효율적인 데이터 전송과 하드웨어 최적화를 위한 AMBA AXI4 BUS Interface 구현

Implementation of the AMBA AXI4 Bus interface for effective data transaction and optimized hardware design

김현욱*, 김근준*, 조기쁨*, 강봉순**
Hyeon-wook Kim*, Geun-jun Kim*, Gi-ppeum Jo*, Bong-soon Kang**

요 약

최근 디지털 기기의 다기능화, 휴대화 및 서비스 정보의 대용량화 등으로 인하여 고집적, 저전력, 고성능 SoC(System on Chip) 설계에 대한 요구가 점차 증가하고 있다. 시스템이 빠르게 발전함에 따라 요구되는 하드웨어 성능이 다양해지고 있으며 빠른 설계 확인을 위하여 FPGA(Field Programmabel Gate Array)를 채택하는 시스템이 증가되고 있는 추세이며 FPGA를 채택한 시스템에서는 FPGA와 제어하는 CPU인 ARM코어를 사용한 SoC 시스템이 늘어났다. 이러한 시스템에서 사용되는 AXI(Advanced eXtensible Interface) Bus는 여러 방법으로 이용되지만, 기존의 연구에서는 AXI Slave 구조로 설계가 되어 있다. Slave 구조에서는 CPU가 계속 데이터 전송에 관여하게 되어 자원을 다른 곳에 사용하지 못하는 문제와 AXI Bus가 사용되지 않는 시간이 길어서 전송효율이 떨어지는 문제가 있다. 본 논문에서는 이와 같은 문제를 해결하고자 AXI Master구조를 제안하고, Slave구조와 Master구조의 소모클럭과 합성결과를 비교한 결과, Master구조가 Slave구조에 비해 소모클럭은 51.99% 감소한 것을 확인하였으며, Slice는 31% 정도 감소하였다. 또한, 최대 동작주파수는 107.84MHz로써 약 140% 증가 되는 것을 확인하였다.

ABSTRACT

Recently, the demand for high-integrated, low-powered, and high-powered SoC design has been increasing due to the multi-functionality and the miniaturization of digital devices and the high capacity of service informations. With the rapid evolution of the system, the required hardware performances have become diversified, the FPGA system has been increasingly adopted for the rapid verification, and SoC system using the FPGA and the ARM core for control has been growingly chosen. While the AXI bus is used in these kinds of systems in various ways, it is traditionally designed with AXI slave structure. In slave structure, there are problems with the CPU resources because CPU is continually involved in the data transfer and can't be used in other jobs, and with the decreased transmission efficiency because the time not used of AXI bus beomes longer. In this paper, an efficient AXI master interface is proposed to solve this problem. The simulation results show that the proposed system achieves reductions in the consumption clock by an average of 51.99% and in the slice by 31% and that the maximum operating frequency is increased to 107.84MHz by about 140%.

Keywords : AMBA Protocol, AXI4 Master, AXI4 Slave, Direct Memory Access(DMA), Field Programmable Gate Array(FPGA)

I. 서 론

최근 디지털 기기의 다기능화, 휴대화 및 서비스 정보의 대용량화 등으로 인하여 고집적, 저전력, 고성능 SoC 설계에 대한 요구가 점차 증가하고 있다. 시스템이 빠르게 발전함에 따라 요구되는 하드웨어 성능이 다양해지고 있으며 빠른 설계 확인을 위하여 FPGA를 채택하는 시스템이 증가되고 있는 추세이다.[1][2] FPGA를 이용한 시스템은 하드웨어 리소스 변화에 신속하게 대처할 수 있고 효율적으로 디지털 하드웨어를 구현할 수 있으며 개발기간을 단축할 수 있다. FPGA를 이용하여 검증을 하려면 FPGA를 제어하는 CPU가 필요함으로, FPGA와 ARM코어를 사용한 SoC 시스템이 늘어났다.

이러한 시스템에서는 FPGA와 ARM코어 간의 통신을 위한 Bus가 존재하여야 하는데, 보편적으로 사용되고 있는 것은 ARM사의

* 동아대학교
** 동아대학교 (교신저자)
투고 일자 : 2014. 2.20 수정완료일자 : 2014. 4. 24
게재확정일자 : 2014. 5. 2
※ 본 논문은 동아대학교 학술연구비 지원에 의하여 연구되었습니다.

AMBA(Advanced Microcontroller Bus Architecture) Protocol이다.[3][4] AMBA 4.0 Protocol에는 AXI4 전송방식이 포함 되어 있는데, AXI는 어드레스와 데이터 및 컨트롤 버스를 분리하고, 시작 주소만 주어지면 연속적인 데이터 통신이 가능한 Burst 기반 프로토콜이다. 기존의 버스는 하나의 트랜잭션이 종료된 이후에 다음을 수행하였다. 따라서 저속 장치에 엑세스해버리면 그것이 끝날 때까지 버스가 점령되어버리는 문제점이 있었는데, AXI에서는 먼저 생성된 트랜잭션이 종료되기 전에 다음 트랜잭션을 수행할 수 있다. 이것은, 트랜잭션에 각각의 ID를 부여하여 가능한 것으로써, 병렬처리를 가능하도록 만들었다. 또한 Read 및 Write 채널을 분리하여 읽기와 쓰기 동작이 동시에 독립적으로 수행 가능하다. 기존의 연구들은 이러한 AXI4 Bus를 이용하여 FPGA와 ARM코어, 주변장치 간의 통신을 구현을 하였다. [5][6]

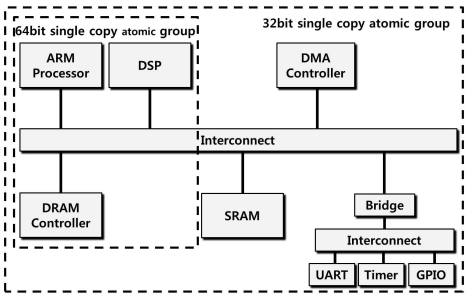


그림 1. MCU 내부의 AXI4 Bus 구조 [7]
Fig .1. AXI4 bus structure of the internal MCU.

그림 1에는 AXI4 Bus의 구조를 나타내었다. 고속의 장치인 ARM processor와 DSP(Digital Signal Processor), RAM등은 Interconnect에 직접적으로 연결되고, 저속의 장치인 UART, GPIO(General Purpose Input/Output), Timer등은 Bridge를 거쳐서 Interconnect에 연결된다. 각 모듈들은 Master와 Slave가 존재한다. Master는 통신을 제어하는 주체가 되는 기능을 수행하며, 데이터의 흐름을 직접적으로 제어할 수 있다. Slave는 Master에서 제어하는 대로 움직이는 기능만을 수행한다. 또한, 데이터의 흐름을 직접 제어할 수 없고, Master에서 전달해달라는 명령이 입력되기 전에는 데이터를 Master로 전송할 수 없다. 기존의 연구에서는 CPU가 Master로 구성되어 있으며, Slave로 DMA(Direct Memory Access)가 연결된다. 그 하위로, DMA의 Master에 만들어진 Logic이 Slave로 연결된다. 만약 데이터를 Logic으로 보내고 싶으면, CPU에서 DMA에 명령을 내려서 DMA가 DDR3에서 데이터를 읽어 들여와서, Logic으로 전송하게 된다. 출력은 Logic에서 입력된 데이터가 모두 처리되고 나면 출력이 있다는 신호를 출력하고, CPU가 이것을 받아들여 DMA에 명령을 내려서 DMA가 Logic에서 DDR3로 데이터를 전송하게 된다. 이러한 방식을 반복하면서 1 Frame 데이터를 처리하게 된다.[8][9]

하지만 Slave 구조에서 Data의 처리는 위와 같은 구조이기 때문에 CPU가 관장하여 직접 전송시킨 데이터에 한해서 처리되고 CPU의 제어 없이는 다른 처리가 불가능하다. 또한, 데이터 전송 시 Slave 모드에서는 IP가 직접 데이터를 DDR3로부터 가져오지 못하기 때문에, CPU의 제어를 받아 DMA를 통해 데이터를 가져오게 되고, 이 시간동안 AXI4 Bus는 동작하지 않게 된다. 따라서 AXI4 Bus와 DDR3 전송이 동시에 동작되지 않으며, 데이터의 전송시간이 많이 걸린다.

따라서 본 논문에서는 CPU의 직접적인 제어가 있어야 하는 문제와 Logic이 DDR3에 직접적인 데이터 제어를 하지 못하는 문제점을 해결 할 수 있는 AXI4 Master구조를 제안한다. 제안된 구조는 CPU에서 하던 제어를 Logic내부에서 직접 제어를 할 수 있는 기능을 포함하고, 이러한 기능을 구현하기 위하여 Logic내부에 제어를 위한 State Machine을 구현하였다. 또한, 직접적으로 AXI4 Bus를 통하여 데이터를 가져올 수 있고, 가져온 데이터를 IP Logic에 전송하는 기능을 구현하였다. 데이터를 Read/Write하는 시간을 절약하기 위해서, 두 개의 버퍼를 사용하여 스위칭하여 각각 Read/Write를 동시에 함으로써 시간을 절약할 수 있다.

Ⅱ장에서는 AXI4 Master구조에 대하여 논하고, 동작에 대한 내용을 설명하는 것과 함께 동작파형을 제시한다. Ⅲ장에서는 AXI4 Slave구조와 Master구조를 Zynq-7000 ZC706 Board를 이용한 합성결과와 성능에 대하여 논한다. Ⅳ장에서는 결론을 맺는다.

Ⅱ. 제안된 AXI4 Master구조 전송방식

AXI4 Slave구조에서 AXI4 Master구조로 변경하게 되면 바뀌는 점은, Slave구조에서는 주변 Interface에 직접적으로 접근이 불가능 했지만 Master구조에서는 주변 Interface에 직접 접근이 가능해진다. 따라서 Slave 구조에서는 불가능했던 메모리에 직접 엑세스가 Master구조에서는 가능하며, 이는 다른 장치의 제어 없이 Logic이 직접 데이터를 가져올 수 있다는 것이며, 다른 장치의 제어를 기다리는 시간이 필요하지 않다. 이러한 점은 Bus의 사용효율을 높여 더욱 빠른 데이터 전송을 가능하게 한다.

그림 2에는 기존의 연구에 사용된 AXI4 Slave구조의 전송방식을 나타내었다. 기존의 전송방식은, CPU가 DMA에 명령을 내려 DDR3에서 데이터를 가져오도록 하고, 그 데이터를 IP Logic으로 보낸다. 입력데이터가 처리가 완료되면 완료되었다는 신호를 출력하고, 그 신호를 읽어서 CPU가 DMA를 통하여 IP Logic에서 DDR3로 데이터를 저장하게 된다. 이러한 동작들은 CPU가 내린 명령을 통해 제어를 하고, 데이터 전송을 하게 되어 CPU가 명령을 내릴 때만 동작을 하게 됨으로써 병렬처리가 불가능하고, CPU가 항상 점유되어 있는 문제점이 있고, 데이터의 로드 시간 때문에 전체 동작시간에 비해 AXI4 Bus가 동작하지 않는 시간이 많다. [9]

그림 3에는 제안된 구조의 전송방식을 나타내었다. 제안된 구조는 CPU가 아닌 Logic이 직접 AXI4 Bus를 통하여 DDR3에 접근하여 데이터를 가져오게 되고, 가져온 데이터를 IP Logic에 전달하는 동시에 Bus를 통하여 다른 버퍼에 그 다음 데이터를 가져와서 저장한다. IP Logic으로 전달하는 동시에 다른 버퍼에 그 다음 데이터를 가져오면 데이터를 가져오는 시간을 줄일 수 있다. IP Logic으로 전달된 데이터의 처리가 끝나면 IP Logic에서 결과를 출력하고, 그것을 AXI4 Bus를 통하여 DDR3에 저장하게 된다.
이러한 과정을 반복하게 되는데, AXI4 Slave 구조와 비교 하였을 때, 데이터를 직접 가져오고, 스위칭을 통하여 데이터를 미리 불러오는 과정으로 인해, AXI4 Bus를 통하여 데이터를 전송하지 않는 시간이 감소하게 되며, 전체 데이터를 전송하는 시간이 감소하게 된다. 이것은 전송효율이 증가하게 되는 이득을 갖게 된다.

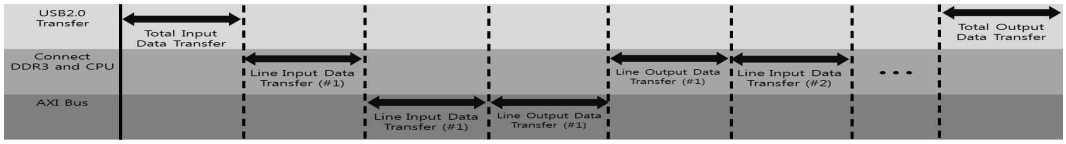


그림 2. AXI4 Slave구조의 데이터 전송방식

Fig .2. Data transmission scheme of AXI4 slave structure.

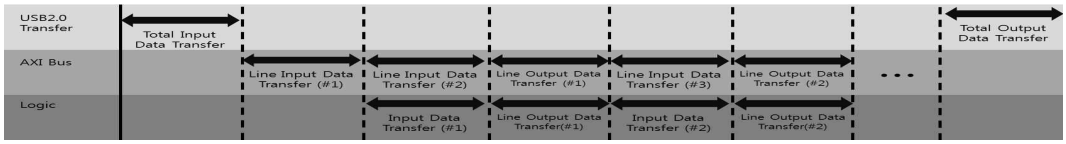


그림 3. AXI4 Master구조의 데이터 전송방식

Fig .3. Data transmission scheme of AXI4 master structure.

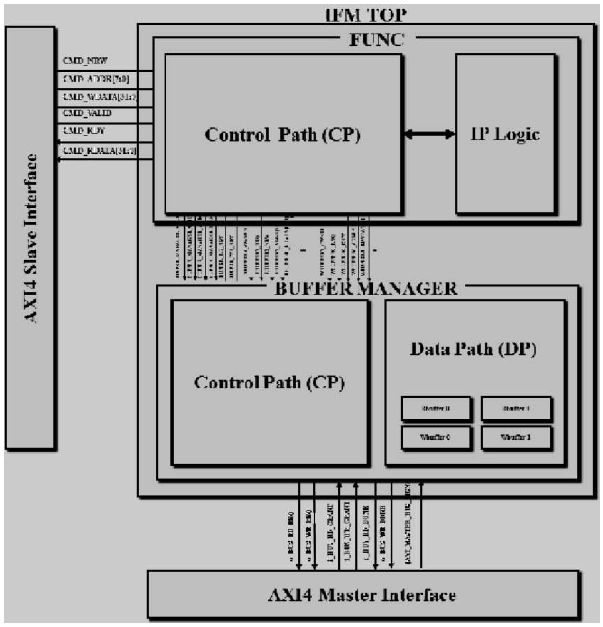


그림 4. 제안된 AXI4 Master 블록 다이어그램

Fig .4. The proposed AXI4 master block diagram.

그림 4에는 제안된 AXI4 Master구조를 나타내었다. 제안된 구조는 크게 AXI4 Slave Interface와 AXI4 Master Interface, 그리고 IFM TOP으로 구분된다. AXI4 Slave Interface는 IFM TOP이 AXI4 Bus에 Slave로 연결해주는 역할을 하며, AXI4 Master Interface는 IFM TOP이 AXI4 Bus에 Master로 연결해주는 역할을 한다. IFM TOP은 FUNC 블록과 BUFFER MANAGER블록으로 나누어지는데, FUNC 블록은 실질적인 데이터의 처리, 즉 알고리즘 역할을 하는 IP Logic이 있으며, BUFFER MANAGER와 IP Logic 간의 데이터흐름을 제어하는 Control Path(CP) 블록이 있다. CP는 CPU에서 내린 명령이 AXI4 Slave Interface를 통하여 입력되게 되면 해당 명령을 저장하고 있다가 전송해야 할 타이밍에 맞게 BUFFER MANAGER로 전송하는 역할을 하고, BUFFER MANAGER에서 명령을 수행하여 데이터들이 수신되면 IP Logic으로 전송해주는 역할을 한다. BUFFER MANAGER 블록은 AXI4 Bus와 Master로 연결되는 블록으로써, DDR3에 저장된 데이터를 가져와서 FUNC블록으로 전송해주는 블록이다. BUFFER MANAGER에는 Data Path(DP)와 Control Path(CP)가 있는데, DP는 Line Memory를 포함한 블록으로써, 입출력데이터를 저장하기 위한 블록이다. CP에서는 FUNC의 CP로부터 입력된 명령을 판단하여 어떤 동작을 할 것인지 결정을 하고, 해당명령을 수행하기 위해서 AXI4 Bus에 접근하여 데이터를 입력받거나 출력한다. 데이터를 입력받게 되면 AXI4 Bus로부터 받은 데이터들을 BUFFER MANAGER의 DP의 Line Memory에 저장하게 되고, 출력할 때에는 DP의 Line Memory에 저장된 데이터를 AXI4 Bus로 출력하게 된다. 각각의 Memory는 2개의 셋트로 구성이 되어있으며, 한 개의 Memory가 Write가 되고 나면 다음번에는 다른 Memory가 Write되고, 앞에서 Write되었던 Memory는 Read됨으로써 Read/Write에 소비되는 시간을 감소시킬 수 있다.

그림 5에는 명령과 Control Register를 받는 동작과형을 나타내었다. 해당 동작은 FUNC의 CP에서 이루어지며, 명령을 받고 Control Register를 받아야만 알고리즘이 동작하므로, 해당 동작을 가장 먼저 수행이 되어야만 모든 동작이 진행이 되도록 설계되어 있다. 만약 명령과 Control Register의 값을 받지 않으면 입력될 때까지 대기한다. AXI4 Slave Interface를 통하여 입력으로 Control Register와 입/출력 주소를 입력하고, 해당 동작이 완료되면 FUNC CP에서 Flag신호를 생성한다. Flag신호를 알고리즘에서 확인하면, FUNC CP가 명령과 Control Register를 모두 받았다고 판단한 후, FUNC의 IP Logic으로 명령과 Control Register를 전송한다. 해당 데이터를 받은 IP Logic은 RST DONE신호와 OP DONE신호를 발생시켜 받았다는 것을 FUNC CP에 알린다. 이것이 완료되면 State가 넘어가면서 동작을 시작한다.

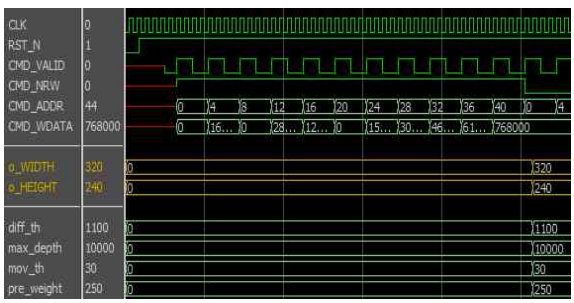


그림 5. 명령과 control register를 받는 동작파형

Fig .5. Receive the control register and command simulation waveform.

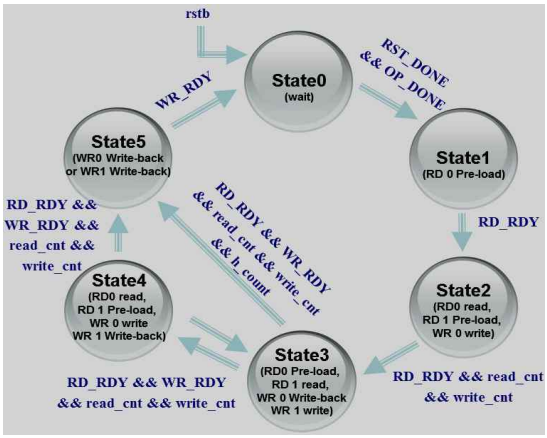


그림 6. FUNC CP의 State Machine
Fig .6. State machine of FUNC CP.

그림 6에는 FUNC의 CP내부의 State Machine의 동작을 나타내었다. 해당 State Machine은 BUFFER MANAGER의 동작과 FUNC의 IP Logic으로 입력되는 데이터를 제어하는 State Machine이다. State 0에서는 명령과 Control Register가 모두 입력될 때까지 IDLE상태를 유지하고, FUNC CP가 명령과 Control Register를 받아서 IP Logic으로 전달한 후, IP Logic이 명령과 Control Register를 받으면 State 1로 넘어간다. State 1에서는 AXI4 Bus를 통하여 DDR3에서 BUFFER MANAGER DP의 BUFFER 0로 데이터를 저장하는 동작을 수행하고, State 2에서는 BUFFER 0에 저장된 데이터를 IP Logic으로 전송해주는 동작과 동시에, 다음 입력 데이터를 AXI4 Bus를 통하여 DDR3에서 BUFFER 1에 데이터를 저장하는 동작을 수행한다. IP Logic으로 전송된 데이터의 처리가 끝나서 출력이 나오면 해당 데이터를 BUFFER MANAGER DP의 WBUFFER 0에 저장한다. State 3에서는 State 2에서 BUFFER 1에 저장한 입력데이터를 IP Logic으로 전송하게 되고 다음의 입력데이터를 AXI4 Bus를 통하여 DDR3에서 BUFFER 0에 저장한다. 그리고 AXI4 Bus를 통하여 WBUFFER 0에 저장된 데이터를 DDR3에 저장하고, IP Logic에서 나오는 출력데이터를 WBUFFER 1에 저장한다. State 4에서는 BUFFER를 바꿔서 동일한 동작을 수행하게 되고, State 3과 4를 번갈아가면서 수행하다가, 마지막 Line이 되면 State 5로 넘어가서 마지막 출력데이터를 AXI4 Bus를 통하여 DDR3에 출력을 하면 State Machine의 동작이 완료된다.

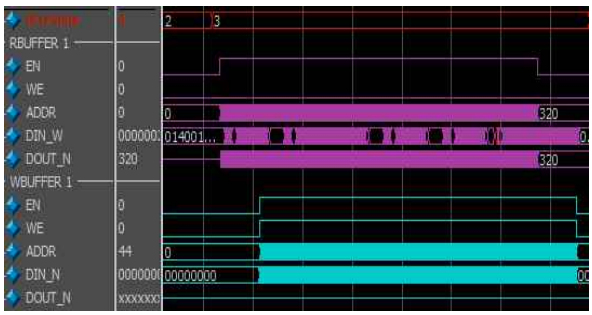


그림 7. Read와 Write의 동작파형
Fig .7. Data read and write simulation waveform.

그림 8. 명령을 받는 BUFFER MANAGER의 동작파형
Fig .8. BUFFER MANAGER to receive the command simulation waveform.

제안된 AXI4 Master구조를 Verilog-HDL로 설계 후 동작시킨 파형을 그림 7과 그림 8에 나타내었다. IP Logic에는 입력데이터가 F/F을 거친 후 출력되는 것으로 하였다. 그림 7에는 BUFFER MANAGER의 Line Memory에 데이터를 Read/Write하는 동작 파형에 대해 나타내었다. 그림 8에는 BUFFER MANAGER가 데이터를 옮기는 명령을 받는 파형을 나타내었다. 데이터를 옮기고자 할 때, 데이터를 옮길 Source의 시작 어드레스와 길이, 동작에 대한 명령을 전송하면 그에 대한 동작을 실행한다. Source의 시작 어드레스와 길이를 받으면 그 길이만큼 데이터를 가지고 오는 기능은 DMA와 같은 기능이라고 말할 수 있다. 이러한 방식은 어드레스 채널의 데이터 전송을 현저히 감소시킬 수 있다.

그림 9에는 Pre-Load와 Write-Back에 대해서 나타내었다. Pre-Load는 DDR3에 있는 데이터를 BUFFER Manager로 가져오는 과정을 말하는 것이다. 또한, Write-Back은 BUFFER MANAGER에 담겨있는 데이터를 DDR3로 저장하는 것이다. 입력된 데이터가 Pre-Load가 되어서 BUFFER MANAGER DP에 저장되는 것을 확인하였으며, 다음 State에서 해당 데이터가 IP Logic으로 입력되는 Read동작을 확인하였다. 그리고 IP Logic에서 출력된 데이터가 BUFFER MANAGER DP에 Write되고, 그 데이터가 다음 State에서 Write-back되어 출력되는 것을 확인하였다. 이러한 동작을 확인했을 때 설계한 Logic이 정상적으로 설계가 되었다는 것을 알 수 있고, AXI4 Bus에 연결되어 제대로 동작한다고 볼 수 있다.

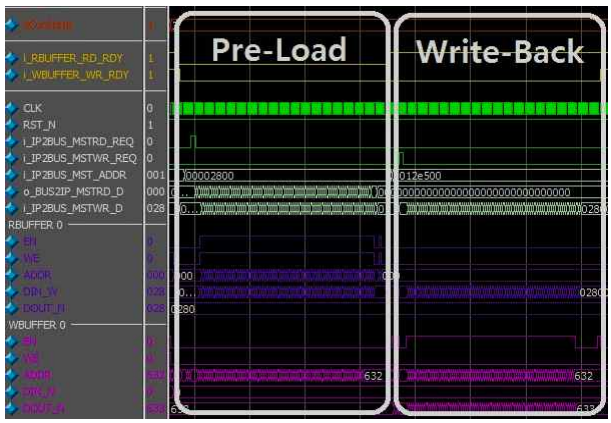


그림 9. Pre-load와 Write-back의 동작파형
Fig .9. Write-back and pre-load simulation waveform.

Ⅲ. 설계 검증 및 결과

본 논문에서 제안된 방법을 Verilog-HDL으로 설계 하였으며, 설계된 Logic을 Xilinx PlanAhead를 이용하여 합성하여, Xilinx사

의 ZC706 Board에 업로드 하여 실제동작을 확인하였다. 설계된 Logic의 알고리즘은 입력된 데이터를 F/F을 거쳐 출력으로 내보내는 것을 사용하였다. 제안된 구조를 검증하기 위해서 1Frame의 영상 데이터를 PC에서 USB2.0을 통하여 ZC706 Board의 DDR3 Memory에 입력하였다. 데이터를 입력하고 나면, 제안된 구조에서 데이터를 입력받게 되고, 알고리즘으로 입력데이터가 전송되게 된다. 여기에서 사용된 알고리즘은 입력된 데이터를 F/F을 거쳐서 출력으로 보내는 것을 사용하였다. 처리가 완료되면 출력데이터를 알고리즘에서 출력하게 되고, 그것을 AXI4 Bus를 이용하여 DDR3에 저장한다. 저장된 데이터를 PC에서 USB2.0을 통하여 가져오게 된다. 이러한 검증을 진행하였을 때, 입력데이터와 출력데이터가 동일하게 나오는 것을 확인하였다. 따라서 제안된 구조가 정상적으로 동작한다는 것을 알 수 있다.

그림 10에는 Slave 구조와 Master 구조에서 측정된 소모클럭을 비교하였다. 소모클럭의 확인 방법은, 데이터 전송 시작 전부터 연산이 끝나서 데이터를 출력했을 때까지의 소모클럭을 측정하였고, 측정된 클럭은 ZC706 Board에 포함된 Cortex-A9의 ARM코어에서 동작되는 클럭을 측정한 것이다. 해당 동작을 할 때의 ARM 코어의 클럭은 666MHz이다. Slave Mode로 동작하는 경우에는 평균 131,696,146클럭을 소모하였고, Master Mode로 동작하는 경우에는 평균 68,476,120클럭을 소모하였다. Master Mode로 변경하였을 때의 소모클럭은 약 51.99% 감소하는 것을 볼 수 있다. 이러한 결과로 Slave Mode보다 Master Mode가 더욱 빠르게 동작한다는 것을 알 수 있다.

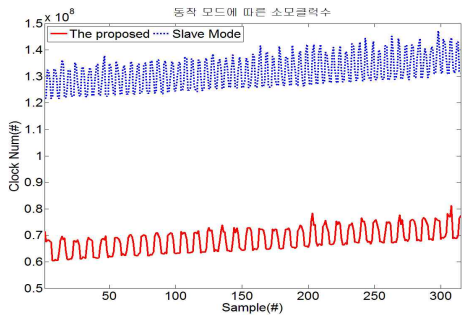


그림 10. AXI4 Master구조와 Slave구조의 소모클럭 비교
Fig .10. Compare of consumption clock AXI4 slave and AXI4 master structure.[9]

표 1은 AXI4 Slave 구조를 Xilinx PlanAhead로 합성한 결과를 나타내고 있다. Slave 구조에서는 데이터를 전송하기 위한 DMA를 포함하고 있다. Registers는 25,560개를 사용하여 전체면적의 5%를 차지하였고, LUTs는 13,627개를 사용하여 6%가 사용되었다. Slice는 20,546개를 사용하여 전체의 37%를 차지하였다. 전체 시스템의 최대 동작주파수는 76.846MHz까지 확보되는 것을 확인하였다. AXI4 Master 구조에서는 Master Logic안에 DMA기능이 포함되어 있으므로 따로 DMA Logic이 존재하지 않는다. 따라서 Slave구조보다 작은 영역을 사용한다. Registers는 7,634개를 사용하여 전체면적의 1%를 차지하였고, LUTs는 9.230개를 사용하여 4%가 사용되었다. Slice는 3,500개를 사용하여 전체의 6%를 차지하였다. 전체 시스템의 최대 동작주파수는 107.84MHz까지 확보되는 것을 확인하였다. Slave 구조와 비교하여 Slice는 31% 정도 감소하였고, 최대 동작주파수는 약 140% 증가 되는 것을 확인하였다. 그림 10과 표1을 보았을 때, Slave 구조보다 Master 구조가 소모클럭이 약 51.99% 감소하였다. 또한, Slice는 31% 정도 감소하였고, 최대 동작주파수는 약 140% 증가 되는 것을 확인하였다. 따라서 제안된 AXI4 Master 구조가 AXI4 Slave 구조보다 더욱 빠르게 동작하고, 적은 면적을 차지하는 것을 확인하였다.

표 1. 제안된 AXI4 Master구조와 Slave 구조의 합성결과비교
Table 1. Compare of implementation result the AXI4 slave and the proposed AXI4 master structure.

Slice Logic Utilization	Available	Slave	The proposed
Registers(#)	437,200	25,560(5%)	7,634(1%)
LUTs(#)	218,600	13,627(6%)	9,230(4%)
Slice (#)	54,650	20,546(37%)	3,500(6%)
RAMB36E1	1090	5(1%)	0(0%)
RAMB18E1	1090	1(1%)	1(1%)
DSP48E1	900	4(1%)	4(1%)
BUFG	32	1(3%)	10(31%)
Minimum period (ns)		13.013	9.273
Maximum Freq.(MHz)		76.846	107.84

*IDEC의 EDA Tool을 제공받아 수행하였음.

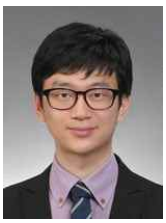
Ⅳ. 결 론

본 논문에서는 AXI4 Bus Master Interface를 구현하였다. 구현된 Logic을 Xilinx사의 ZC706 Board에서 Test하였으며, Test한 알고리즘의 출력 값이 정상적으로 나오는 것으로 보아 설계된 Logic이 정상적으로 설계되었다고 볼 수 있다. 기존의 AXI4 Slave 구조와 본 논문에서 제안된 AXI4 Master 구조를 비교한 결과 소모클럭은 51.99% 감소한 것을 확인하였으며, 전체 시스템의 최대 동작주파수는 107.84MHz까지 확보되는 것을 확인하였다. 또한, Slice는 31% 정도 감소하였고, 최대 동작주파수는 약 140% 증가되는 것을 확인하였다.

참 고 문 헌

[1] 서형선, 이서훈, 황선영 “AMBA 버스와 IP간의 통신을 위한 인터페이스 자동생성에 관한 연구,” 한국통신학회논문지, Vol. 29, No. 4A, pp. 390-398, April 2004.
[2] Shaila S Math, Manjula R. B, S.S. Manvi, Paul Kaunds “Data transactions on system-on-chip bus using AXI4 protocol,” 2011 International Conference on Recent Advancements in Electrical, Electronics and Control Engineering, pp. 423-427, December 2011.
[3] 이준섭, 정혜란, M. A. Ansari, 박성주 “AMBA AXI 기반의 효율적인 SoC 테스트 제어기 설계,” 대한전자공학회 2009년 SoC학술대회, pp. 146-149, May 2009.
[4] 이경호, 공진흥 “MPSoC 인터커넥션을위한 AXI 하이

브리드온칩버스구조설계," 전자공학회 논문지, Vol. 48, No. 8, pp. 33-44, August 2011.
[5] Santhoshi Yadav Pulicharla, V. Koteswara Rao "AMBA-AXI COMPLIANT MEMORY CONTROLLER," IJRRECS, pp. 316-321, August 2013.
[6] 정의봉, 송용호 "AMBA 기반 IP를 위한 온칩 네트워크 인터페이스 설계 및 구현," 대한전자공학회 전자 정보 통신 학술 대회 논문집 (CEIC) 2005, pp. 285-288, December 2005.
[7] ARM, "AMBA AXI and ACE Protocol Specification," www.arm.com, December 2013.
[8] 김상돈, 이승은 "AMBA AHB 기반 SDRAM 컨트롤러 설계," 한국산업정보학회논문지, Vol. 18, No. 5, pp. 33-37, October 2013.
[9] 김현욱, 최대성, 조호상, 강봉순 "범용적인 사용을 위한 AMBA AXI4 Bus Interface 설계," 한국신호처리시스템학회 2013 하계 학술대회 논문집, pp. 116-118, July 2013.



김 현 욱 (Hyeon-wook Kim)

2013년 2월 동아대학교 전자공학과(공학사)
2013년 3월 ~ 현재 동아대학교 전자공학과 석사과정

※주관심분야 : 영상 신호처리, VLSI architecture design

김 근 준 (Geun-jun Kim)

2013년 2월 동아대학교 전자공학과(공학사)
2013년 3월 ~ 현재 동아대학교 전자공학과 석·박사 통합 과정

※주관심분야 : 영상 신호처리, VLSI architecture design

조 기 쁨 (Gi-ppeum Jo)

2013년 2월 동아대학교 전자공학과(공학사)
2013년 3월 ~ 현재 동아대학교 전자공학과 석사 과정

※주관심분야 : 영상 신호처리, VLSI architecture design

강 봉 순 (Bong-soon Kang)
正會員
1985년 연세대학교 전자공학과(공학사)
1987년 미국 University of Pennsylvania 전기공학과 (공학석사)
1990년 미국 Drexel Universitiy 전기 및 컴퓨터공학과(공학박사)
1989년 ~ 1999년 삼성전자 반도체 수석연구원
1999년 ~ 현재 동아대학교 전자공학과 교수
2006년 ~ 2011년 멀티미디어 연구센터 소장
2006년 ~ 2013년 2단계 BK21 사업팀장
2013년 ~ 현재 BK21Plus 사업팀장
※주관심분야 : 영상신호처리, SoC설계 및 무선 통신