을 승인한다. 설계한 Arbiter의 우선순위는 다음과 같다.

(1) Async. Reset = ARM
(2) Pause mode = Default
(3) TIC
(4) 003
(5) 004
(6) ARM, when no others granted and ARM not split
(7) Default, when no others granted and ARM split

3.3 Decoder(Address decoder)

HSELx 출력을 통해 시스템 슬레이브를 선택하고, read data 멀티플렉서를 제어한다. HADDR 의 어드레스를 디코딩함으로써 슬레이브를 선택하게 된다. 설계한

Decoder의 어드레스 맵은 다음과 같다.

3.4 Multiplexor Master to Slave

중앙 멀티플렉서로써 마스터의 신호들을 슬레이브로 전달한다. 다른 마스터가 선택되지 않았을 때 기본 마스터 출력을 발생시킨다.

3.5 Multiplexor Slave to Master

중앙 멀티플렉서로써 슬레이브의 신호들을 마스터로 전달한다.

3.6 Reset Controller

Reset state machine 으로써, 전원이 인가된 후 reset 신호가 입력되면 HRESETn 신호를 출력한다.

3.7 Static Memory Interface

AMBA Static Memory Interface 로써 설정가능한 wait state를 가지고 있다.

3.8 AMBA Test Interface Controller

TIC는 시스템 테스트를 위한 AMBA 버스 마스터를 제공하기 위한 State machine 이다. 테스트 인터페이스 포맷인 TIF 테스트 벡터가 적용되는 동안 external 테스트 버스와 AHB 데이터 버스인 HRDATA 와 HWDATA를 제어한다.

TIC는 3개의 주요 블록으로 구성되어있다.

(1) 테스트 벡터 State Machine

TESTREQA 와 TESTREQB 신호들을 사용하여 external 테스트 버스에 적용되어질 테스트 벡터의 유형을 결정한다. state 다이어그램은 그림 3 과 같다.

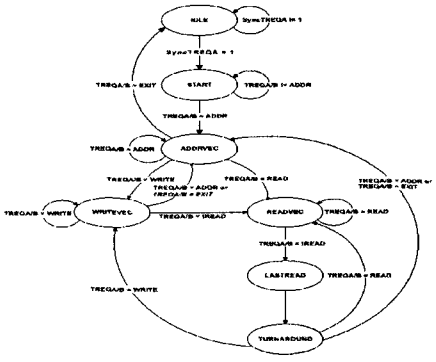

그림 3. 테스트 인터페이스 컨트롤러의 state 다이어그램

(2) Address and Control 레지스터

버스트 접근을 통한 어드레스 incrementer를 포함해서 어드레스와 제어신호들의 값을 붙잡아 두게 한다.

(3) AMBA Bus Master Interface

AMBA 버스 마스트 state machine의 역할을 하며, AMBA 버스 신호들을 제어한다.

테스트 인터페이스 컨트롤러를 통해 테스트가 시작되는 과정의 타이밍은 그림 4 와 같다.

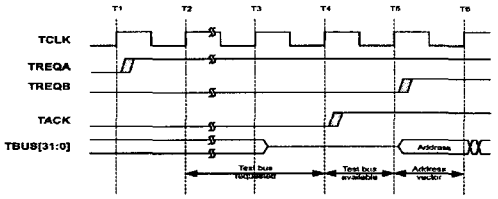

그림 4. Test start sequence

Ⅳ. 테스트벡터 환경을 통한 GPIO 검증

2절에서 설명한 GPIO를 AMBA 시스템으로 통합하여 테스트하였다. 이 테스트를 통해 임베디드 프로세서인 ARM 코어와 AMBA 버스에 Integration되기 전에 가상의 AMBA 시스템을 통하여 주변장치로서의 기능을 검증할 수 있다. AMBA 시스템에 Integration하기 위한 테스트벡터 환경을 그림 5 에 나타내었다. 테스트 벡터 환경의 구성내용은 크게 3절에서 설명한 가상 AMBA 시스템 부분과 주변장치 시스템 부분, 그리고 테스트 벡터를 받아들이는 테스트 인터페이스 컨트롤러 박스와 Integration 트릭박스로 구성되어진다.

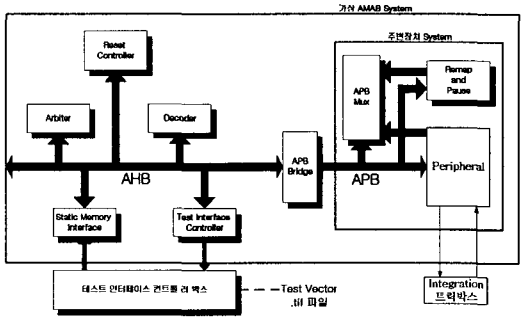

그림 5. AMBA System Integration 테스트 벡터 환경

4.1 테스트 인터페이스 컨트롤러 박스

테스트 인터페이스 컨트롤러 박스는 테스트 벡터파일을 읽어 들이는 부분으로써 시스템 테스트를 위한 버스 마스터 state machine을 제공하며, TIF 테스트 벡터가 적용되는 동안에 External 테스트 버스와 AHB Data Bus(HRDATA, HWDATA)를 제어한다.

TIF 테스트 벡터파일은 C프로그램에 의해 생성한다.

4.2 Integration 트릭박스

그림 6 에서처럼 트릭박스는 GPIO의 주된 입출력들이 Integration 벡터를 통해 잘 수행되어지는지를 테스트하는 블록으로써 이를 위해 GPIO의 입출력 값인 출력 nGPEN[7:0] 과GPOUT[7:0] 값을 XOR 시켜 입력 GPIN[7:0] 값으로 다시 내보낸다. 이를 통해 GPIO 가 실제 시스템에 Integration 되었을 때 데이터의 입출력을 정확히 하는지를 검증 할 수 있다.

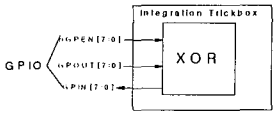

그림 6. Integration 트릭박스

4.3 테스트 벡터를 이용한 시뮬레이션 결과

C 소스를 통해 생성된 테스트 벡터를 테스트 인터페이스 컨트롤 박스에 입력하여 Mentor 의 modelsim 시뮬레이터를 통해 GPIO의 기능을 검증한 것이 그림 7 과 같다.

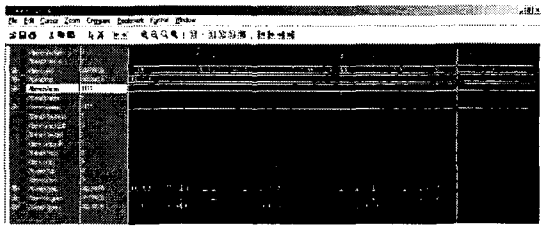

그림 7. GPIO Integration 시뮬레이션 결과 1

그림 7 에서 nGPEN[7:0] xor GPOUT[7:0] = GPIIN[7:0] 즉, nGPEN 값이 "00000000" 이고, GPOUT 값이 "11111111"일 때 GPIN 값이 xor 되어 "11111111" 값으로 나옴으로써 GPIO의 입출력이 정확하게 나옴을 검증하였다.

그림 8 에서는 GPIO 가 AMBA 버스 프로토콜에 맞게 동작하는지를 검증한 시뮬레이션 파형으로 테스트되어진 내용은 다음과 같다.

hburst = "001" : INCR (Incrementing burst of unspecified length),
htrans = "00" : IDLE, htrans = "10" : NONSEQ,
hmaster = "0010" : 마스터 번호,
hresp = "00" : OK 응답신호,
hsize = "010" : transfer 크기 (32 bit)
hprot = "0011" : Protection 신호(Not cacheable, Not bufferable, Privileged access, Data access)

컨트롤 벡터값 중에서 hburst 는 INCR이고, 전송타입은 IDLE일때와 NONSEQ 일 때, 그리고, 슬레이브 응답신호가 OK일 때, 전송 크기는 32비트일 때를 테스트하였는데, 전송타입이 IDLE일 때 데이터 전송이 이루어지지 않음을 볼 수 있고, 어드레스와 제어 신호가 앞전에 이루어진 전송과 무관하게 Non-sequential하게 이루어짐을 볼 수 있다. 이를 통해 GPIO 가 AMBA 버스 규정에 맞게 동작함을 확인하였다.

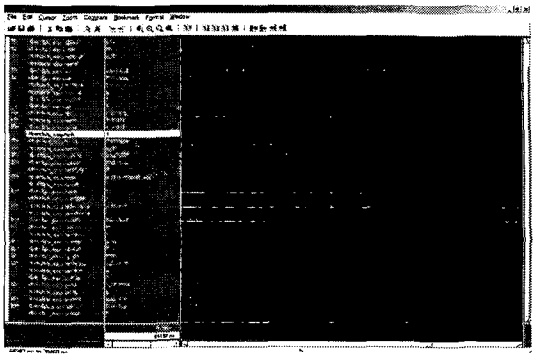

그림 8. GPIO Integration 시뮬레이션 결과 2

V. 결론

SoC을 위한 시스템 Integration을 위해 시스템 버스인 AMBA의 프로토콜에 맞추어 가상 AMBA 시스템에 필요한 AHB 마스터, 슬레이브, 아비터, 디코더 모델과 APB 브릿지 모델과 슬레이브 모델을 설계하였다. 여기에 가상 AMBA 시스템 부분과 주변장치 시스템 부분, 그리고 테스트 벡터를 받아들이는 테스트 인터페이스 컨트롤러 박스와 Integration 트릭박스를 설계하여, 주변장치를 테스트 할 수 있는 테스트 벡터환경을 구성하였다. 테스트 벡터 환경을 통해 설계한 GPIO가 AMBA 버스 프로토콜에 맞게 동작하는지를 검증하였다. 주변장치의 동작을 시스템에 통합되기 이전에 검증함으로써, SoC 구현과정에서 AMB 시스템을 통한 주변장치 테스트의 중요성을 알 수 있다.

참고문헌

[1] Steve Furber, ARM System-On-Chip architecture second edition, ADDISON WESLEY, 2000

[2] 김희용, "ARM SOC Design Example for Multi-Video Processor", 대한 전자공학회 2001 SOC Design Conference, Vol.1, pp.250, 2001

[3] AMBA Specification(Rev 2.0), Advanced RISC Machines. Ltd., May, 1999

[4] ARM PrimeCell General Purpose Input/Output(PL061) Technical Reference Manual, Advanced RISC Machines. Ltd., August, 2000