Excalibur™ 상에서의 DMAC 구현

황인기
한국전자통신연구원 네크워크핵심기술연구부

DMAC implementation on Excalibur™

Inki Hwang
Network Technology Laboratory
Electronics and Telecommunications Research Institute
E-mail : ikhwang74@etri.re.kr

Abstract

In this paper, we describe implemented DMAC (Direct Memory Access Controller) architecture on Altera's Excalibur™ that includes industry-standard ARM922T™ 32-bit RISC processor core operating at 200 MHz. We implemented DMAC based on AMBA (Advanced Micro-controller Bus Architecture) AHB (Advanced High-performance Bus) interface. Implemented DMAC has 8-channel and can extend supportable channel count according to user application. We used round-robin method for priority selection. Implemented DMAC supports data transfer between Memory-to-Memory, Memory-to-Peripheral and Peripheral-to-Memory. The max transfer count is 1024 per a time and it can support byte, half-word and word transfer according to AHB protocol (HSIZE signals). We implemented with VHDL and functional verification using ModelSim™. Then, we synthesized using LeonardoSpectrum™ with Altera Excalibur™ library. We did FPGA P&R and targeting using Quartus™. We can use implemented DMAC module at any system that needs high speed and broad bandwidth data transfers.

I. 소 개

멀티미디어 응용 시스템이 증가함에 따라 대량의 데이터를 빠른 시간 안에 처리할 수 있는 시스템 구조와 고성능의 DSP(Digital Signal Processing)의 요구가 증가되고 있다. 본 논문은 고속 데이터 처리를 가능하게 하는 DMAC (Direct Memory Access Controller) 장치의 구조와 DMAC 를 Excalibur™ 상에서 구현하는 방법에 대해 기술한다.

Excalibur™ 은 SoC (System On-a Chip) 개발의 편이를 위해 ARM922T™ 32-bit RISC processor 와 AMBA[1] bus architecture 를 구성하기 위한 component 들을 내장하고 있는 FPGA 이다. 그림 1 은 Excalibur™ 의 구조를 나타낸다.

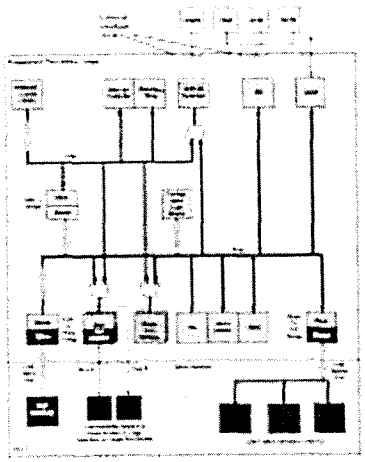

그림 1. Excalibur™ 의 구조

구현된 DMAC 는 peripheral(Peri) to Peri, Memory(MEM) to Peri 그리고 Peri to MEM 의 data 전송을 지원한다. 한 번에 전송되는 최대 데이터량은 1Kbyte 이며, 이는 AMBA 규격이 정한 최대 데이터 전송량이다. 전송되는 데이터의 크기는 Peri.의 특성에 맞춰 byte, half-word,

word 단위의 전송이 가능하다. 또한 해당 Peri.의 특성에 맞춰 생성하는 어드레스의 값의 고정, 증가(+1, 2, 4), 또는 감소(-1, 2, 4)를 제어할 수 있게 설계되었다. 지원 가능한 채널의 수는 사용자의 용도에 맞춰 쉽게 변경 가능하다. 각 채널의 우선순위 결정은 라운드-로빈 방법을 이용한다.

II. DMAC Architecture

그림 2 는 구현된 DMAC 의 구조로 AMBA 프로토콜에 맞춰 동작하도록 설계되었다.

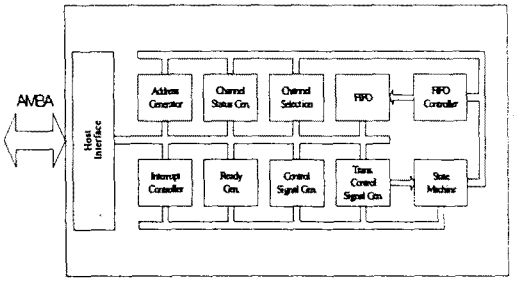

그림 2. DMAC 구조

구현된 DMAC 는, 프로세서 또는 선택된 Peri. 와의 제어 신호, 데이터의 입출력을 위한 Host interface, 입,출력 레지스터의 베이스 값을 가지고 어드레스의 고정, 증가, 감소에 맞춰 어드레스를 생성해 내는 Address generator, DMA 동작의 완료나 에러의 발생으로 프로세서의 제어를 필요로 할 때, interrupt 를 발생시키는 interrupt controller, 현재 DMA 동작 중인 채널의 상태를 감시하는 Channel Status generator, 라운드 로빈 방식으로 채널의 우선순위를 결정하여, DMA req. 입력에 따라 DMA 채널을 선택하는 Channel selection, AMBA 프로토콜에 맞춰 신호를 생성해주는 기능을 수행하는 Ready Generator 와 Control signal Generator, DMA 동작 중 DMA 의 상태 정보를 생성하는 Tans. Control signal generator, DMAC 동작 전반을 제어하고, 현재의 상태 정보를 생성하는 State machine, 그리고, 데이터의 입,출력과 제어 기능을 수행하는 FIFO, FIFO controller 로 구성된다.

그림 3 은 상기 그림 2 의 State machine 의 동작 순서를 나타내는 것으로, Trans control signal generator 모듈에서 생성하는 제어 신호를 입력으로 DMAC 의 현재 동작 상태를 나타낸다. 상태 정보는 IDLE, Write_start, Write_busy, Write_last, Write_complete, Read_start, Read_busy, Read_last, Read_complete, Wait 로 구성된다. IDLE 은 동작이 없이 대기 중임을 나타내고, Read_start 는 DMA 요청이 발생하여, DMAC 가 arbiter 로부터 버스의 마스터를 획득하면, DMA 를 요청한 Peri.로부터 최초로 데이터 읽기를 시작했음을 의미한다. Read_busy 는 burst 전송 중임을 의미한다. Read_last 는 마지막 읽기 데이터 임을 의미한다. Read_complete 는 DMA 를 요청한 Peri. 로부터 모든 데이터를 읽어왔음을 의미한다. Write_start 는 목적 모듈로 데이터 쓰기 동작이 시작되었음을 의미하고, Write_busy 는 burst 동작 중임을, Write_last 는 마지막 데이터 쓰기 과정임을 의미한다. Write_complete 는 쓰기 과정을 끝으로 모든 DMA 동작이 종료하였음을 의미하고, interrupt 를 발생시켜 프로세서로 하여금 버스의 마스터를 가져가도록 한다. WAIT 는 비 정상적인 동작이 발생 하였을 때, DMA 동작을 중지시키는 상태로 interrupt 를 발생시켜 프로세서에 DMA 동작 중 오류가 발생했음을 알린다.

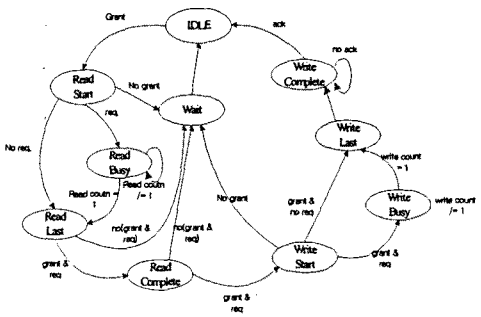

그림 3. DMA 동작 시 state machine

구현된 DMAC 는 Quartus™ 을 이용하여 Excalibur™ 에 targeting 하였다. 제공되는 AMBA interface 에 DMAC 와 새로운 마스터를 연결시키기 위해 필요한 AMBA component 들을 구현하여 그림 1 의 PLD(Programmable Logic Device)에 targeting 하였다. 합성의 과정은 Leonardo spectrum™ 을 이용하였고, Altera™ library 를 적용시켰다.

Targeting 이 완료 되면, Quartus™ 상의 ADU™(ARM Debugger Unit)를 이용하여, 프로세서로부터 DMAC 의 각 레지스터 값에 액세스 하여, 그 동작의 정확성 여부를 판단할 수 있다.

III. Verification

구현된 DMAC 의 기능 검증은 Modelsim™ 을 이용하여 수행하였다. Quartus™ 은 프로세서를 이용한 시스템에서 프로세서 없이 하드웨어만으로 검증이 가능할 수 있도록 유사 버스 제어 신호 및 데이터 신호를 생성해 줄 수 있는 기능을 지원해준다. 아래의 그림 3 과 4 는 Peri to Peri 의 DMA 동작 시의 Timing diagram 이다.

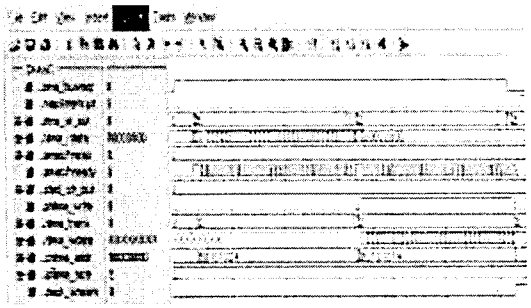

그림 4. DMA 동작 시 DMAC timing diagram

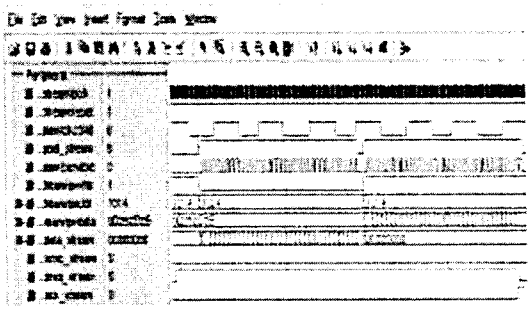

그림 5. DMA 동작 시 Peri. Timing diagram

그림 4, 5 에서 Peri 의 DMA req. 신호를 입력 받아, 버스 마스터 req. 신호를 생성하고, 마스터로 grant 되었을 때, Peri.의 저장 매체로부터 DMAC 로 32 비트*메인 동작 주파수/2 cycle 의 속도로 데이터를 전송하는 것을 볼 수 있다. 2 cycle 의 delay 가 발생하는 것은 AMBA APB(Advanced Peripheral Bus)가 전력 소비를 최소화하기 위해 Peri. 의 동작 속도를 AHB 메인 클록 2 cycle 에 한번씩 수행하도록 하기 때문이다. Peri.로부터 읽기 동작이 완료하면, 목적 Peri(동일 Peri.를 예로 하였다)로 쓰기 동작을 수행한다. 쓰기 동작이 완료 후 DMA 동작의 종료를 프로세서에 인지시키기 위해 interrupt 를 발생시킨다.

IV. 결론

Excalibur™ 상에서 AMBA 의 규격을 따른 DMAC 를 구현하였다. 구현된 DMAC 는 Peri to Peri, MEM to Peri 그리고 Peri to MEM 의 data 전송을 지원한다. 전송되는 데이터의 크기는 Peri.의 특성에 맞춰 byte, half-word, word 단위의 전송이 가능하다. 또한 생성되는 어드레스의 값의 고정, 증가(+1, 2, 4), 또는 감소(-1, 2, 4)를 제어할 수 있게 설계되었다. 지원 가능한 채널의 수는 사용자의 용도에 맞춰 쉽게 변경 가능하며, 각 채널의 우선순위 결정은 라운드-로빈 방법을 이용한다.

Excalibur™ 의 resource 사용률은 아래의 표와 같다.

표 1. DMAC 의 Resource 사용률

	사용량	전체용량	비율(%)
Logic element	8286	38400	21
Total ESB bits	20480	327680	6

참고문헌

[1] AMBA™ Specification(Rev 2.0)

[2]ARM PrimeCell™ DMA Controller(PL080)

[3]Excalibur™ Hardware Reference Manual