우선순위 폴링방식의 중재교환 제어기 설계

정회원 김 동 원*, 김 도 영**, 신 현 식***

A Design of a Switching Arbiter using a Priority Polling Scheme

Dong Won Kim*, Do Young Kim**, Hyeon Sik Shin*** *Regular Members*

요 약

본 논문은 개방형 대용량 통신처리시스템에서 사용되는 내부 고속 연동망 시스템 중 중재교환 제어기의 설계 및 구현에 관한 기술 내용이다.

가입자 입출력부의 서비스 요구에 대응하는 FIFO 버퍼의 상태를 폴링할 때 각 포트에 따라 우선 순위를 부여하고, 부여된 우선 순위와 함께 해당 포트의 서비스 원칙도 함께 설정할 수 있도록 우선순위 폴링방식의 중재교환제어기 설계를 하여 다양한 미디어나 망 정합 장치의 특성에 따른 중재교환 서비스가 가능하도록 하였다.

중재교환제어기는 FPGA를 사용하여 구현하였으며 설계 방법적인 측면에서는 현재 가용 가능한 개념 수준의 설계 입력 수단부터 직접 VHDL로 코딩 입력까지 다양한 방법의 수단을 복합적으로 사용하였다. 설계 입력 후 검증은 VHDL 시뮬레이터를 이용하여 검증하였고 구현 결과는 타이밍까지 반영된 VHDL 출력을 back annotation하여 설계 당시 VHDL 시뮬레이터에서 검증 하였다. 구현에 사용된 FPGA는 62,000 사용 가능 게이트(usable gate) 급의 FLEX10K100이며 약 66%의 자원을 사용하였다.

ABSTRACT

This paper describes design and implementation of the switching arbiter based on priority polling concept, which is the core part of the internal high-speed switching fabric of open-architecture AICPS(Advanced Information Communication Processing System).

The priority polling scheme enables versatile switching capabilities for the multimedia applications and flexibility for the characteristics of the network interface. The proposed scheme was designed to allow the priority to each switch port and service discipline as well when polling the FIFO status to check the requested service from subscriber input/output parts.

We implemented the switching arbiter using FPGA(Field Programmable Gate Array) and mixed design technology, which comprises top-down approach from the behavioral-level design of algorithm to manual VHDL coding implementation. Also we verified the design using VHDL simulator and finally confirmed the result in the original VHDL simulator using the VHDL output containing time-analysis information via back annotation. As the result of the implementation, about 66% of gate in FLEX10K100 FPGA device (62,000 usable gates) was used.

I. 서 론

대용량 통신처리시스템(AICPS : Advanced Information Communication Processing System)[1]은 기존의 01410 하이텔 서비스에 사용중인 통신처리시스템을 고속으로 대용량화 하여 복수의 VAN 사업자들을 동등 접속시키

* 한국전자통신연구원 초빙연구원겸 옥천전문대 정보통신과(dwkim@occ.ac.kr) 정회원
** 한국전자통신연구원 통신처리팀 선임연구원(dyk@etri.re.kr) 정회원, *** 임프레스 정보통신(hsshin@imtresstek.co.kr) 정회원
논문번호 : 98016-0913, 접수일자 : 1998년 9월 13일

고, 향후 널리 사용될 ISDN과 프레임릴레이망, B-ISDN 등을 용이하게 수용하여 인터넷 서비스까지 확장할 수 있도록 개방형 구조를 기반으로 개발중에 있다. 대용량 통신처리시스템에서 앞으로 널리 사용될 다양한 망들을 용이하게 수용하고 부가적인 정보통신 서비스를 개발 탑재하기 위해서는, 내부 고속연동망을 중심으로 다양한 망정합 모듈과 프로토콜 처리모듈들이 연동되는 모듈화 및 분산처리 구조를 가지는 것이 가장 효율적인 것으로 검토되었다[1,2].

이러한 구조연구를 토대로 설계된 내부 고속 연동망은 공유버스를 기반으로 32개의 가입자 입출력부 노드가 접속되어 있으며, 고속의 중재교환부를 통해 각 노드로 부터 순차적으로 패킷을 전송하며, 패킷전송 중에 다음 대기 노드를 찾는 우선순위 폴링방식의 중재를 해나가는 병렬처리를 하여 버스사용률을 개선하였고, 동축 케이블을 이용하여 직렬로 근거리의 분산 노드들을 성형으로 접속할 수 있는 병렬 공통버스 기반의 패킷교환방식인 내부 고속연동망(HSSF : High Speed Switching Fabric)을 설계하였다. 설계된 공유버스의 스위칭 대역폭은 640Mbps이며 각 노드별 할당 가능한 대역폭은 모든 노드가 동등한 우선순위를 가지며 약 20Mbps가 된다[3,4].

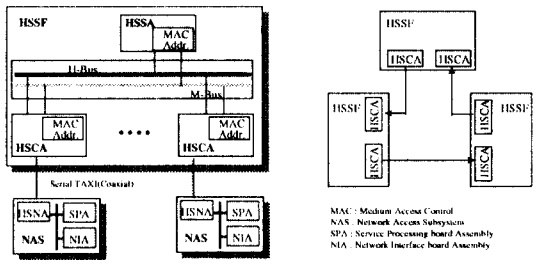
본 논문에서는 기존의 HSSF가 모든 노드들이 동등한 트래픽이 부가 될 때 각 노드별로 약 20Mbps 정도의 대역폭을 할당할 수 있는 순환 폴링 중재교환방식을 사용하던 것을 실제적으로 트래픽 부하가 큰 노드(예를 들어 OAM과 연결되는 노드 및 ATM 정합부 노드 등)에게는 보다 더 많은 대역폭을 할당하고 상대적으로 트래픽 부하가 적은 노드들에게는 대역폭이 덜 할당되도록 하는 우선순위 폴링 중재교환방식으로 개선을 하였다. 또한 실장밀도를 높이기 위해 단일 FPGA소자로 구현하는데 따른 우선순위 폴링방식[5,6]의 중재교환제어기의 설계 방법론 및 검증 방법과 결과에 대해 기술하였다.

설계 방법적인 측면에서는 현재 가용 가능한 개념 수준의 설계 입력 수단부터 직접 VHDL로 코딩 입력 까지 다양한 방법의 복합적 수단(mixed methodology)을 사용하였다. 설계 입력 후 검증은 VHDL 시뮬레이터를 이용하여 검증하였고 구현 결과는 타이밍까지 반영된 VHDL 출력을 back annotation하여 설계 당시 VHDL 시뮬레이터에서 검증 하였다[7-9]. 특히, 현재 주요 VHDL 설계는 RTL(Register Transfer Level) 수준으로 이루어 지고 있으나 좀더 개념(behavioral) 수준에서 코딩한 것을 툴을 이용하여 자동적으로 RTL 코드를 생성하여 합성하는 방법으로 일부 블록에 도입하였다. 즉, 구조(architecture)를 자동적으로 만들어 주는 툴의 도움을 받아 설계 기간을 단축하였다. 최근, 많이 소개된 그래픽 입력(graphical entry)툴[7]들을 이용하여 블록마다 상태도(state diagram), 흐름도(flow chart), 진리표(truth table)들을 이용하여 입력한 후 합성 가능한 VHDL 코드를 생성하였고 일부 블록은 VHDL로 매뉴얼 코딩[11]한 것을 사용하였다. 설계 검증을 위하여 각 블록별로 서로 다른 설계 입력 방법을 사용하였으나 전체 블록을 통합하여 VHDL 테스트벤치(testbench)에 의한 시뮬레이션을 수행하였으며 개념수준, RTL 수준 및 합성결과인 게이트 수준들이 모두 동일함을 검증하였다. 구현은 62,000 사용가능 게이트급의 FPGA인 FLEX10K100[10]을 사용하였고 약 66%의 자원을 사용하였다. 향후 ASIC으로의 용이한 재제작(retargetting)을 위해 본 디바이스의 내장된(Embedded) RAM은 사용하지 않았다.

II. 중재교환제어기 전체 구조

우선순위 폴링방식을 적용하고 단일소자로 개선 구현한 중재교환제어기의 이해를 돕기위해 먼저 내부 고속연동망인 HSSF의 전체 구조 및 동작을 간략히 살펴보고자 하며 보다 상세한 구성과 동작 설명은 참고문헌[3]에 소개되었다.

그림 1에서는 공유버스를 기반으로 한 고속 패킷 교환방식을 채택하고 있는 HSSF의 전체 구성도를 나타낸다. HSSF는 각 가입자 노드들과 직렬접속되어 패킷교환의 입출력을 담당하는 가입자 입출력부(HSCA : High Speed Channel Adaptor)와 입출력가입자부들간에 공통매체인 공유버스 사용권을 중재하고 패킷데이타들의 전달을 담당하는 중재교환부(HSSA : High Speed



(a) 단일 구조 (b) 확장 구조

그림 1. HSSF 구조
Fig. 1 HSSF Structure

Switching Assembly)와 패킷데이타들의 실제 수송로 역할의 데이타버스 및 입출력 가입자부와 중재교환부 사이의 제어신호버스 및 주소신호버스로 구성된 H-bus와 채널의 상태 및 장애관리를 위한 M-bus로 구성되는 병렬공통버스부의 3개 기능부로 구성된다.

● 프레임 형식

프레임의 형식은 그림 2(a)와 같이 최대 2048 바이트의 크기를 갖는 가변길이의 구조이며 내부는 헤더(header) 4 바이트와 트레일러(trailer) 4바이트는 현재 프레임에 대한 CRC-32 영역으로 구분된다.

공통버스 상에서 프레임이 전송되는 구조는 그림 2(b)에 나타낸 바와 같이 64비트의 병렬 데이타 버스상으로 헤더를 포함한 첫 프레임 데이타와 동시에 각 프레임을 구분하는 태그 버스상으로 SOF/COF/EOF의 태그 정보가 전송되며 이어 유효 데이타와 끝으로 CRC 영역과 마지막 데이타 4바이트가 전송되어 데이타의 길이에 따라 최대 256번의 전송이 이루어지게 된다.

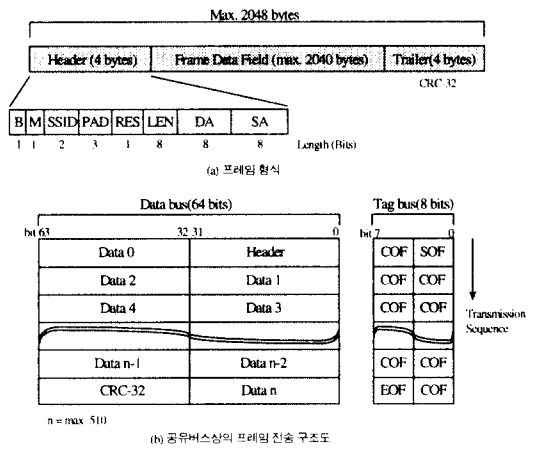


그림 2. 프레임 형식 및 H-Bus상의 전송 구조
Fig. 2 Frame format and transmission structure on H-Bus

● 중재교환 및 관리부(HSSA)

중재교환부는 H-bus를 통해 각 가입자 입출력부의 서비스 요구상태 파악 및 버스사용권 중재를 위한 폴링중재동작기능과 패킷헤더분석 및 데이타 교환전달의 기능을 담당하며 본 논문에서 소개하고자 하는 중재교환제어기가 위치하는 부분이다. 관리부는 M-bus를 통해 각 HSCA의 채널 상태를 감시하는 기능을 수행한다.

● 가입자 입출력부(HSCA)

가입자 입출력부는 가입자 노드와는 TAXI로 연결되어 직렬 통신을 하고, H-bus를 통해 패킷 교환이 될 때까지 패킷을 저장하는 버퍼 역할을 한다.

● 가입자노드 어댑터(HSNA)

가입자노드 어댑터는 HSSF를 통하여 통신을 원하는 각각의 망정합모듈을 구성할 때 사용되는 네트워크 카드로서 망정합모듈에서 처리된 프로토콜 데이타 유니트(PDU : Protocol Data Unit)를 라우팅테이블정보에 따라 패킷 헤더를 HSSF 패킷 포맷으로 변환하고 CRC 생성/검사를 수행하여 HSSF 내부 프로토콜에 따라 PDU를 송수신하는 역할을 한다. 따라서 실제적으로 HSNA에서는 HSSF 내부고속 연동망에서 규정한 계층 3네트워크계층 기능과 계층 2 링크계층 기능 및 물리계층 기능을 전담한다.

● 중재교환부의 동작 개요

패킷의 생성, 소멸이 이루어 지는 곳은 HSNA를 갖고 있는 망정합모듈(노드) 이다. 이곳에서 생성된 패킷은 1 : 1로 대응되는 HSCA로 보내어 지며, 원하는 노드로 전달되기 위해서는 중재교환부의 서비스(H-bus의 사용권 허가) 차례가 될 때까지 HSCA내의 FIFO 버퍼에 대기한다. 중재교환부의 폴링 때 대기된 패킷들을 갖고 있는 HSCA는 서비스 요청을 하며 이때 중재교환부에서 패킷헤더의 분석을 통해 소스(source) HSCA와 목적지(destination) HSCA 간의 메모리 읽기와 쓰기 동작에 의한 데이터 교환 전달이 되도록 적절한 제어 신호들을 발생시킨다.

이러한 중재교환부의 기능을 HBUSS_CNTR라 불리우는 중재교환제어기 하드웨어에서 수행을 하며, 이는 POLLING 블록, 데이터 교환 전달을 처리하는 MOVING 블록, 헤더의 내용을 분석하여 POLLING 블록과 MOVING 블록에 적절한 제어신호를 송출하는 DECODER 블록, 폴링 순서와 서비스원칙을 규정하는 폴링테이블을 소프트웨어로 설정하고 내부 하드웨어 블록들의 제어와 상태를 검사하기 위한 CPU_INTF와 SA_LTCH 블록으로 구성된다.

III. 우선순위 폴링방식 중재교환제어기 설계

개선된 우선순위 폴링방식 중재교환제어기는 기존의 중재교환제어기가 가입자 입출력부의 서비스 요구에 해당하는 FIFO 버퍼의 상태(전송요구신호)를 폴링할 때 모든 가입자 노드를 동등한 등급으로 순환식

(circular or round robin)으로 서비스를 제공하는데 비해, 각 포트에 따라 우선 순위를 부여하고 부여된 우선 순위와 함께 해당 포트의 서비스 원칙(policy)도 함께 설정할 수 있도록 하여 다양한 미디어나 망정합 장치의 특성에 따른 중재교환 서비스가 가능하도록 하였다. 우선순위 폴링 방식으로는 별도 폴링할 순서와 횟수를 운용자 임의로 제어 가능한 폴링테이블을 참조하는 구조를 적용하였고 각 포트에 대한 폴링 서비스 모드도 전송할 데이타가 없을 때 까지 서비스를 해주는 완전서비스(exhaustive service) 원칙 혹은 한 번 폴링에 단 한 개의 데이터만 서비스하는 제한서비스(limited-1 service) 원칙 등의 등급을 구분할 수 있도록 하였다.

3.1 설계 방법

설계시 기본 방침은 VHDL 기반의 설계를 주로 하는 것이며, 기존 설계 된 라이블러리(design library)들을 응용하는 수준의 설계는 VHDL 코딩 방법을 사용하며, 기타 블록들은 그래픽 형태의 입력 수단을 이용하되 블록의 특성에 맞는 방법을 이용하였다. 설계의 상위 블록인 HBUSS_CNTR 블록은 블록 다이애그램(block diagram)으로, POLLING 블록은 클럭상태(clock state)들을 가미한 흐름도로, MOVING 블록은 유한 상태도로, DECODER 부분은 진리표 입력 방법으로, 기타 CPU_INTF 및 SA_LTCH블럭들은 매뉴얼 VHDL 코딩으로 설계 하였다. 따라서 설계 시간의 절감, 설계의 이해도 증진, 차후의 설계변경 발생시 편리한 기술 자료화 등의 효과를 가져왔다.

그림 3에서 나타낸 바와 같이 각 입력 방법에 따른 입력된 설계의 각 단계별 기능 검증 방법과 구현까지의 흐름을 보였다. 주로 사용된 설계 툴들은 그래픽 형태 입력 후 VHDL 코드를 자동 생성해 주는 Summit Design 사의 Visual_hdl[7]을 사용하였고, VHDL simulation을 위해 VSS(VHDL System simulation), 개념수준 합성(Behavioral Synthesis)를 위한 Behavioral Compiler, RTL 수준 합성을 위한 Design compiler 등은 synopsys사의 툴[8]들이며, 최종적으로 구현을 위해 Altera 사의 Maxplus2[9]를 사용하였다.

특히 Behavioral Compiler의 사용으로 기존 개념수준의 시뮬레이션을 위한 코드를 조금만 수정한다면 바로 합성 단계로 이어질 수 있어 설계 시간을 줄여 줄 수 있는 방법이라 할 수 있다.

3.2 H-bus controller(HBUS_CNTR) 블록 설계

HBUS_CNTR 블록의 구성도를 그림 4에 나타내었

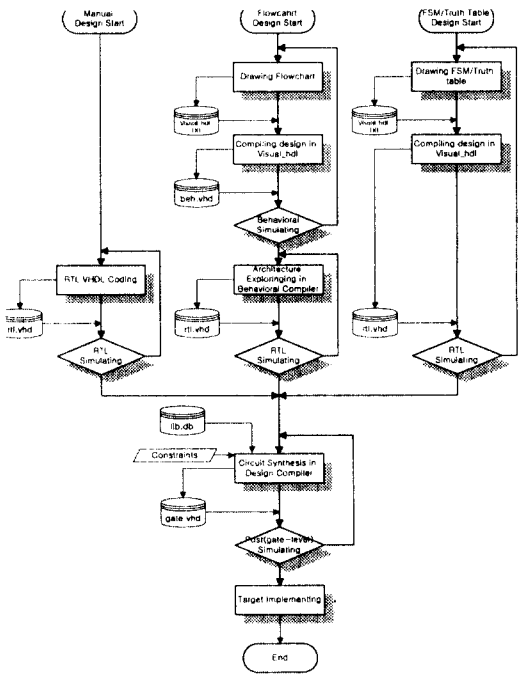
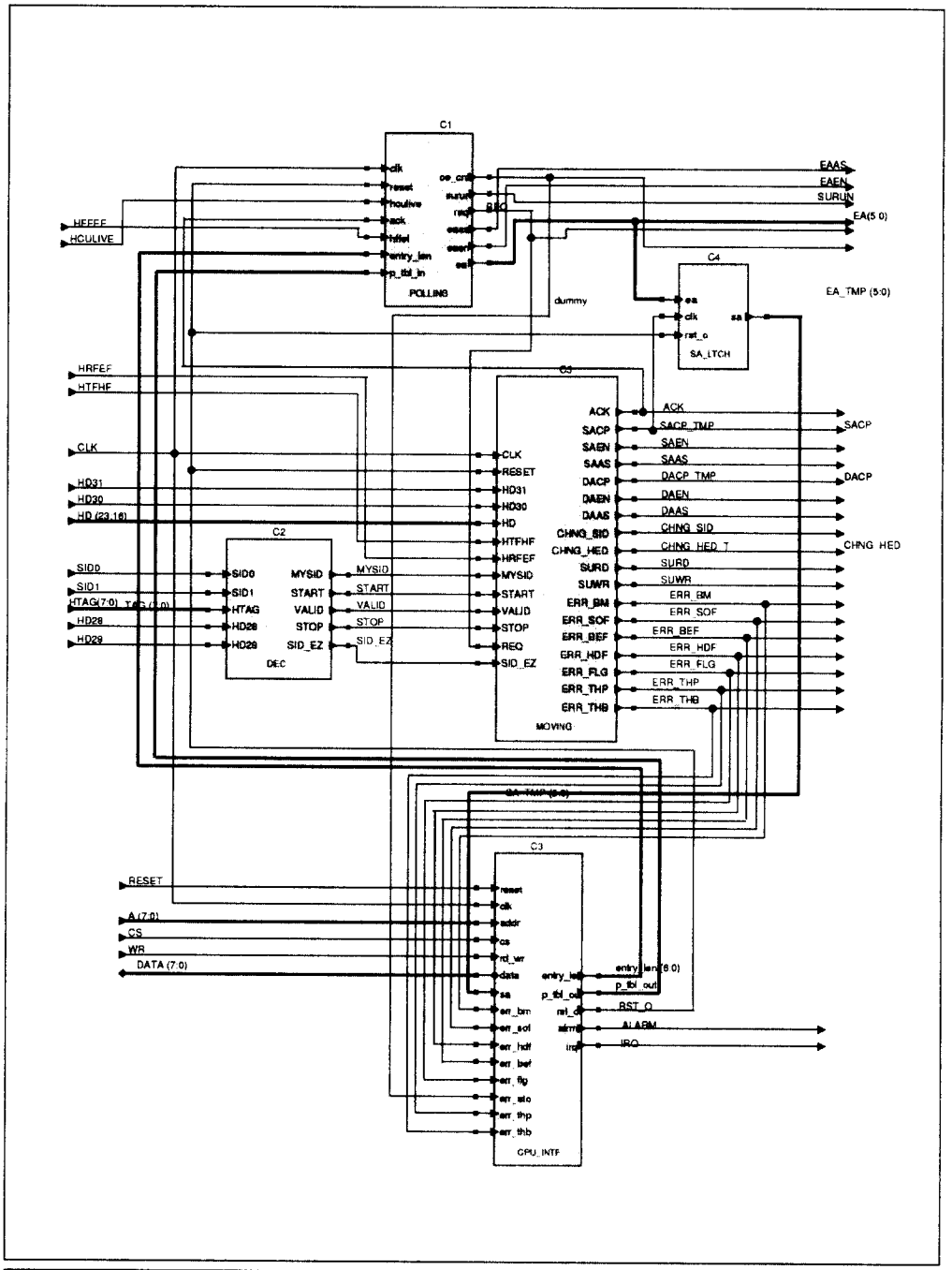


그림 3. 설계 흐름도
Fig. 3 Design flow

다. 프레임 데이타의 교환 및 전달 통로인 H-Bus에서 먼저 순서대로 폴링할 주소를 갖고 있는 폴링테이블에 따라 각 HSCA내의 FIFO 버퍼의 상태인 프레임의 전송 요구 신호(/HFFEF)를 연속적으로 검사하여 MOVING 블록에게 교환 전달을 요구하는 POLLING 블록, 전송 요구된 채널에 대해 프레임의 헤더를 분석하고 연속되는 프레임 정보를 목적지로 전달하는 MOVING 블록, 이 MOVING 블록의 기능을 수행하도록 프레임의 순서 정보(시작, 연속 및 끝)와 자신의 ID를 비교 해석하는 DECODER 블록, 폴링할 순서대로의 주소정보에 대한 레지스터, 전체 기능의 동작을 초기화 또는 계속하도록 제어하는 레지스터, 및 인터럽트 처리를 위한 상태와 제어 레지스터와 교환전달 과정에서 일어나는 오류 상태에 대한 제어 프로세서로의 보고 또는 확인 처리기능을 내장한 CPU_INTF 및 오류가 발생한 채널의 원천 주소(Source Address)를 일시 저장하여 제어 프로세서의 오류 처리를 도와주는 SA_LTCH (Source Address Latch) 등의 기능부로 구성된다. 전체 구성은 그림 4와 같이 블록도로 입력되었고, 각 기능 블럭의 설계 방법은 유한 상태도, 흐름도, 진리표, 및 VHDL 코딩등의 복합적인 방법으로 수행되었다.



Design: hbus_alt:Hbus_cntr	Date: Tue Jul 29 16:46:20 1997	Page: 1

그림 4. HBUS_CNTR의 블록도
Fig. 4 Block diagram of HBUS_CNTR

3.2.1 POLLING 블록

32개의 가입자 입출력부(HSCA)의 버퍼에 하나 이상의 정상 프레임이 입력되면 프레임 전송요구신호(HFFEF)를 보내게 되는데, 해당 채널의 이 전송요구신호의 상태를 확인하면 버스를 점유하여 전송할 수 있는 권한을 부여하기 위한 폴링 동작이 이루어진다. 가입자 채널의 트래픽 특성이나 서비스 특성을 고려한 패킷 교환을 위해 외부 호스트 인터페이스(CPU_INTF)를 통해 폴링테이블 및 레지스터의 값을 적절히 설정해 주어야 한다. 그림 5의 구조와 같이 여기서는 폴링주소 및 서비스 모드를 설정하는 POLL_SEQ_ENTRY 테이블 레지스터와 유효한 테이블의 길이 정보인 POLL_TBL_LENGTH 레지스터를 반드시 초기에 올바로 설정되어야 한다. 7 비트의 POLL_SEQ_ENTRY는 모두 128개가 있으며 이는 서비스 할 포트의 주소 영역과 각 포트(채널)의 연속 서비스 횟수를 결정하는 모드 영역으로 구성된다. 주소 영역에는 폴링 할 물리적인 주소를 적어주면 되며 0~31사이의 값으로 관리자 임의의 우선순위를 고려한 순서대로 주소들을 각 테이블에 써 줄 수 있으며 모드 영역은 그림 6과 같이 해당 주소에 대한 연속하는 폴링 횟수를 기록한다. 현재는 FIFO 버퍼에 유효한 데이타 프레임이 존재하지 않을 때 까지 연속하여 서비스하는 완전 서비스 모드와 단 1회만 서비스하도록 하는 제한서비스(limited-1) 모드만을 사용한다. 테이블의 구성을 완료한 후 전체 테이블의 길이를 설정하는 POLL_TBL_LENGTH 레지스터를 설정하며 실제 (유효테이블 개수-1) 만큼의 값으로 설정해야 한다.

이렇게 레지스터의 초기화가 이루어지고 나면 POLLING 블록은 폴링주소(EA)와 폴링주소 스트로브(EAAS)

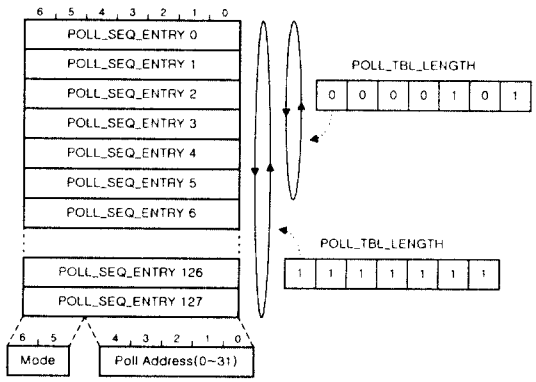


그림 5. 폴링 테이블 구조
Fig. 5 Polling table structure

Mode	Description
00	Exhaustive : 1회 폴링에 대기중인 모든 패킷을 모두 서비스
01	Limited-1 : 1회 폴링에 하나의 패킷만 서비스
10	Limited-2 : 추후 확장 모드
11	Limited-3 : 추후 확장 모드

그림 6. 폴링 서비스 모드
Fig. 6 Polling service discipline

를 활성화시켜서 첫번째 테이블의 채널에 대해 전송요구신호 상태를 확인하게 되며 유효한 프레임이 있음을 확인하면 MOVING 블록 측으로 /REQ 신호를 보내어 무빙을 요구하게 된다. 이어 MOVING 블록으로부터 /REQ신호에 대한 응답으로 /ACK신호를 받게 되면 현재 첫번째 채널의 전송요구가 인지되어 무빙중임을 확인한다. 이어 테이블의 모드 영역을 참조하여 연속하여 폴링 할 것인지를 결정하며 모드의 값이 '0'인 경우 유효한 프레임이 FIFO에 남아있지 않을 때 까지 연속하여 현재의 주소로 계속 폴링하게 되며, '1'의 값인 경우 다음 테이블을 참조하여 새로운 주소를 활성화시켜 폴링하게 된다. 여기서 POLLING 블록은 현재의 POLL_SEQ_ENTRY 값과 테이블 길이 정보를 담고있는 POLL_TBL_LENGTH 레지스터와 비교하여 값이 일치하면 한 주기의 폴링 싸이클을 완료한 것으로 간주하고 폴링테이블의 첫번째 ENTRY부터 다시 시작하여 순환방식으로 폴링 작업을 하게 된다.

이상의 폴링 과정을 흐름도 입력 방법을 사용하여 그림 7과 같이 설계하였으며 각 상태의 설명은 다음과 같다.

- INIT_ST(초기화 상태) : 모든 출력신호에 대한 초기 상태를 정의한다.
- Algorithm Loop와 K_loop사이에서는 폴링주소(EA)와 폴링주소유효신호(EAEN)를 출력하게 된다.
- P_STRT_ST (폴링시작 상태) : 폴링주소스트로브 신호(EAAS)를 출력
- WAIT_EF_ST(전송요구신호폴링 대기상태) : 현재 폴링주소에 대한 전송요구신호가 유효한 시점까지 대기하는 제한된 클럭만큼(현재 3 clock) QUEUE_CHECK_LOOP를 돌도록 하고 있다.
- CHK_EF_ST(전송요구신호 폴링 상태) : 전송요구신호가 유효한지를 판단하는 상태로 전송요구상태라면 MV_REQ_ST로 천이하여 REQ신호를 출력하게

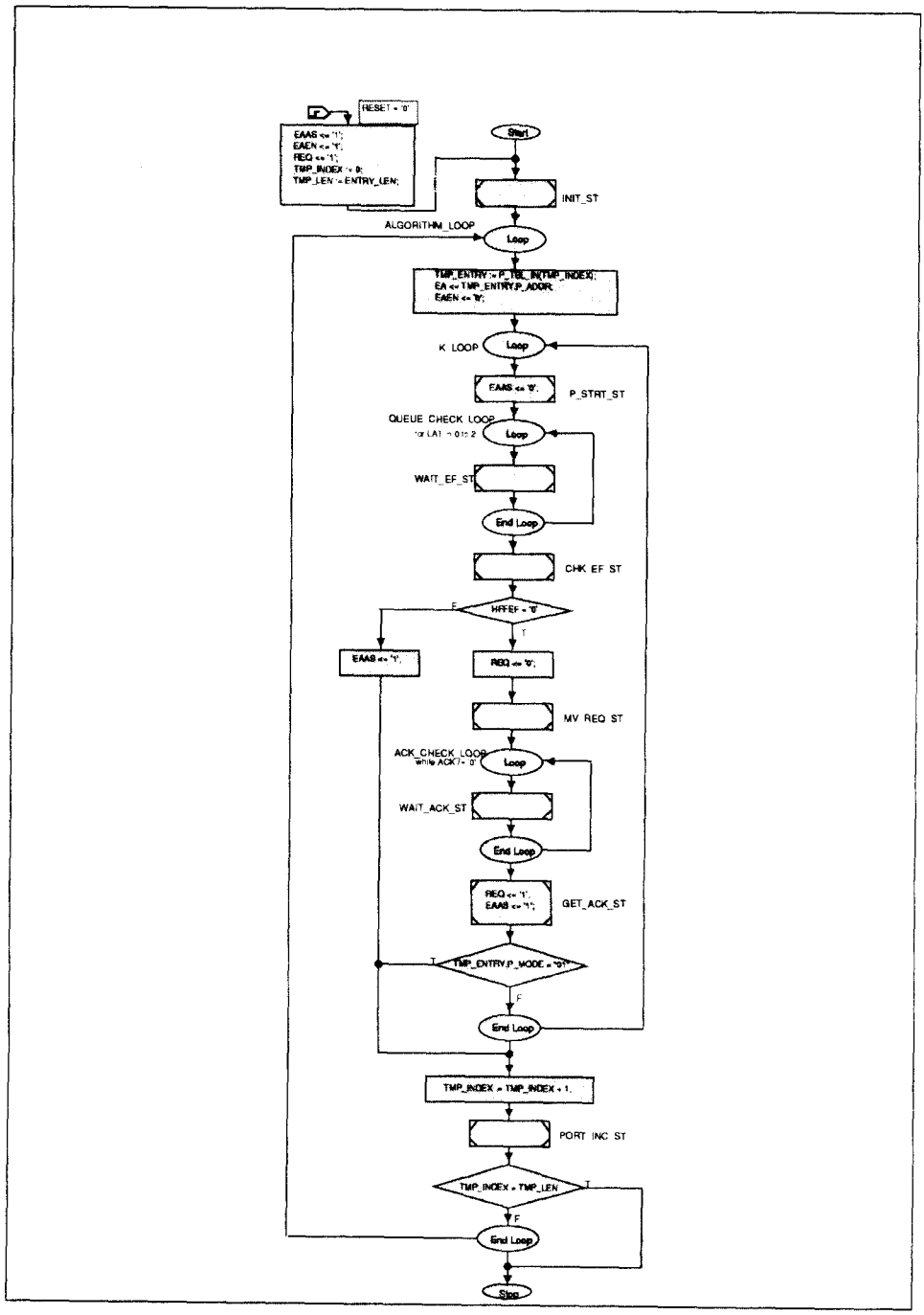


그림 7. POLLING 블록 흐름도

Fig. 7 POLLING block flow chart with clock states

되며 전송요구가 없으면 폴링주소 스트로브를 원상태로 하면서 다음 HSCA 포트를 지정하도록 하는 인덱스를 증가하는 루틴으로 점프하게 된다.

- MV_REQ_ST(전송/교환 요구 상태) : REQ 신호를 유효하게 출력하여 MOVING 블록에게 교환 행위가 일어나도록 요구하는 상태이다.
- WAIT_ACK_ST(응답 대기 상태) : REQ신호에 대해 MOVING 블록으로부터 응답 신호인 ACK가 있는지를 확인하는 상태로 ACK가 올 때까지 ACK_CHECK_LOOP를 돌게 된다.
- GET_ACK_ST(응답신호 인지 상태) : 응답신호를 인지하게 되면 REQ신호와 폴링주소스트로브를 원상태로 만들어 다음 채널에 대한 폴링을 준비하도록 한다.
- PORT_INC_ST(채널증가 상태) : 다음 채널에 대한 폴링을 수행하기 위한 현재 설정된 폴링모드를 판단하여 non-priority 모드라면 K_LOOP를 빠져 나와 다음 채널을 선택하기 위한 인덱스를 증가시키게 되며, non-priority모드가 아니면 현재 채널에 대하여 QUeue에 데이터가 없어서 전송요구신호가 없을 때까지 연속적인 폴링이 되도록K_LOOP를 수행하게 된다.
- Algorithm_loop : 현재 설정된 폴링을 해야할 최대 채널 번호와 일치하면 처음 채널로 다시 폴링을 수행하도록 INIT_ST로 이동하게 되며 아직 최대 채널 번호보다 작은 상태라면 algorithm_loop를 수행하여 다음 채널에 대한 폴링이 수행된다.

3.2.2 MOVING

MOVING 블록은 그림 8의 유한 상태도와 같이 단지 POLLING 블록의 무빙 요구 신호인 /REQ에 의해서만 동작이 시작되며 동시에 현재 폴링주소를 전송을 요구한 원천주소(Source Address)로 저장한다. /REQ 신호가 유효하면 상태제어기(state machine)은 우선 프레임의 시작인지 여부를 Tag(7 : 0)신호를 해석하여 판단하게 되며 시작 태그(SOF)가 아니면 오류를 선언하고 나머지의 데이타에 대한 소거 작업에 들어간다. 일단 태그가 정확하면 브로드캐스트 또는 멀티캐스트인지를 헤더 정보에서 확인하게 되며 B또는 M비트가 설정되어 있으면 SID영역을 자신의 고유 시스템 ID로 새겨넣고, 이 변경된 헤더를 목적지로 쓰기동작을 수행하기 전에 헤더 전체를 변경하게 된다. 그러나 브로드캐스트나 멀티캐스트가 아닌 경우는 헤더 내의 길이 정보와 목적지 주소를 래치하도록 하고 헤더에 대한 목적지로의 쓰기 동작을 수행하여 전송을 완료한다. 이어서 프레임 데이타 영역을 원천주소지로 부터 읽기와 목적지로의 쓰기를 연속하여 수행하되 저장한 길이 정보를 하나씩 감소시켜가면서 0이 되었는지를 확인하며 0인 경우 다시 마지막을 표시하는 태그정보가 EOF(End of Frame) 확인하며 맞는경우 정상적인 한 프레임의 데이타를 전송 완료하게 되며 다음의 프레임 전송요구를 기다리게 된다. 한편, 마지막에 해당하는 태그와 일치하지 않는 경우도 오류로 선언되어 오류 처리 루틴으로 들어가며 마지막 태그가 확인될 때 까지 소거 작업을 수행한다. 길이가 0이 되지 않을 때는 연속적인 읽기와 쓰기 동작을 수행하는데 이때는 현재 원천채널의 버퍼의 상태가 고갈(empty)이 아닌지 여부를 확인하며 고갈인 경우도 오류로 처리된다.

이상과 같은 무빙 과정을 상태도 입력 방법을 사용하여 설계하였으며 각 상태의 설명은 다음과 같다.

- Notation
  - 원 : 하나의 상태를 정의한 것으로 굵은 선은 시작 상태를 의미하며 이중원은 내부에 또다른 상태가 존재함을 의미
  - 화살표 : 천이의 방향의미
  - 조건상자 : 천이화살표 상의 상자가 두개 혹은 하나가 있으며 보통 위에 표현한 것이 조건에 해당하는 것이며 하부에 표현한 것은 조건에 따른 신호의 변화를 정의한 것으로 VHDL에서 사용하는 할당문(assign)과 동일하게 표현되어 있다.
- RESET신호 : 모든 출력신호에 대한 초기값을 설정하는 상태.
- S(시작상태) : POLLING 블록으로부터 교환요구신호인 REQ신호가 유효한지를 확인하는 상태로 REQ신호가 유효하면 source주소유효신호 및 스트로브와 프레임의 헤더를 읽기 위한 SURD신호를 출력하면서 FLAG_CHK상태로 천이한다.
- FLAG_CHK(헤더검사 상태) : 내부에 단지 지연목적의 상태가 있으며 헤더의 각 영역의 상태와 태그를 해석하여 다음상태로 천이하는 조건들을 확인하는 상태이다
  - 태그가 시작이면서 브로드캐스트 혹은 멀티캐스트이면서 SID가 0이 아니면 자신의 ID를 변경하도록 하는 CHANGE_SID상태로 천이하며 SID가 1이면 CHANGE_HDR상태로 천이하면서SURD신호는 원상태로 돌리고 DAEN 및 DAAS를 유효하게 한다.
  - 브로드캐스트나 멀티캐스트가 아닌 일대일의 전송이라고 판단되면 역시 DAEN 및 DAAS를 유효하게 하면서 STRT_PTP상태로 천이시킨다.
  - 그러나 동시에 브로드캐스트나 멀티캐스트가 세트

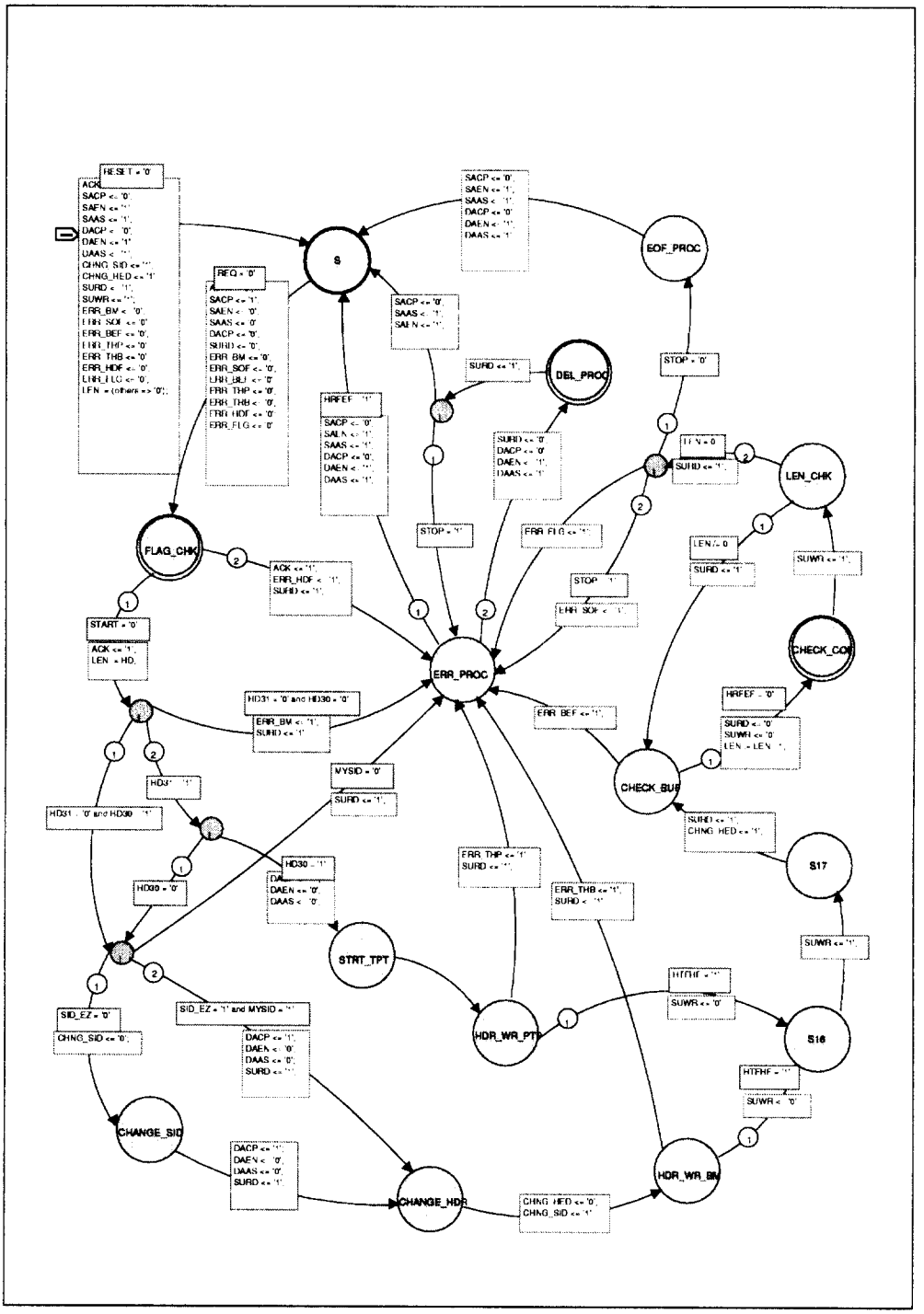


그림 8. MOVING 블록의 상태천이도

Fig. 8 State transition diagram of MOVING block

된 헤더이거나 태그가 시작(SOF)이 아인 프레임에 대해서는 모든 내용을 소거하는 오류 ERR_PROC 상태로 천이시킨다.

- CHANGE_SID(SID변경상태) : SID영역에 자신의 ID를 설정하도록하는 상태
- CHANGE_HDR(헤더 변경상태) : SID변경제어신호의 철회 및 변경된 헤더의 내용이 목적지주소(DA)로 쓰여지도록 제어하는 상태
- HDR_WR_BM(브로드캐스트 및 멀티캐스트에 대한 헤더 쓰기 상태) : 실제적으로 변경된 헤더가 목적지에 쓰여지도록 SUWR신호를 유효하게 만드는 상태로 목적지의 FIFO가 half full이 아닐때 S16으로 천이하여 헤더가 쓰이도록 한다.
- STRT_PTP(일대일교환 시작 상태) : 브로드캐스트나 멀티캐스트가 아닌 일대일로 교환해야하는 처리의 시작상태이다.
- HDR_WR_PTP(일대일교환 헤더쓰기 상태) : 일대일 교환에 대해 SUWR신호를 유효하게 만들어 목적지의 FIFO의 상태가 half full이 아니면 S16으로 상태를 천이하여 헤더가 쓰여지도록 한다.
- S16(헤더쓰기 구간 상태) : 실제적인 쓰기 행위가 이루어지도록 SUWR신호를 비활성화(deassert)시키는 상태
- S17(임의 지연 상태) : 헤더의 쓰기에 연이어 데이터를 쓰기 위한 단순 지연 상태이며 이전에 assert시킨 헤더변경제어신호와 SURD신호를 비활성화 시키게 된다.
- CHECK_BUF(버퍼 확인상태) : 헤더를 목적지주소에 씀으로써 교환이 시작되고 난 후 실제 프레임의 유효 데이타를 source로부터 읽어서 목적지로 쓰기를 할 때 우선 source측의 FIFO상태를 확인하여 데이터가 존재한다면 source측으로는 SURD및 목적지에는 SUWR 신호를 활성화시키고 헤더내의 길이정보를 하나씩 감소시키는 단계의 상태이다.
- CHECK_COF(연속태그 확인 상태) : 데이터의 교환에 앞서 우선 태그를 해석하여 데이터의 연속임을 나타내는 COF임을 확인하여 SUWR를 반전시킴으로써 목적지에 쓰기를 완료하게 된다.
- LEN_CHK(데이터길이 확인 상태) : 64비트 단위의 데이터를 하나씩 전송(교환)하고 난 후 바로 길이를 확인하는 상태로서 길이 영역이 아직 0이 되지 않았다면 상태는 CHECK_BUF로 되돌아가서 다음 데이터를 전송하게 되며 길이가 0이라면 source측으로 인가한 SURD를 반전시키고 태그정보도 같이 확인하여 EOF(End of Frame)이라면 EOF_PROC로 천이하지만 만약 아니라면 오류처리를 하는 ERR_PROC상태로 천이하게 된다.
- EOF_PROC(EOF처리 상태) : 정상적인 한 프레임을 전송(교환) 완료한 상태로서 모든 제어 신호를 일단 초기로 돌려놓는 역할을 한다.
- ERR_PROC(오류 처리 상태) : 각 상태마다 오류의 조건들이 발생시 이를 각 오류 종류에 대해 발생회수를 누적하고 시스템 제어부로 인터럽트의 요청 등의 후속 조치들이 따르게 된다. 아울러 이때 Source측의 FIFO상태를 확인하여 비어있음을 확인하면 바로 처음 상태인 S로 돌아가게 되지만 FIFO에 데이터가 잔류한다면 오류가 있는 데이터로 보고 읽어서 버리는 작업을 하도록 DEL_PROC로 천이하게 된다.
- DEL_PROC(데이터 소거 상태) : source측의 FIFO상태를 확인하면서 빌 때까지 계속 읽어내도록 SURD를 제어하여 버리게 되고 모두 빈 것을 확인하면 모든 신호들을 초기상태로 돌리면서 역시 S상태로 천이한다.

3.2.3 DECODER

MOVING 블록의 동작시 필요한 정보 중 해석이 필요한 Tag(7 : 0)를 해석하여 SOF(Start of Frame), COF (Continuous of Frame) 및 EOF를 구분하는 기능과 SSID와 헤더 영역의 SID를 비교 해석하기 위한 디코더이다.

3.2.4 CPU_INTF & SA_LATCH

범용 마이크로프로세서와 8비트의 데이타로 접속할 수 있도록 하였으며 본 제어기의 기능을 다할 수 없는 오류나 외부적인 상황에 의해 소프트웨어적인 초기화를 위한 제어레지스터, 각종 오류의 발생에 대한 인터럽트 통보를 제어하는 레지스터, 인터럽트 발생에 대한 상태플래그를 나타내는 레지스터로 구성되어 있어 읽기와 쓰기가 가능하도록 하였다. 오류 발생에 대한 인터럽트 처리 기능이 있어 호스트로 인터럽트 요구 신호를 자동으로 생성하며 호스트가 인터럽트의 상태를 나타내는 플래그 레지스터를 읽기 동작이 이루어지면 곧바로 인터럽트 요구신호와 상태 플래그는 자동으로 클리어(clear) 된다. SA_LTCH(Source Address Latch) 블록은 오류가 발생된 주소를 일시 저장하여 호스트로 하여금 오류의 발생 포트를 확인할 수 있도록 하였다.

3.2.5 Register 기능

H-Bus Controller의 장애 관리와 H-Bus상의 프레임 데이타의 헤더나 프레임 순서 등의 오류에 대한 신속한 시스템 관리가 이루어질 수 있도록 폴링과 무빙 과

ADDR	Register Name	Descriptions
00	ARB_CNTR_REG	Internal function reset by software, (bit0 = '0')
01	SRC_ADDR_REG	Source address when an error occured
02	INT_CNTR_REG	Interrupt Enable/disable control
03	INT_FLG_REG	Interrupt status register
04	TBL_LEN_REG	Polling Entry Table length register
05	Reserved	
06	Reserved	
07	Reserved	
08	POLL_SEQ_REG(0)	poll sequence table(1st entry)
09	POLL_SEQ_REG(1)	poll sequence table(2nd entry)
0A	POLL_SEQ_REG(2)	poll sequence table(3rd entry)
~	~	~
~	POLL_SEQ_REG(i)	poll sequence table(i-th entry)
~	~	~
87	POLL_SEQ_REG(127)	poll sequence table(128th entry)

그림 11. 레지스터 맵
Fig. 11 Register Map

정의 오류 발생시 인터럽트(interrupt)를 이용한 상태 보고와 해당 오류 채널 주소의 보고, 복구 불가능한 치명적인 오류시 소프트웨어적인 초기화, 인터럽트 발생의 제어 등과 관련된 레지스터를 갖는다. 호스트 인터페이스의 데이타 버스는 바이트 크기이며 제어용 레지스터 5개와 폴링 테이블용 레지스터로 128개가 있다.

IV. 설계 검증

각 블록별로 서로 다른 설계 입력 방법을 사용하였으나, 각 블록별로 전체 기능 검증을 위해 시뮬레이션을 수행하지는 않았다. 단 중요 기능은 간단하게 VHDL simulator(VSS/Synopsys사)[8]를 이용하여 확인하였다. 전체 블록(HBUS_CNTR)을 통합하여 VHDL 테스트벤치에 의한 시뮬레이션을 수행하였다.

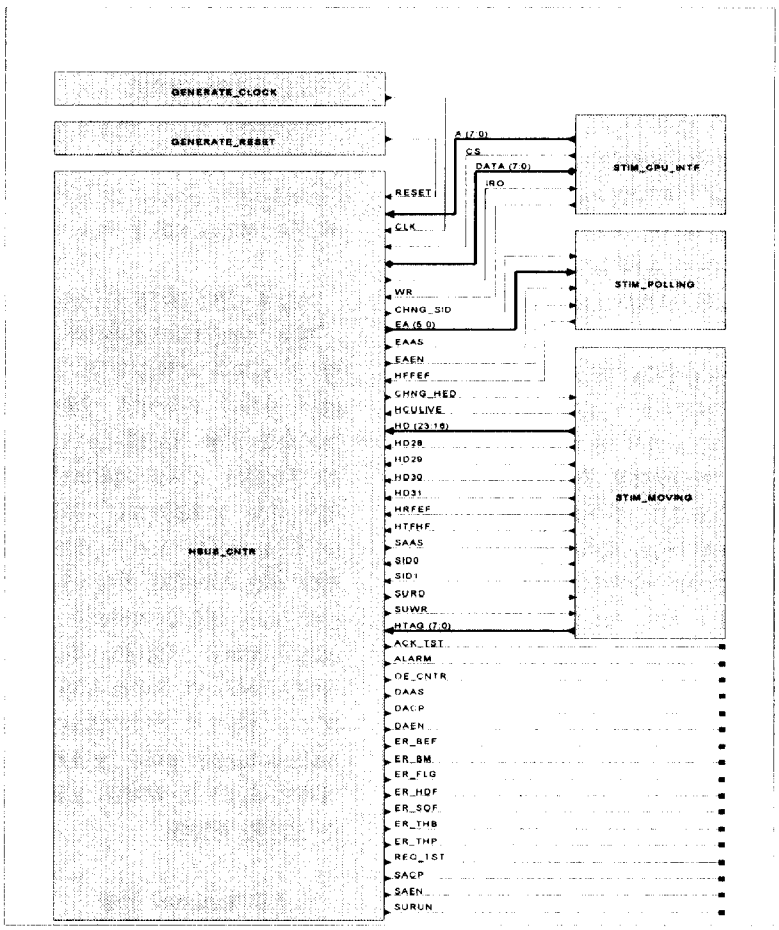


그림 9. 테스트벤치 블록도
Fig. 9 Test bench block diagram

4.1 테스트벤치(testbench)

전체 블록의 통합 시뮬레이션을 위한 테스트벤치의 블록도를 그림 9에 나타내었다. 여기서 HBUS_CNTR 블록이 기능 검증할 UUT(Unit Under Test)가 되며, 연속적인 클럭을 생성하는 Generate_clock부, Reset 신호를 생성하는 Generate_reset부, Host 에서 레지스터의 읽기/쓰기 및 인터럽트 서비스 등의 사건발생(stimulus) 기능을 하는 STIM_CPU_INTF부, 폴링시 필요한 사건발생 기능을 갖는 STIM_POLLING부 및 데이터 무빙시 필요한 사건발생 기능을 제공하는 STIM_MOVING부 등으로 이루어진다.

4.2. 시뮬레이션

본 중재교환 제어기의 기능을 검증하기 위한 시뮬레이션은 VHDL Simulator인 VSS(Synopsys사)에서 수행하였다. 기능의 정상 여부를 시뮬레이션한 결과를 보면 개념수준, RTL 수준 및 합성결과인 게이트 수준들이 모두 동일함을 검증하였으며 그림 10에 개념 수준의 시뮬레이션 결과 파형을 나타내었다.

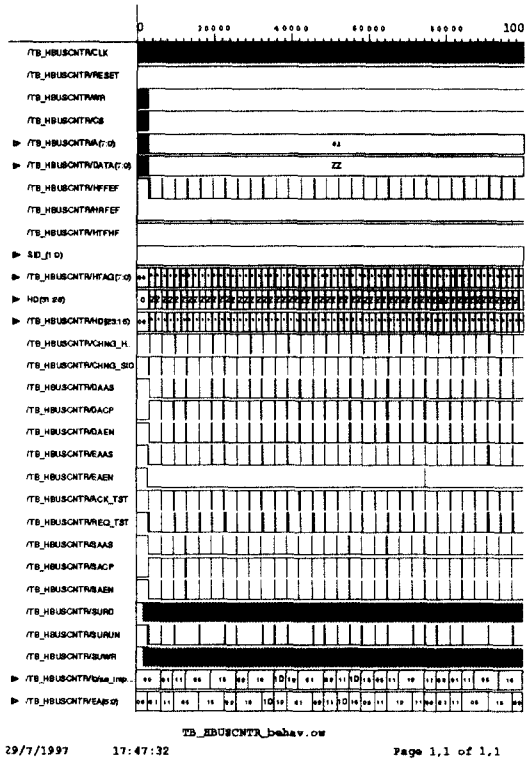


그림 10. 시뮬레이션 결과 예
Fig. 10 Simulation example

V. 결 론

본 논문에서는 기존의 HSSF가 모든 노드들이 동등한 트래픽이 부가 될 때 각 노드별로 약 20Mbps 정도의 대역폭을 할당할 수 있는 순환 폴링 중재교환 방식을 사용하던 것을 실제적으로 트래픽 부하가 큰 노드에게는 보다 더 많은 대역폭을 할당하고 상대적으로 트래픽 부하가 적은 노드들에게는 대역폭이 덜 할당되도록 하는 우선순위 폴링 중재교환방식으로 개선을 하였다. 또한 단일 FPGA소자로 구현하여 실장밀도를 높일 수 있었다.

설계 방법적인 측면에서는 현재 가용 가능한 개념수준의 설계 입력 수단부터 직접 VHDL로 코딩 입력까지 다양한 방법의 복합적 수단(mixed methodology)을 사용하여 설계 기간을 단축할 수 있었다. 특히, 현재 주요 VHDL 설계는 RTL(Register Transfer Level) 수준으로 이루어 지고 있으나 좀더 개념(behavioral) 수준에서 코딩한 것을 툴을 이용하여 자동적으로 RTL 코드를 생성하여 합성하는 방법으로 일부 블록에 도입하였다. 즉, 구조(architecture)를 자동적으로 만들어 주는 툴의 도움을 받아 설계 기간을 단축하였다. 최근, 많이 소개된 그래픽 입력 툴들을 이용하여 블록마다 상태도, 흐름도, 진리표 들을 이용하여 입력한 후 합성 가능한 VHDL 코드를 생성하였고 일부 블록은 VHDL로 매뉴얼 코딩한 것을 사용하였다.

본 제어기의 설계 과정에서 POLLING 블록의 설계에 흐름도 입력 후 개념수준 코드의 자동 생성 과정과 매뉴얼로 개념수준 코딩하는 방법을 병행하여 비교하여 보았다. 흐름도의 입력후 자동 생성 코드는 전적으로 Behavioral compiler에서 합성 가능하지 않을 수 있어 합성 가능한 형태를 경험적으로 인지하는데 어려움과 일부 생성 코드가 합성 가능하지 않아 매뉴얼로 수정할 수 밖에 없었다. 역시 RTL과 마찬가지로 합성 가능한 코딩형태(style)을 숙지하는 것이 숙제라고 생각된다.

설계 입력 후 검증은 VHDL 시뮬레이터를 이용하여 검증하였고 구현 결과는 타이밍까지 반영된 VHDL 출력을 back annotation하여 설계 당시 VHDL 시뮬레이터에서 검증 하였다. 설계 검증을 위하여 각 블록별로 서로 다른 설계 입력 방법을 사용하였으나 전체 블록을 통합하여 VHDL 테스트벤치에 의한 시뮬레이션을 수행하였으며 개념수준, RTL 수준 및 합성결과인 게이트 수준들이 모두 동일함을 검증하였다. 구현은 62,000 사용가능 게이트급의 FPGA인 FLEX10K100을 사용하였고 약 66%의 자원을 사용하였다. 향후

ASIC으로의 용이한 재제작을 위해 본 디바이스의 내장된 RAM은 사용하지 않았다.

본 논문에서 설계 구현한 우선 순위 폴링방식의 중재교환 제어기는 PC통신 및 인터넷 서비스를 제공하는 한국통신의 게이트웨이 장치인 대용량 통신처리 시스템에 활용되고 있으며 현재 상용서비스 시험 중에 있다.

참 고 문 헌

1. 김동원, 전경표, 류근택, 배현덕, "Development of Infoshop Service System," Proceeding of ICCE, pp. 388~389, Jun. 1996.
2. 김동원, 신현식, 류원, 이현우, 전경표, 배현덕, "개방형 정보검색시스템의 설계 및 성능분석," 정보처리학회 논문지, 제 3권 7호, 1996. 12.
3. 김동원, 신현식, 류원, 이현우, 전경표, 배현덕, "이종망간의 상호 연동 게이트웨이 시스템을 위한 내부 고속연동망," 정보처리학회 논문지, 제 4권 2호, pp. 499~514, 1997. 2.
4. 류원, 신현식, 김동원, "대용량통신처리시스템에서의 고속스위치부 소프트웨어 설계 및 구현," 통신학회 하계학술, pp. 938~941, 1996.9.
5. D. R. Manfield, "Analysis of a Priority Polling System for Two-way Traffic", IEEE Transaction on Communications, Vol. COM-33, No. 9, pp. 1001-1006, 1985.
6. Joseph E. Baker, Izhak Rubin, "Polling with a General-Service Order Table", IEEE Transactions on Communications, Vol. COM-35, No. 3, pp.283-288, 1987.
7. Summit Design Inc., Visual HDL for VHDL, 1994.
8. Steve Carson, "Introduction to HDL-Based Design using VHDL," Synopsys Inc., 1991.
9. ALTERA Corp., MAX+PLUS II Getting Strated, 1995.
10. ALTERA Corp., Data Book-F10K, 1996.
11. S. Mazor, P. Langstraat, A Guide to VHDL, KAP, 1993.

김 동 원(Dong Won Kim) 정회원
1983년 : 경북대학교 전자공학과 공학사
1990년 : 경북대학교 대학원 전자공학과 공학석사
1998년 : 충북대학교 대학원 전자공학과 공학박사
1983년 3월~1998년 2월 : 한국전자통신연구원 선임연구원
1998년 3월~현재 : 충북도립 옥천전문대학 정보통신과 전임강사
1998년 8월~현재 : 한국전자통신연구원 초빙연구원

김 도 영(Do Young Kim) 정회원
1985년 : 성균관대학교 전자공학과 공학사
1987년 : 성균관대학교 대학원 전자공학과 공학석사
1997년~현재 : 충남대학교 대학원 박사과정
1987년 2월~현재 : 한국전자통신연구원 교환전송기술연구소 지능망연구부 선임연구원

신 현 식(Hyeon Sik Shin) 정회원
1991년 : 금오공과대학교 전자공학과 공학사
1991년~1998년 6월 : 한국전자통신연구원 선임연구원
1998년 7월~현재 : 임프레스 정보통신