논문 2007-44SD-4-9

iPROVE 기반 SoC 검증을 위한 트랜잭터 구현

(A Transactor Implementation for SoC Verification with iPROVE)

조 종 현*, 조 중 휘**

(Chong Hyun Cho and Joong Hwee Cho)

요 약

본 논문에서는 트랜잭터를 정형화하고 DUT(Design Under Test)의 다양한 입출력에 따라 자동으로 트랜잭터를 생성해주는 생성기를 구현하였다. 호스트 컴퓨터와 FPGA(Field Programmable Gate Array) 사이의 PCI(Peripheral Component Interconnect) 인터페이스 신호들로 구성된 트랜잭터 프로토콜에 의존하는 블록과 DUT에 의존하는 블록으로 신호들을 재정리 함으로써 트랜잭터를 정형화하고 설계하였다. 구현된 트랜잭터의 자동 생성기는 DUT의 입출력에 관한 정보를 GUI(Graphical User Interface)를 통하여 입력받아 정형화된 하드웨어 블록들을 근간으로 입력정보를 추가하여 각각의 블록들을 만들어 하나의 Verilog 코드로 생성하는 동작을 한다. 자동 생성기의 정상동작을 확인하기 위하여 이미 검증된 하드웨어 블록을 이용하여 생성된 트랜잭터의 정상동작을 입증하였고, 사용자가 직접 설계한 트랜잭터와 비교함으로써 생성된 트랜잭터의 효율성을 입증하였으며 DUT의 다양한 입출력 정보들에 대하여 융통성 있게 동작하는 자동 생성기를 검증하였다. 트랜잭터 자동 생성기를 이용하는 경우 트랜잭터 설계시간을 단축 할 수 있고, 사용자가 트랜잭터 프로토콜를 이해하고 트랜잭터를 설계하는 부담을 줄여 시뮬레이션 속도가 빠른 트랜잭션 레벨 검증모드를 쉽게 사용 할 수 있도록 하였다.

Abstract

In this paper the proposed transactor is customized and a generator which roles of automatically generating the transactor according to DUT(Design Under Test)'s input and output is implemented. The customized transactor is designed by rearranging the signals of depending on DUT and transactor protocol which consists of signals of the PCI interface between host computer and FPGA(Field Programmable Gate Array). The implemented automatic generator of transactor generates a Verilog code of transactor by adding DUT's information about input and output ports. Performance and normal working of the generated transactor has been verified by experiments with some verified hardware IPs. Also, an efficiency of the transactor has been verified by comparing with user's manually designed transactor and generated transactor. Moreover, the generator's flexibility has been verified for DUT's information of variable input and output. In case of using the implemented generator, a design time of transactor is reduced.

Keywords : iPROVE, transactor, verification, SoC, H.264

Ⅰ. 서 론

오늘날 시스템이 대형화되고 복잡도가 증가됨에 따라 소프트웨어와 하드웨어를 함께 이용한 IP (Intellectual Property)기반의 SoC(System on a Chip) 설계가 보편화 되고 있다.[1] 이러한 변화에 따라 SoC 설계시 검증이 차지하는 비율이 전체 설계 중 50%~80%에 이른다. 검증에 보다 많은 인력과 시간을 필요로 하게 되었다. 이와 같은 SoC 설계에 대한 IP검증의 비중이 높아진 추세에 따라 기본적으로 Hardware Accelerated simulation, 넓게는 Emulation 검증 환경까지 지원하여 HW/SW를 동시에 개발할 수 있는 환경[2]을 제공하고 검증시간을 단축시킬 수 있는 툴로 다이나릿의 iPROVE을 사용하였다.

다이나릿의 iPROVE는 크게 두 가지 모드로 사이클 레벨 모드와 트랜잭션 레벨 모드의 검증모드를 지원한

* 정회원, 엠텍비젼(주)
(Mtekvison Co., Ltd.)
** 정회원, 인천대학교 멀티미디어시스템공학과
(Dept. of Multimedia Systems Eng., Univ. of Incheon)
※ 본 연구는 인천대학교 교내연구지원사업, 전자통신연구원 IT-SoC 사업단 및 IDEC의 지원으로 수행되었음
접수일자: 2007년3월6일, 수정완료일: 2007년4월10일

다. 사이클 레벨 모드는 사이클 단위로 데이터를 전달하지만 트랜잭션 레벨 모드에서는 읽기, 쓰기 등의 커맨드 단위로 데이터를 전달하므로 좀 더 빠른 시뮬레이션 결과를 얻을 수 있다.

트랜잭션 레벨 모드가 사이클 레벨 모드 보다 빠른 시뮬레이션 결과를 얻을 수 있음에도 불구하고 사용자가 반드시 인터페이스 프로토콜을 이해하고 검증할 사용자의 디자인에 의존된 트랜잭터를 반드시 설계해야 하는 부담과 RTL(Register Transfer Level) IP를 검증하는 사용자들이 대부분 하드웨어 엔지니어이기에 트랜잭션이라는 개념에 친숙하지 않아 사이클 레벨모드를 선호한다. 따라서 보다 나은 퍼포먼스를 얻을 수 있는 트랜잭션 레벨 모드를 사용자들이 쉽게 사용할 수 있도록 본 논문에서는 유저 인터페이스를 단순화하고 다양한 디자인 입출력에도 적용이 가능하도록 하는 트랜잭터를 생성하는 생성기를 구현하고 검증한다.

본론 I 장에서는 iPROVE의 구성 및 동작과 각 모드별 특성에 대해 간단히 기술하고, 본문 II장에서는 인터페이스 프로토콜을 간략히 정형화하여 구현된 트랜잭터의 구조 및 동작에 대하여 알아보고, 실험에서는 정형화된 트랜잭터를 근간으로 하는 트랜잭터 자동 생성기를 구현하고 실험 및 고찰을 통해 동작을 검증하고 결론으로 끝을 맺는다.

II. 본 론

1. iPROVE의 개요

iPROVE의 디자인 환경은 그림 1과 같이 호스트 컴퓨터상의 소프트웨어와 iPROVE 하드웨어 보드로 구성된다. iPROVE 하드웨어 보드 상에서의 DUT (Design Under Test)와 호스트 컴퓨터상에서 테스트 벤치 또는 사용자가 디자인한 시스템 중 DUT를 제외한 블록이 동작된다. 그래서 소프트웨어 시뮬레이션의 부담을 하드웨어에서 처리하도록 하여서 시뮬레이션 속도를 향상시킨다. iPROVE는 고성능 FPGA (Field Programmable Gate Array)를 이용하여, 개발하고자 하는 RTL설계를 PCI 인터페이스를 통해 소프트웨어 환경과 연동하는 방법에 의한 SoC의 하드웨어 소프트웨어 co-design 설계 검증 솔루션이다. 또한 각종 HDL 시뮬레이터와 연동하여, FPGA에 내장한 RTL회로를 시뮬레이션 함으로, 순수 소프트웨어 시뮬레이션에 비해 가속효과를 얻을 수 있으므로 복잡한 SoC의 RTL 개발 기간을 단축할 수 있다. VHDL, Verilog와

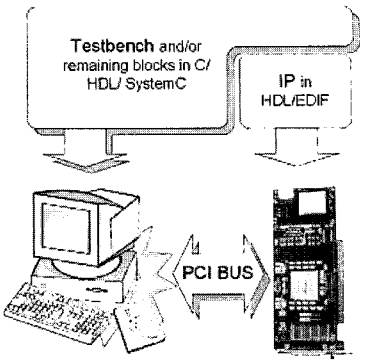

그림 1. 설계와 검증을 위한 iProve 환경
Fig. 1. iProve Environment for design and verification.

더불어 SystemC를 이용한 시뮬레이션을 통해 시스템 레벨 설계 과정을 지원한다.

가. 사이클 레벨 검증모드[3]

그림 2에서와 같이 사이클 레벨 모드에서는 사이클 단위로 데이터를 호스트 컴퓨터의 테스트 벤치에서 iPROVE의 하드웨어 상의 DUT로 전달을 한다. 이때 PCI 인터페이스는 자동적으로 생성되어 상관되는 DUT 입출력 포트에 값이 전달된다. 그리고 호스트 컴퓨터상의 테스트벤치에 사용하는 언어는 기본적으로 HDL (VHDL/Verilog) 언어 또는 C/C++ 언어를 지원하지만 C/C++ 언어를 사용하는 것이 HDL 을 이용하는 것보다 빠른 시뮬레이션 결과를 얻는다. 테스트벤치에 HDL을 사용할 경우에는 호스트 컴퓨터 상에서 시뮬레이션 툴을 이용하고 C/C++언어를 사용 하는 경우는 C/C++ 컴파일러를 통하여 iPROVE 하드웨어와 연동된다. 사이클 레벨의 장점은 사용자가 별도의 노력 없이 iPROVE 소프트웨어에서 요구 하는 사항들을 기입하는 것만으로 사용자의 디자인을 검증할 수 있다는 점이다.

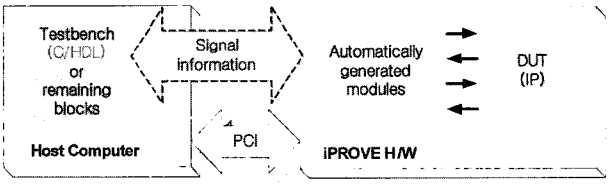

그림 2. 사이클 레벨의 검증 모드
Fig. 2. Cycle level verification mode.

나. 트랜잭션 레벨 검증모드[4]

그림 3에서 보여주듯이 트랜잭션 레벨 모드에서는 읽기, 쓰기 등의 커맨드를 호스트 컴퓨터의 테스트 벤치에서 DUT로 전달하는 방식으로 좀더 높은 abstract

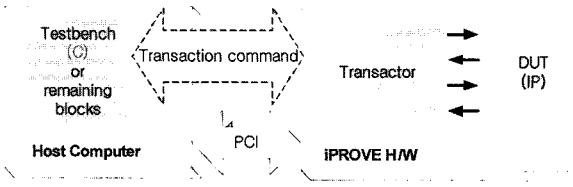

그림 3. 트랜잭션 레벨 검증 모드
Fig. 3. Transaction level verification mode.

level의에서 디자인을 검증하고 디버깅하는데 사용된다. 이때 호스트 컴퓨터상의 테스트벤치를 기술하는 언어로는 HDL언어는 사용되지 않고 오직 C/C++ 언어만 사용된다. 사이클 레벨 모드와는 달리 트랜잭션 레벨모드에서는 PIC 인터페이스가 자동으로 생성되지 않아 테스트벤치 에서 전달하는 커맨드를 해석하여 DUT의 상관되는 핀에 신호의 값을 전달하는 기능을 가지는 트랜잭터를 사용자가 반드시 추가 설계해야 한다.

(1) 트랜잭터 개요 및 구성

트랜잭터는 그림 4에서 보여주듯이 DUT와 iPROVE PCI 인터페이스 레이어 사이에 위치하며 C-port혹은 R-port를 통하여 각각의 FIFO에 저장되었던 데이터를 받고 W-port를 통하여 데이터를 WRITE FIFO에 전달함으로써 호스트 컴퓨터의 테스트벤치와 통신한다. 트랜잭터는 CMD FIFO나 READ FIFO로부터 데이터를 받고, 받은 그 데이터의 신호를 분석하고, DUT의 입출력을 위해서 분석한 신호의 순서를 결정 하는 기능들을 가져야 한다.

각각 FIFO의 데이터 32비트로 구성되며 CMD FIFO의 크기는 128, 256, 512이며 READ FIFO와 WR- ITE FIFO의 크기는 512, 1024로 iPROVE 소프트웨어 에서 선택적으로 결정한다. 이때 FIFO는 DUT와 같은 FPGA 상에 존재하게 되므로 적절하게 FIFO의 크기를

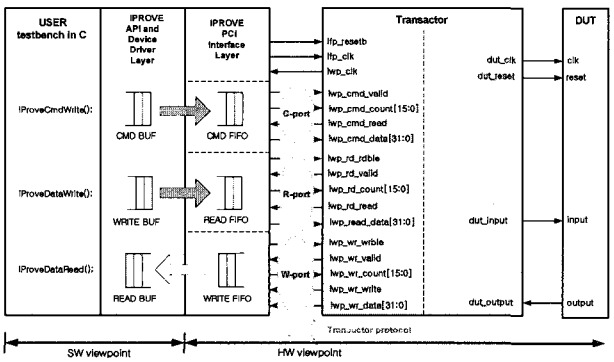

그림 4. 트랜잭터의 개념
Fig. 4. Concept of transactor.

사용자의 어플리케이션에 맞게 적절히 조절을 하여 사용해야 한다.

(2) 트랜잭터 프로토콜

그림 5에서와 같이 C-port의 데이터 전달 방법은 iwp_cmd_valid 신호가 high로 되면 CMD FIFO로부터 데이터를 읽는 것을 가능하게 하는 신호인 iwp_cmd_read를 high로 만들고 트랜잭터의 메인 클록인 iwp_clk의 동기에 맞추어 CMD FIFO로부터 저장된 데이터를 iwp_cmd_data로 샘플링한다. R-port 의 경우도 C-port와 유사하게 iwp_rd_rdble 신호가 high로 되었을 때 READ FIFO로부터 데이터를 읽어오는 것을 가능하게 하는 신호인 iwp_rd_read 신호를 high로 만들고 iwp_clk의 동기에 맞추어 iwp_rd_data로 샘플링한다. 그러나 W-port의 경우는 iwp_wr_wrble 신호가 high가 되었을 때 iwp_wr_write 신호를 high로 만들면 그 시점에 준비된iwp_wr_data 값은 WRITE FIFO에 전달된다.

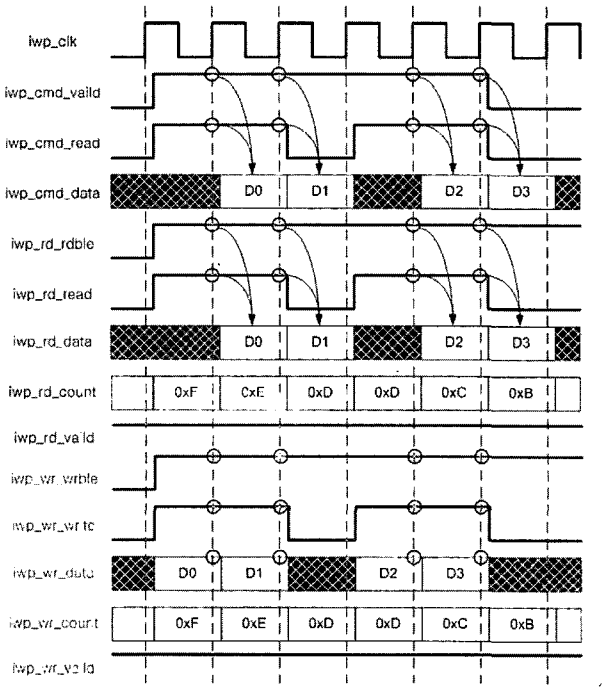

그림 5. 트랜잭터 프로토콜의 타이밍 구성도
Fig. 5. Timing diagram of transactor protocol.

2. 트랜잭터 정형화 및 구현

가. 트랜잭터 H/W 정형화

그림 6은 호스트 컴퓨터상의 테스트벤치에서 트랜잭터에 데이터를 쓰고 읽는 동작을 나타낸다. 처음으로

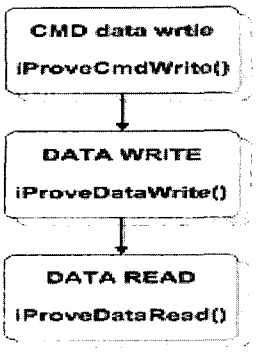

그림 6. 테스트벤치에서 데이터 흐름 과정
Fig. 6. Procedure of data transfer in testbench.

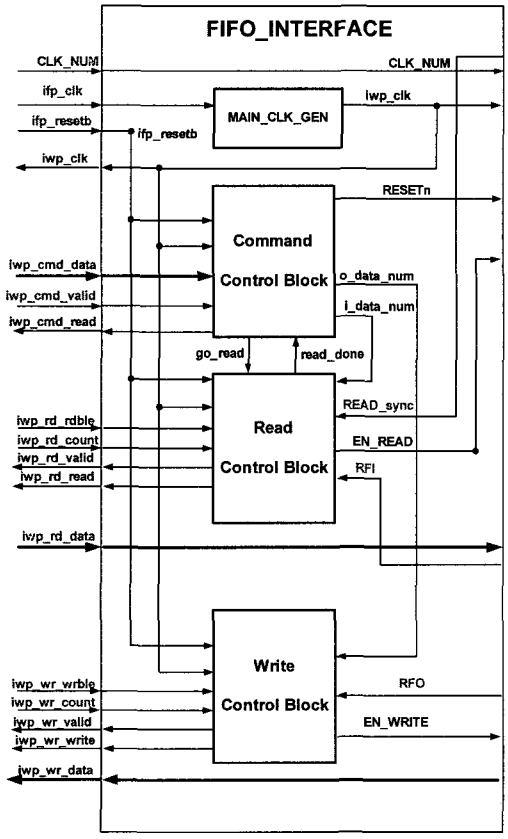

그림 7. 제안한 트랜잭터-1의 블록구성도
Fig. 7. Block diagram of the proposed transactor-1.

커맨드 데이터를 트랜잭터에 전달하여 트랜잭터 의 상태를 초기화한다. 커맨드 데이터에는 이후 데이터를 트랜잭터로 쓰고 읽을 개수를 내포하고 있다. 다음 과정으로 커맨드 데이터의 정보만큼 데이터를 쓰고 읽는다. 트랜잭터 역시 이러한 테스트벤치의 동작에 대응하는 구조로 설계 되어야 한다.

정형화된 트랜잭터의 전체 구조는 그림 7, 8과 같이 크게 프로토콜에 의존하는 FIFO 인터페이스 블록 과 DUT 인터페이스 블록으로 구성된다.

나. 트랜잭터 생성기 S/W 구현

(1) 개요 및 알고리즘

정형화한트랜잭터는 검증 적용대상인 DUT의 입출력 정보에 따라서 트랜잭터의 입출력 선언, 하위 블록 파생, 신호들의 비트 사이즈, Verilog의 구문 표현 등이 달라지기 때문에 사용자가 편집을 해야 한다. 이러한 사용자의 판단과 편집을 필요로 하는 부분을 소프트웨어가 사용자대신 추가 삽입하여 모듈을 생성 하고 생성된 모듈과 미리 준비된 모듈들을 합성하여 완성된 하나의 트랜잭터 코드를 만들어낸다. DUT의 입력정보에 따라 트랜잭터를 사용자의 편집 없이 자동으로 생성하는 생성기는 사용자가 DUT의 입출력 정보를 입력하면 미리 내장하고 있는 정형화된 트랜잭터 중 수정을 필요로 하는 모듈을 판별하고 입력된 정보에 따라 생성하고 내장된 다른 모듈들과 합성하여 하나의 트랜잭터 코드로 생성을 한다. 그림 9은 하나의 모듈을 생성하는 방법을 보여준다.

그림 10은 트랜잭터 생성기 소프트웨어의 순서도이다. 소프트웨어의 GUI를 통하여 DUT의 입력 포트와 출력 포트의 개수와 각각의 이름과 비트 사이즈를 입력

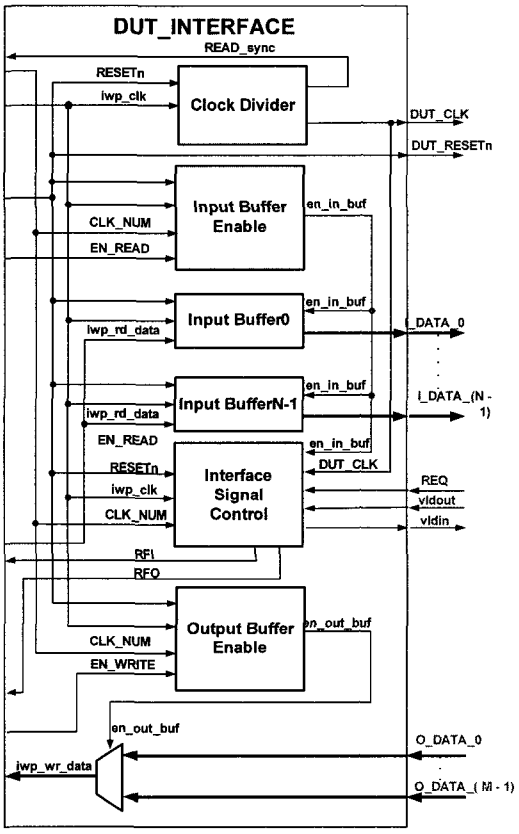

그림 8. 제안한 트랜잭터-2의 블록구성도
Fig. 8. Block diagram of the proposed transactor-2..

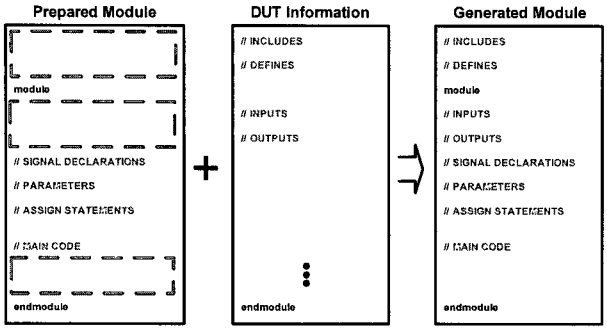

그림 9. 모듈 생성 방법
Fig. 9. Method of generating a module.

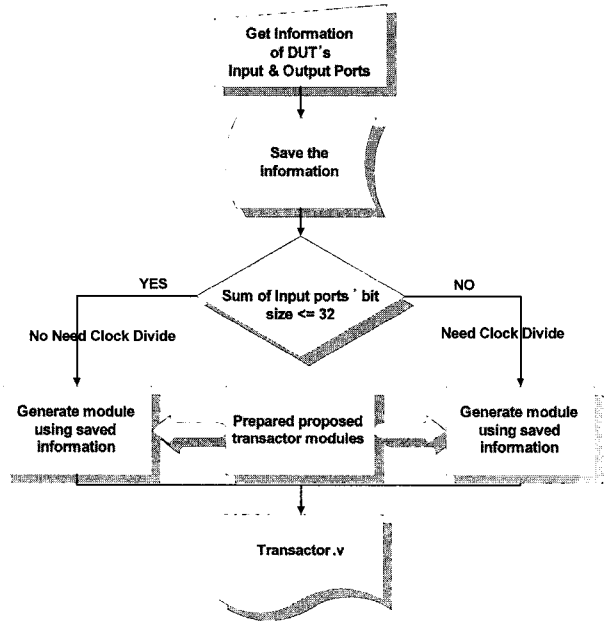

그림 10. 트랜잭터 생성 소프트웨어 흐름도
Fig. 10. Flow chart of transactor generation software.

받아 내부에 저장을 하였다가 DUT의 클록을 분주를 하여 사용할 것인지 아닌지를 판별한다. 이때 DUT의 전체 입력 포트들의 합이 FIFO의 폭인 32비트를 초과하는지 이하인지를 기준으로 판별한다. 판별 후 준비되고 정형화된 트랜잭터의 하위 모듈들 중에 편집을 필요로 하는 모듈을 저장해두었던 DUT 의 입출력 정보들을 삽입하여 생성하고 생성된 모듈들 과 그 외 준비된 모듈들을 하나로 합성하여 그 결과물로 사용자의 디자인을 iPROVE의 트랜잭션 레벨 모드로 검증함에 반드시 필요한 트랜잭터.v 코드를 생성한다.

(2) 검증대상에 따른 동작특성

DUT의 입력 포트들의 총 비트수의 합이 32 이하여서 분주된 클록이 필요 없는 DUT의 경우 FIFO 와 DUT간의 데이터 전달방식을 살펴보면 그림 11과 같다.

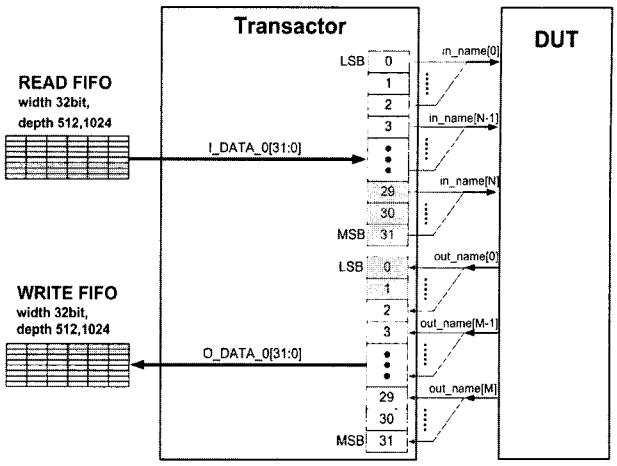

그림 11. 클럭 분주가 필요 없는 데이터 전달 개념
Fig. 11. Concept of data transfer without clock divide.

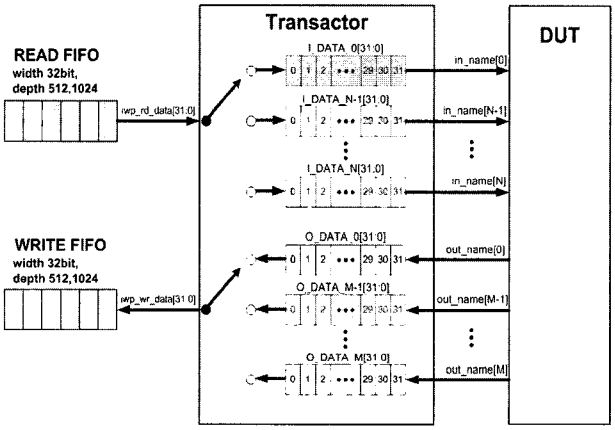

그림 12. 클럭 분주가 필요한 데이터 전달 개념
Fig. 12. Concept of data transfer with clock divide.

테스트벤치에서 FIFO를 통하여 transator에 전달되는 데이터는 하위부터 입력포트의 개수만큼 각각의 비트사이즈 만큼씩 채워져 전달되고, 트랜잭터 에서 DUT로 전달이 될 때는 다시 내부 버퍼인 I_DATA_0 [31:0]의 하위부터 DUT의 입력 포트의 순 서대로 각 비트 사이즈에 맞게 연결된다. DUT에서 테스트벤치로 전달되는 데이터 역시 동일방법으로 전달된다.

DUT의 입력 포트들의 총 비트수의 합이 32를 초과하여서 분주된 클록을 필요로 하는 DUT의 경우 트랜잭터를 통해 FIFO와 DUT간의 데이터 전달방식을 살펴보면 그림 12와 같다. 입력 포트들의 총 비트 사이즈가 32비트를 초과되면 FIFO의 크기가 32 비트이므로 한번에 각 DUT의 입출력 포트에 데이터를 전달할 수가 없다. 따라서 모든 입출력 포트들의 데이터는 비트 사이즈에 관계없이 32비트로 채워져서 FIFO 에 전달되고 FIFO의 동작이 의존되는 iwp_clk 마다 FIFO의 데이터는 각 32비트 사이즈를 갖는 버퍼에 스위칭되어 저

장된다. 이 저장된 데이터는 DUT 클록에 동기되어 DUT_CLK 한 사이클마다 DUT 의 각 입출력 포트에 전달된다.

III. 실 험

1. 생성된 트랜잭터 검증

생성기의 정상동작과 생성된 트랜잭터의 정상 동작을 확인하기 위하여 Pentium3 266Mhz, RAM 1G 인 컴퓨터 환경에서 그림 13과 같이 호스트 컴퓨터상의 Integer DCT와 Integer IDCT 블록을 제외한 H.264 영상 압축 코덱의 소프트웨어 모델과 FPGA상의 이미 검증된 Integer DCT와 Integer IDCT의 하드웨어 모델을 본 논문에서 구현한 생성기로 만들어진 트랜잭터를 사용하여 동작 실험을 하였다.

176x144 QCIF 영상 데이터를 입력으로 받아들여 호스트 컴퓨터상의 소프트웨어 모델과 하드웨어 모델과의 데이터 전송을 생성기를 통하여 얻어진 그림 14의 트랜잭터를 사용하여 예상되는 결과인 영상 결과물을 얻어냄으로써 제안된 생성기를 통해 생성된 트랜잭터의 정상동작을 입증하였다.

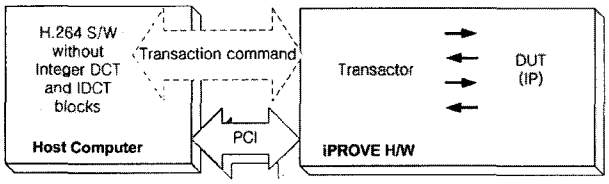

그림 13. H.264 블록 검증을 위한 개념도

Fig. 13. Concept of verification for DCT/IDCT block in H.264.

```
55
56 //****************** FIFO signals ********************
57
58 //********** C PORT **************
59
60     //input (CMD FIFO -> Transactor)
61 input           iup_cmd_valid;
62 input   [15:0]  iup_cmd_count;
63 input   [31:0]  iup_cmd_data;
64     //output (CMD FIFO <- Transactor)
65 output          iup_cmd_read;
66
67
68 //********** R PORT ***************
69     //input (READ FIFO -> Transactor)
70 input   [15:0]  iup_rd_count;
71 input           iup_rd_rdble;
72 input   [31:0]  iup_rd_data;
73     //output (READ FIFO <- Transactor)
74 output          iup_rd_valid;
75 output          iup_rd_read;
76     //********** W PORT **************
77     //input (WRITE FIFO -> Transactor)
78 input   [15:0]  iup_wr_count;
79 input           iup_wr_wrble;
80
81     //output (WRITE FIFO <- Transactor)
82 output          iup_wr_valid;
83 output          iup_wr_write;
84 output  [31:0]  iup_wr_data;
```

그림 14. 생성된 트랜잭터의 FIFO 신호 예

Fig. 14. Example of FIFO signals for generated transactor.

2. 생성된 트랜잭터의 특성 비교

표 1.은 다양한 하드웨어 IP들을 제안된 생성기로 얻어 낸 트랜잭터를 이용한 트랜잭션 레벨 검증 방법과 사이클 레벨 검증 방법 그리고 ModelSim(소프트웨어 시뮬레이터)를 이용하는 검증방법들을 각각 사용하여 시뮬레이션 속도를 측정한 결과이다.

검증된 실험 모두 트랜잭션 레벨 검증 방법을 사 용한 결과가 가장 빠른 속도를 보였으나 실험별로 각 기 다른 비율을 보이는 것은 검증대상인 하드웨어 블록의 복잡도와 연산량 그리고 IP 특성상 전송되는 데이터의 전달방법이 시뮬레이션 속도에 영향을 주기 때문이다.

표 2.는 6명의 학생(HDL 설계가능)들을 대상으로 트랜잭터의 기능과 프로토콜에 관하여 설명을 해준 뒤 각자 맡은 IP에 해당하는 트랜잭터를 설계 하도록 하였

표 1. 예제들의 시뮬레이션 속도 비교

Table 1. Comparison of simulation speed for examples.

	Modelsim	Cycle Level	Transaction Level	Note
16bit Adder	27.85 ms	15 ms	0.31 ms	1K pair
32x32 Multiplier	192.3 ms	172 ms	0.78 ms	1K pair
8bit GCD	291.1 ms	94 ms	16 ms	1K pair
8x8 2D DCT	12,070 ms	2,031 ms	109 ms	1 frame of qcif image
128 Point FFT	3,770 ms	31 ms	0.16 ms	128 point
H.264 Integer DCT	10.243 ms	1.831 ms	82 ms	1 frame of qcif image

표 2. 생성된 트랜잭터의 효용성

Table 2. Efficiency of generated transactor.

	Design Time (hour)		Logic Size (LUT)		Operating Frequency (Mhz)	
	M	A	M	A	M	A
16bit Adder	7	0	66	136	868.1	177.1
32x32 Multiplier	4.5	0	136	142	NA	132.9
8bit GCD (Greatest Common Divisor)	4.5	0	2	111	566.3	205.5
8x8 2D DCT	6	0	13	111	566.3	179.8
128 Point FFT	5.5	0	116	151	108	146.6
H.264 Interger DCT	6	0	12	105	589.1	164.4
	Depend on user		Total LUT in FPGA(Virtex1000E) : 24576 LUTs		Required frequency : 33 Mhz	

M: Manual designed transactor

A: Automatic designed transactor

을 때 걸리는 시간과 직접 트랜잭터를 설계한 것과 구현한 생성기를 통하여 생성된 트랜잭터와의 크기 및 최대 동작 주파수 등의 특성을 비교한 결과를 보여준다. 사용자가 직접설계를 하는데 걸리는 시간은 사용자의 경험과 하드웨어 IP의 이해도에 따라 편차가 있을 수 있으나 각각의 하드웨어 IP에 대하여 사용자가 직접 설계하는데 소요되는 시간은 평균적으로 5.6시간이 소요되지만 생성기를 이용하여 트랜잭터를 생성하는 경우 해당 DUT의 입출력 정보를 입력하는 시간 인 2분 이하의 시간이 소요된다.

두 가지 트랜잭터를 이용한 시뮬레이션 속도의 차이는 데이터의 전달방식의 차이와 호스트 컴퓨터 상 의 테스트벤치를 작성하는 방법에 따라서 달라질 수 있으나 동일한 검증환경을 사용하였기 때문에 표 1.의 결과와 같은 동일한 결과를 보인다.

두 가지 트랜잭터의 크기는 생성기를 통하여 얻은 트랜잭터의 경우 비슷한 크기를 가지고 있고 사용자가 직접 설계한 트랜잭터가 적은 크기를 가지지만 iPROVE에 사용되는 FPGA중 가장 작은 게이트 사이즈를 가지고 있는 Virtex 1000E의 경우 24,576개의 LUT (Look-Up 표) 중 모두 1% 미만을 차지하므로 DUT와 함께 FPGA 상에서 동작함에 있어서 트랜잭터의 크기로 인해 DUT를 검증할 수 없는 문제는 야기하지 않는다. 트랜잭터 의 최대동작 주파수도 역시 대체적으로 사용자가 직접 설계한 트랜잭터가 빠르지만 *iPROVE* 에서 동작시키기 위해서 요구되는 동작스피드인 33Mhz에 모두 적합하다.

IV. 결 론

본 논문에서 구현한 생성기로 얻은 트랜잭터의 정상 동작을 확인하기 위하여 이미 검증된 H.264 영상 코덱 소프트웨어 모델과 Integer DCT, Integer IDCT 하드웨어 블록을 이용하여 검증하였다. 그리고 사용자 가 직접 설계한 트랜잭터와의 설계시간, 시뮬레이션 시간, 면적, 최대 동작 스피드 측면 비교를 통하여 자동 생성기로 생성된 트랜잭터의 효율성을 입증하였다.

트랜잭터 생성기는 사용의 편리함을 제공하고 사용자가 트랜잭터 프로토콜을 이해하고 트랜잭터를 설계하는 부담을 줄여 트랜잭션 레벨 검증방법을 용이하게 사용 할 수 있도록 하여 얻게 되는 시뮬레이션 이득과 검증하려하는 디자인의 특성과 사용자의 설계 경험에 따라 다소 차이는 있으나 약 5시간 반 정도의 트랜잭터 설계 시간을 합한 만큼의 시간을 총 설계 및 검증에 필요한 시간에서 단축할 수 있었다.

참 고 문 헌

[1] Wooseung Yang, Moo-Kyeong Chung, and Chong-Min Kyung, "Status and challenges of SoC verification for embedded systems market." SOC Design Conference, - 216. 2003.

[2] Moussa I, Grellier T, and Nguyen G, "Exploring SW performance using SoC transaction-level modeling." Design, Automation and Test in Europe Conference and Exhibition, pp.120 - 125. 2003.

[3] Dynalith, Inc., *iPROVE User Manual*, Seoul. 2006.

[4] Dynalith, Inc., *iPROVE Transactor Design Guide*, Seoul. 2006.

저 자 소 개

조 중 휘(정회원)
1981년 한양대학교 전자공학과 공학사
1983년 한양대학교 전자공학과 공학석사
1986년 한양대학교 전자공학과 공학박사
2003년～현재 인천대학교 멀티미디어시스템 공학과 교수
<주관심분야: 영상처리SoC설계, 설계방법론>

조 종 현(정회원)
2003년 인천대학교 전자공학과 공학사
2005년 인천대학교 전자공학과 공학석사
<주관심분야 : 영상처리SoC설계>