Distributed Network Protocol Version 3.0 을 이용한 필드버스 시스템 구현

Implementation of a Fieldbus System Based On Distributed Network Protocol Version 3.0

문 전 일*, 김 정 섭, 김 종 배, 최 병 욱, 임 계 영
(Jeon-Il Moon, Jung-Sub Kim, Jong-Bae Kim, Byoung-Wook Choi, and Kye-Young Lim)

Abstract : Distributed Network Protocol Version 3.0 (DNP3.0) is the communication protocol developed for the interoperability between a RTU and a central control station of SCADA in the power utility industry. In this paper DNP3.0 is implemented by using HDL with FPGA and C program on Hitachi H8/532 processor. DNP3.0 is implemented from physical layer to network layer in hardware level to reduce the computing load on a CPU. Finally, the ASIC for DNP3.0 has been manufactured from Hynix Semiconductor. The commercial feasibility of the hardware through the communication test with ASE2000 and DNP Master Simulator is performed. The developed protocol becomes one of IP, and can be used to implement SoC for the terminal device in SCADA systems. Also, the result can be applicable to various industrial controllers because it is implemented in HDL.

Keywords : bistributed network protocol (DNP), verilog HDL, FPGA, ASIC

I. 서론

필드버스는 분산 제어 및 자동화 시스템에서 필드에 설치된 장비들 간에 실시간으로 데이터를 교환하도록 하는 디지털 직렬 통신망이다. 필드버스에는 센서, 루프제어기, PLC, 모터, 밸브, 로봇, NC 머신, operator station 등의 각종 산업용 제어, 자동화 장비들이 접속된다. 공정 제어를 위한 신호 전송 체계로는 1950년대까지는 3-50 psi의 공압 계측 신호가 표준으로 사용되었고, 1960년대부터는 4-20 mA의 전류 또는 전압의 아날로그 신호가 표준으로 채택되었다. 1980년대 중반부터 디지털 기술을 이용하는 통신망 신호 전송 체계의 필요성이 대두되기 시작하면서 필드버스 기술이 개발되기 시작하였다. 공정 제어 및 각종 자동화 시스템에서 필드버스를 도입함으로써 얻을 수 있는 장점으로는 다음과 같은 사항들이 있다[1,2].

- 필드버스는 단일 배선을 이용한 일-대-다 통신 기술을 사용함으로써 자동화 시스템의 초기 설치비용을 크게 줄일 수 있다.
- 필드버스에서는 디지털 신호를 사용함으로써 기존의 아날로그 신호에 비하여 외부의 잡음에 의한 영향이 감소되고, 여러 개의 중복 신호를 동시에 전송할 수 있다.
- 필드버스는 양방향 통신을 가능하게 하여 자동화 시스템의 모니터링 및 유지 보수에 소요되는 비용을 크게 절감시킬 수 있다.
- 필드버스는 시스템의 변형을 용이하게 하여 자동화 시스템의 유연성과 확장성을 증대시키며, 따라서 자동화 관련 신기술의 도입이 용이해진다.
- 필드버스는 스마트센서의 도입을 가능하게 하여 센서 신호의 전처리 과정이 스마트 센서 내에서 모두 이루어질 수 있도록 한다. 또한, 기존의 시스템에서는 제어 센터의 컴퓨터에서 처리되던 제어기능이 필드 장비에서 바로 구현될 수 있어 자동화 시스템의 분산화가 가능해진다.

이런 필드버스 프로토콜 중의 하나로써 DNP (Distributed Network Protocol)이 있다. DNP는 1990년에 Westronic, Inc.(현재 GE Harris)에서 최초로 개발되었으며, 1993년에 DNP 3.0 으로 일반에 공개되었다. DNP는 전력회사, 오일, 가스, 수자원 분야에서 사용되는 중앙집중형 원격감시장치인 SCADA시스템에서 RTU-중앙제어장치간 호완성보장을 위하여 개발된 표준화된 통신프로토콜이다. 특히, 이 프로토콜은 특성상 안정성이 뛰어나고, 저급의 통신환경에서도 우수한 성능을 나타내고 있어서 기존의 방식에 비해 SCADA환경에 잘 적용될 수 있으며, Open Standard를 지향하기 때문에 특정 회사에 소속된 고유의 프로토콜이 아닌 공개된 프로토콜로서 모든 권한은 사용자 그룹이 소유하고 있다. IEEE에서는 RTU-IED간 메시지교환의 표준으로 추천하고 있으며, 국내에서는 한전, 가스공사등 관련기업에서 도입하였거나 준비중이며 전세계적으로 각국에서 많은 Vendor들의 산업표준으로 자리잡아가고 있다[3].

DNP는 OSI-7(Open System Interconnection) 계층 모델을 선택하는 것 대신에 좀더 간단한 구성을 위해서 IEC(International Electrotechnical Commission)에 의해서 제안된 단순한 3 계층을 채택하고 있다. 그러나, 실제적으로 DNP 3.0은 메시지 분할을 제공하기 위해서 가상 트랜스포트 계층이라는 4번째 계층을 추가해 사용하고 있다.

DNP 프로토콜의 특징은 개방형이고, 16바이트 프레임을 전송할 때마다 16 비트의 CRC를 추가함으로써 안정성을 높였으며, 잡음이 심한 환경에서 큰 장점을 나타낸다. 또한 Poll-Response, Polled report-by-exception, unsolicited response, 그리고

* 책임저자(Corresponding Author)
논문접수 : 2003. 7. 9., 채택확정 : 2004. 1. 8.
문전일, 김종배 : LG 산전(주) 중앙연구소
(jimoon@lgis.com/jongbaek@lgis.com)
김정섭 : Electrical Engineering, Penn State University
(subbykjs@hotmail.com)
최병욱 : 선문대학교 제어계측공학과(bwchoi@sunmoon.ac.kr)
임계영 : 한국산업기술대학교(kylim0403@hanmail.net)

peer-to-peer 등 다양한 통신 방식을 제공하고, 단순한 구조로 확정성이 용이하다. RS-232, RS-485, 모뎀, 이더넷 등 다양한 물리계층의 인터페이스를 제공하고, 단순 연결로 65000개 이상의 디바이스를 지원하고, 브로드캐스트 모드 또한 제공한다. 16 비트 Source/Destination 주소를 이용하여 다수개의 마스터를 지원하고, Unsolicited Mode라는 슬레이브가 독자적으로 마스터에 event data를 전송하는 모드를 지원한다.

본 논문에서는 DNP 3.0 프로토콜의 물리 계층과 데이터 링크 계층을 하드웨어로 구현하였으며, 구현 방법으로는 HDL을 사용하였고 FPGA를 이용하여 기능 검증을 수행하였고, 최종적으로 ASIC을 만들었다. 로직의 검증을 위하여 FPGA와 H8/532로 전체 시스템을 구성, 기존의 DNP 마스터 장비와의 상호 테스트를 수행하여 동작을 검증하였다. 본 논문에서 II장은 DNP 3.0 프로토콜에 대한 전반적인 개요, III장은 기존의 구현 방법을 사용할 때 발생할 수 있는 문제점, IV장은 개발한 시스템의 구성 요소에 대한 내용, V장은 실제로 테스트를 수행한 실험 결과, VI장은 결론으로 구성된다.

II. DNP 3.0 프로토콜의 개요

DNP 3.0 은 그림 1과 같은 계층 구조를 가진다.

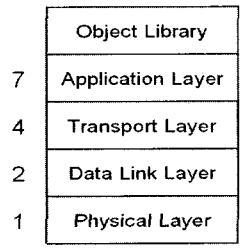


그림 1. DNP 3.0 프로토콜 계층 구조.
Fig. 1. DNP 3.0 Protocol Layering.

각 계층에서 제공되는 서비스는 다음과 같다. 물리 계층에서는 통신하는 매체와 직접 연결되어 매체의 상태, 동기화 등을 제어한다. 주로 DNP에서는 RS-232와 RS-485가 사용되면, 광통신, 무선, 인공위성, 이더넷 등을 이용한 통신 방법이 연구되고 있다.

데이터 링크 계층의 송신단과 수신단 사이에서 논리적 연결을 처리하고, 물리적인 채널 에러 특성을 향상시키기 위해 사용된다. 예를 들어, DNP는 data link header 끝과 송신 프레임의 매 16 바이트마다 16 비트의 CRC를 추가함으로써 에러에 대한 보정을 한다. 데이터 링크 프레임의 최대 크기는 256 바이트이며, 각 프레임은 16 비트의 source 어드레스, 16 비트의 destination 어드레스로 구성되며, 또한 broadcast 어드레스(0xffff)도 포함된다. 10 바이트의 data link header에는 어드레스 정보, 16비트의 시작 코드, 프레임 길이, data link control byte 등이 포함된다.

가상 네트워크 계층은 응용 계층의 메시지를 다수의 데이터 링크 프레임으로 분할한다. 각 프레임에 대해 메시지의 첫 번째 프레임인지, 마지막 프레임인지, 단일 프레임인지를 나타내는 1 바이트의 기능 코드가 삽입된다. 또한 기능 코드는 각 프레임별로 일련 번호를 가지게 되고, 이것을 이용하여 잃어버린 프레임을 검출해 낼 수 있다.

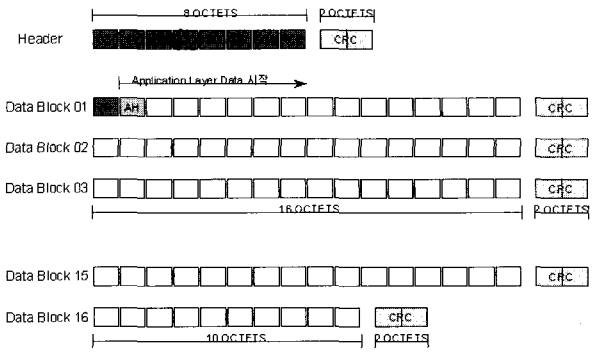


그림 2. DNP 3.0의 프레임 형식.
Fig. 2. DNP 3.0 Frame Format.

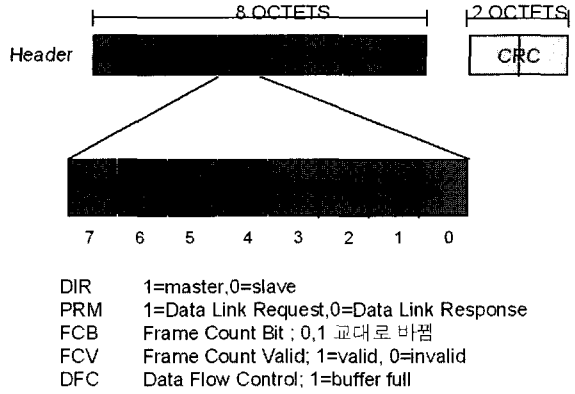


그림 3. DNP 3.0의 데이터 링크 콘트롤 필드 형식.
Fig. 3. DNP 3.0 Data Link Control Field Format.

응용 계층에서는 필요로 하는 메시지를 만들고, 받은 메시지에 대해서 응답을 수행한다. 일반적으로 DNP는 마스터와 슬레이브로 구성되는데, 마스터는 데이터 오브젝트에 대한 동작을 요구하고, 슬레이브는 이런 요구에 대해서 응답을 수행한다. 또한 Unsolicited response라는 모드가 있어서 슬레이브가 마스터의 요구 없이도 필요시 메시지를 보낼 수도 있다.

III. 기존의 구현 방법을 이용한 개발시 문제점

지금까지의 DNP의 구성은 물리 계층에 주로 RS-232, RS-485, 광모듈 등을 이용하고, 그 위의 계층은 소프트웨어로 구성하였다. 어드레스 판별이나, CRC의 비교, 헤더의 구성 등과 같은 모든 프레임에 대한 반복 작업을 소프트웨어로 처리하였다. 따라서, 자기 노드의 프레임이 아님에도 불구하고, 일단 수신 동작을 모두 수행한 후에 일일이 소프트웨어로 destination 어드레스와 자신의 어드레스를 비교함으로써 자기 노드의 프레임을 판별할 수 있었다. 또한, CRC의 정상 여부도 모든 프레임을 수신한 후에야 비교가 가능했다. 이는 CPU의 과부하를 발생시켜, 훨씬 낮은 성능의 CPU를 이용할 수 있음에도 불구하고, 불필요한 모든 프레임을 처리하기 위하여 과다한 메모리와 고성능의 CPU를 사용하게 되었다. 그러나, 이는 현재 DNP를 전용으로 처리해 주는 하드웨어 칩이 존재하지

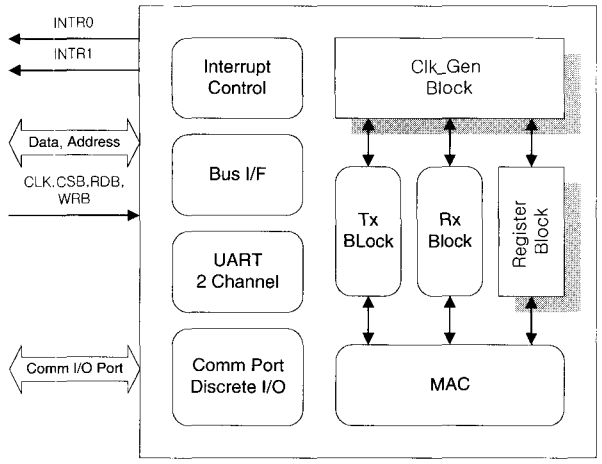


그림 4. DNP 3.0의 블록 다이어그램.
Fig. 4. DNP 3.0 Block Diagram.

않기 때문에 피할 수가 없는 일이어서, ASIC을 이용해서 DNP용 데이터 링크 전담 하드웨어를 구성함으로써 이를 해결할 수 있게 되었다.

또 다른 문제로는 Unsolicited response라는 모드에 관한 것이다. 이 모드는 슬레이브가 유사시 갑작스러운 변화를 마스터에게 직접 알릴 수 있는 모드임에도 불구하고, 기존의 물리계층을 이용해서는 구현할 방법이 없어서 사용하지 못하였다. 마스터, 슬레이브 방식에서 갑작스러운 슬레이브에서의 송출은 패킷간의 충돌을 발생시킬 수가 있기 때문에 데이터의 안정성을 심각하게 위협하게 된다. 충돌 회피 방식이 적용되어 있는 이더넷 모듈 같은 것을 물리 계층으로 사용한다면 가능할 수도 있지만, 이럴 경우는 이더넷 모듈을 제어하기 위한 불필요한 작업이 포함되게 된다. 따라서, 충돌회피방식이 적용된 DNP 전용 ASIC을 구현함으로써 unsolicited response 모드를 불필요한 작업 없이 구현할 수 있게 되었다.

IV. 개발한 시스템의 구성

본 시스템은 DNP 3.0 프로토콜에서 물리 계층과 데이터 링크 계층을ASIC으로 구현하였으며, HDL을 이용하였다. 기능 테스트를 위해 FPGA를 사용하였고, 펌웨어와 테스트 보드를 설계하였다.

1. ASIC의 내부 구성

본 ASIC는 DNP 3.0 Core와 UART, Discrete IO 블록의 크게 3부분으로 나누어진다. DNP 3.0 프로토콜의 처리를 위한 DNP 처리 블록 외에, 기존처럼 소프트웨어로 처리를 가능하게 해주기 위한 2 채널 UART, 범용 IO 제어를 위해 16 비트의 Discrete IO 블록이 설계되어 있다. CPU 관련 인터페이스 등의 공통 부분은 DNP 3.0 Core부분에서 일단 다루어지고, 다른 블록은 이 블록의 신호를 받아서 사용한다. UART와 Discrete IO는 범용적으로 사용되는 IP(Intellectual Property)를 사용하였으므로, 본 논문에서는 다루지 아니한다. 전체 Block Diagram은 그림 4과 같다.

DNP 3.0 Core는 MAC_BLK, TX_BLK, RX_BLK, INT_BLK, REG_BLK등 크게 6개 블록으로 나누어 진다. 각각의 블록에 대한 기능 및 동작은 이후 자세히 기술하도록 하겠다.

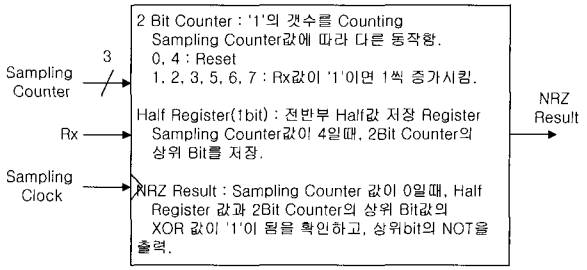


그림 5. 맨체스터 엔코더/디코더 알고리즘.
Fig. 5. Manchester Encoder/Decoder Algorithm.

2. MAC(Media Access Control) 블록의 구현

DNP의 엔코딩과 디코딩 방법에는 2가지 모드가 존재한다. 첫째UART 모드일때는 일반적인 UART와 똑같은 NRZ 형태로 데이터를 주고 받는다. 1비트의 start bit와 data bit, 선택가능한 parity bit, 1 stop bit로 구성되어 통신에 사용된다. 둘째로 데이터 스트림 모드일때는 Manchester Coding 방식을 이용해서 통신을 수행한다. Manchester Coding이란 1 Bit Time의 중간에서 꼭 한번의 Transition을 발생하고, 0인 경우는 high에서 시작하여 low로 끝나고, 1인 경우는 low에서 시작하여 high로 Transition이 발생하는 방식이다. 이렇게 두 가지 모두를 사용하는 이유는 UART 모드일때는 일반적인 RS-232 방식을 택하므로써, 범용 소프트웨어 DNP와의 호환성을 주고, 데이터 스트림 모드는 Unsolicited Response 모드를 지원하기 위해 항상 한 비트당 라인 상태를 체크함으로써 데이터의 충돌을 방지하기 위해서이다. 맨체스터 엔코더와 디코더의 설계 알고리즘는 그림 5 과 같다

충돌회피를 위해서 구성한CSMA/CD(Carrier Sense Multiple Access/Collision Detect)는 UART 모드에서는 적용하지 않고, 데이터 스트림 모드일때만 적용한다. 일단 Idle 상태는 Manchester Coding 방식을 사용하기 때문에, 한 비트 타임동안 반전이 발생하지 않으면 Idle 상태로 본다. CSMA 알고리즘으로는 Non-persistent CSMA 방식을 사용한다. Non-persistent CSMA 방식은 Idle 상태일 때는 즉시 전송하고, idle상태에 있지 않으면 Random Back Off 시간동안 지연 후 idle 상태를 다시 체크한다. Random Count는 네트워크의 특성을 보아, 최대값 31보다 작은 수 중 랜덤하게 선택하도록 하고, time tick(=1 bit pulse * 8 bit * 평균 Data Byte 수)값도 차후 네트워크의 특성을 보아 정하도록 하였다. 충돌 감지 핀이 존재할 경우, 충돌 신호가 들어오면 즉시 전송을 정지하고, CPU에 충돌 감지를 알리게 하였고, 충돌 감지 핀이 없을 경우에도, 충돌이 발생할 경우는 Poll에 대한 Response와 Unsolicited Response 뿐이므로, Poll에 대한 Response는 Master에서 제어를 할 것이고, Unsolicited Response에 대한 것은 꼭 Ack 프레임을 이용해서 Ack를 받도록 구성함으로써 전송 완료를 판단하여 충돌 여부를 감지할 수 있도록 하였다.

CRC 검출을 수신 프레임에 대한 유효 여부를 체크하기 위해 사용하며, 수식은 다음에서 보는 바와 같다.

$G(x) = X^{16}+X^{13}+X^{12}+X^{11}+X^{10}+X^{8}+X^{6}+X^{5}+X^{2}+1$

CRC 로직은 XOR와 Shift Register를 이용해서 구현될 수 있다.

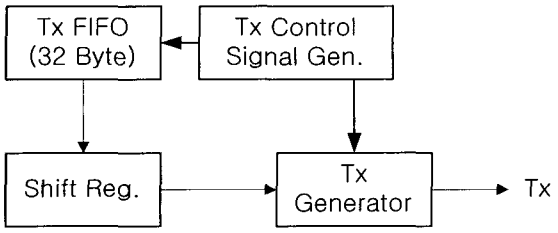


그림 6. 송신부 블록 다이어그램.
Fig. 6. Transmit Block Diagram.

이때 레지스터의 초기값은 1이고, 계산 후 송신단으로 출력되는 값은 반전되어 나가게 된다. 수신한 CRC 데이타와 수신시 생성한 CRC값을 비교하여 에러 발생시 상태 레지스터의 해당 비트에 세팅하고, 인터럽트를 발생시킨다. FIFO 및 CPU에 전달하는 데이터는 CRC 부분을 제거한 순수 데이터 프레임만을 상위로 전달하게 된다.

3. 송신부(Transmit) 블록의 구현

송신 블록은 CPU에서 보내야 할 데이터를 FIFO에 저장한 후, Tx_Start 신호에 따라 차례대로 송신핀에 보내게 된다. 보낼 때는 위에서 언급한 바와 같이 Uart 모드와 데이터 스트림 모드에 따라서 다른 Encoding 방식으로 보내진다. 송신 블록에서는 CRC가 이미 모두 생성되어 FIFO에 저장되기 때문에 자동으로 생성하지 않는다. 송신 블록의 블록 다이어그램은 그림 6과 같다.

전송의 시작은 반드시 0x0564로 시작하고, 전송이 완료되면, 즉 주어진 송신 데이터 길이만큼 전송을 수행한 후, Data Length Counter가 '0'이고 FIFO가 비어 있으면 정상적으로 전송이 완료된 것으로 본다. 그러나, FIFO가 비어있지 않으면 에러를 발생시키고 난 후 리셋한다. TX Length Counter의 값은 0인데 FIFO가 비어있지 않은 경우와 TX Length Counter가 0이 아닌데 FIFO가 비어있는 경우 모두 송신을 중지하고, 이를 상태 레지스터에 저장한다.

TX FIFO Overrun은 CPU가 Threshold와 기타 Status를 제대로 보았으면 문제 없으므로 두지 않고 Underrun은 TX_Start에 의해 송신을 시작하므로 CPU가 오동작하지 않으면 발생할 수가 없다. 그러나, TX_FIFO_FULL은 라인 상태 레지스터에 셋팅하여 향후 에러 체크를 위해 사용한다.

4. 수신부(Receive) 블록의 구현

수신부 블록은 수신부 조정부, 수신 FIFO(32 byte), 쉬프트 레지스터, 어드레스 검출 블록과 CRC 비교기로 구성되고, 블록다이어그램은 그림 7과 같다.

디코더 블록에서 새로운 패킷를 인식하면 쉬프트 레지스터로 serial data를 넘겨준다. 쉬프트 레지스터에 16 비트의 데이터가 저장되면 상위 8 비트를 송신 FIFO에 저장한다. 동시에 쉬프트 레지스터에 있는 어드레스 영역를 인식하여 수신해야 할 데이터인지 판단한다. 어드레스가 일치하지 않는 경우에는 수신을 중지하고, FIFO도 데이터를 받기 전의 값으로 돌린다. 그리고, 새로운 패킷이 시작되기를 기다린다. FIFO에 저장되는 내용은 CRC를 제거한 순수한 데이터 프레임만을 저장한다. 브로드 캐스트 모드를 지원하므로, 수신된 어드레스가 0xffff일 경우는 모든 노드에서 수신하도록 설계되었다.

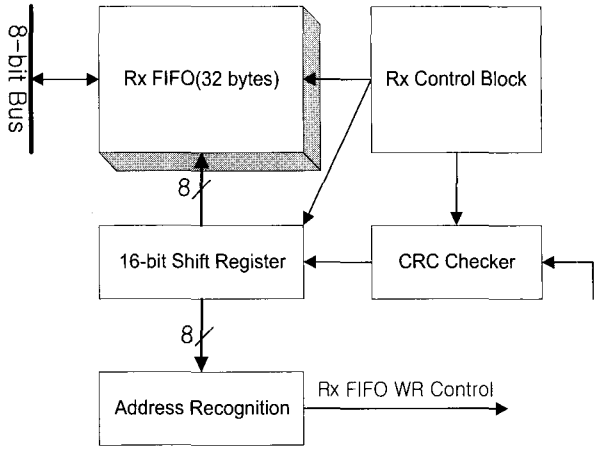


그림 7. 수신부 블록 다이어그램.
Fig. 7. Receive Block Diagram.

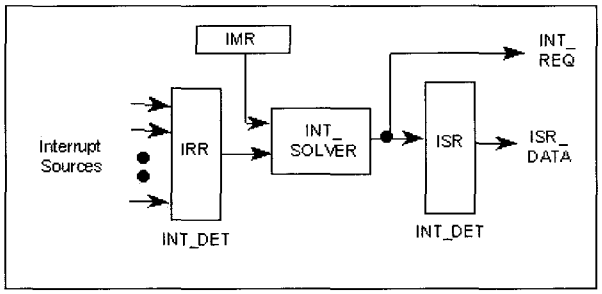


그림 8. 인터럽트 콘트롤 블록 다이어그램.
Fig. 8. Interrupt Control Block Diagram.

5. 인터럽트 콘트롤 블록의 구현

인터럽트 콘트롤 블록은 Interrupt Pin에 의한 처리를 지원하며, 총 8개의 인터럽트 소스에 대하여 우선 순위에 따라 인터럽트를 선택적으로 처리한다. 인터럽트 소스는 Edge Trigger Mode만 지원하며, 블록 다이어그램은 그림 8과 같다.

6. 기타 블록의 구현

기타 구현으로는 디버깅의 편리를 위해서 루프백(LoopBack) 모드를 지원하고, 소프트웨어 리셋 기능 및 현재 통신 상태를 나타내 주는 출력 포트를 두었다. 또한, 2 채널의 UART와 16 비트의 Discrete IO를 제공함으로써 범용적인 이용이 가능하도록 하였다.

7. 테스트 보드의 구현

본 테스트 보드는 Hitachi사의 H8/532 CPU를 사용하고, ROM 64K 바이트, RAM 32K 바이트로 구성되었다. 테스트를 위해 입력 16비트, 출력 16비트, 2개의 RS-485채널과 3개의 RS-232 채널을 구성하였다. FPGA는 Xilinx의 40만 게이트급인 XCV400HQ240-6C를 이용하여 구현하였다. 그림 9는 DNP 3.0 프로토콜을 구현한 FPGA를 탑재한 보드에 대한 자세한 사진이다.

8. 펌웨어의 구현

펌웨어는 Triangle MicroWorks의 DNP 슬레이브 라이브러리를 변형하여 사용하였다. C로 구현된 라이브러리 중 물리 계층과 데이터 링크 계층의 일부를 수정하여, 본 ASIC에 맞게

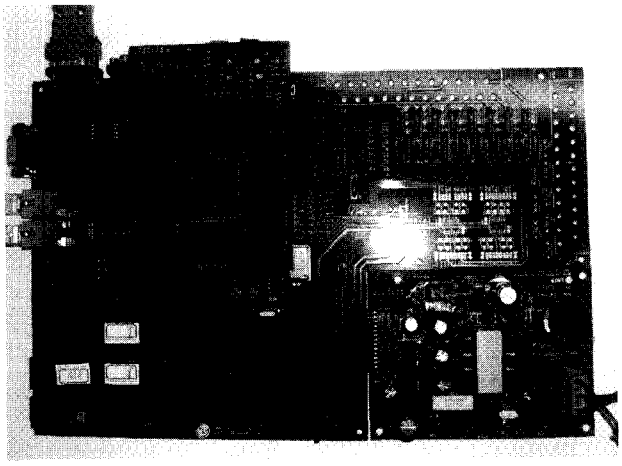


그림 9. DNP 3.0 프로토콜을 구현한 FPGA를 탑재한 보드.
Fig. 9. The Board with FPGA implementing DNP 3.0 Protocol.

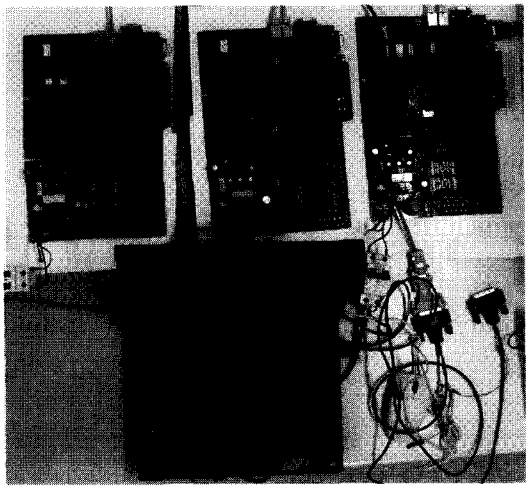


그림 10. 테스트 시스템의 구성.
Fig. 10. Test System Configuration.

동작하도록 하였다. H8/532 전용 컴파일러를 사용하여 컴파일하였으며, Emulator와 ROM 방식을 이용하여 디버깅하였다.

V. 실험 결과

1. 테스트 시스템 구성

테스트 환경은 그림 10에서 보여지는 바와 같이, 3개의 DNP 3.0 프로토콜을 구현한 FPGA를 탑재한 보드와 ASE2000이라는 DNP Master Simulator로 구성되어 있다. 일반 Uart 모드에서는 Master Simulator의 RS-232 포트와 연결하여 테스트를 수행하였고, 데이터 스트림 모드에서는 맨체스트 코딩을 Master Simulator에서 직접 구현할 수 없기 때문에, RS-232와 맨체스터 코드를 변환시켜주는 변환기를 만들어서 테스트를 수행하였다. 통신 속도는 19200Kbps로 맞추어 수행하였다.

2. 검증 결과

먼저 시뮬레이션 결과는 Cadence사의 NC-Verilog와 Simvision을 이용해서 검증하였다. 그림 11은 송신 블록에서 10개의 데이터를 FIFO에 써준 후에, Tx_Start 신호를 보내므로써, 송신

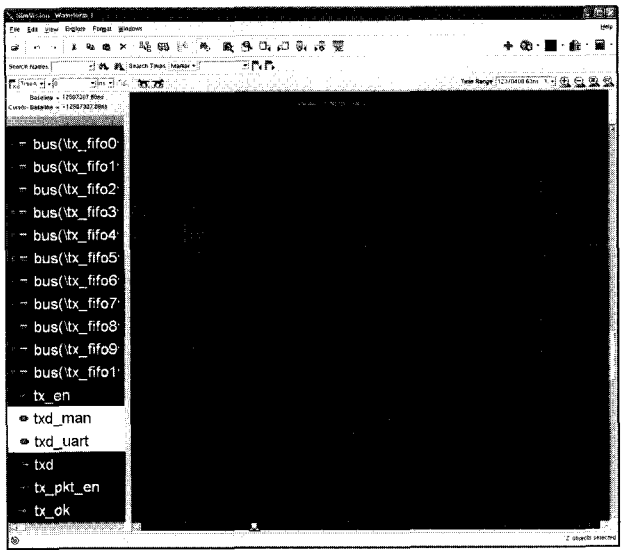


그림 11. 송신부의 시뮬레이션 파형.
Fig. 11. Transmit Unit Simulation Waveform.

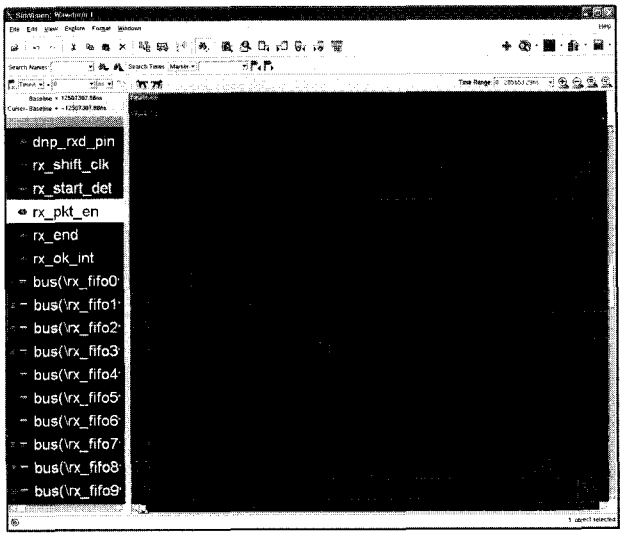


그림 12. 수신부의 시뮬레이션 파형.
Fig. 12. Receive Unit Simulation Waveform.

패킷이 생성되어서 나가고, 송신 동작의 완료를 알려주는 것을 볼 수 있다.

그림 12은 수신부의 시뮬레이션 파형을 보여준다. 이것은 0x0564부터 시작하여, 수신 패킷을 1번째 FIFO부터 차례대로 저장하는 것을 보여주며, 각 상태에 맞는 콘트롤 신호를 생성하는 것을 볼 수 있다.

실제 보드 테스트는 그림 13에서 보여지는 바와 같이 ASE2000 마스터 시뮬레이터를 이용하여 모니터링하였다. 마스터 시뮬레이터에서 각각의 보드에 어드레스를 다르게 주고, 각 보드에 패킷을 보내어 돌아오는 응답을 체크하였다. 다양한 종류의 요구 패킷에 대하여, 모든 DNP 슬레이브 보드가 정확한 응답을 보내는 것을 확인할 수 있었으며, 데이터 스트림 모드에서도 데이터의 응답을 확인할 수 있었다. 또한, CSMA/CD 테스트를 위해 브로드 캐스트 명령을 보내고 응답

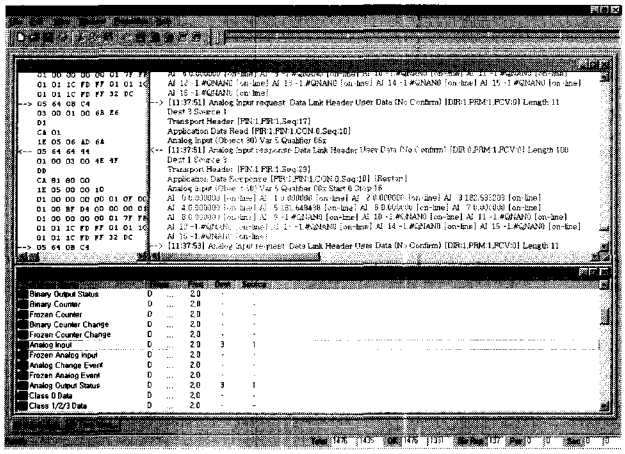


그림 13. ASE2000 마스터 시뮬레이터의 데이터 표시.
Fig. 13. Data Display of ASE2000 Master Simulator.

패킷을 검토한 결과, 3개의 DNP 슬레이브 보드에서 시간차를 두고 응답하는 것을 봄으로써 라인 활성화 상태 체크 및 랜덤 타임만큼 지연되어 송신을 수행하는 것을 확인할 수 있었다.

3. ASIC 설계 결과

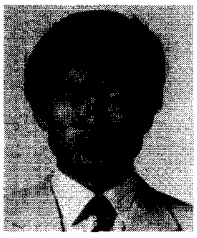
ASIC은 Hynix사의 0.8 micron GateArray를 사용하여 제작하였다. 전체적으로 41000개 정도의 게이트로 구성되며, 14mm X 20mm의 80핀 MQFP로 제작되었다. 현재 Fab In상태에 있으며, 8월초에 제작 완료될 예정이다.

VI. 결론

본 논문에서는 DNP 3.0 Protocol을 구현하였다. 기존의 UART를 이용한 통신 대신에, 물리 계층과 네크워크 계층을 HDL로 구현하여 하드웨어로 만들었다. 즉, 프레임의 동기화 및 어드레스 검출, CRC 비교 등의 기능을 하드웨어에서 처리함으로써, CPU의 부담을 덜어낼 수 있었다. 또한 추가로 데이터 스트림 모드를 구현함으로써 기존의 UART를 이용해서는 구현할 수 없었던 Unsolicited Response 모드를 실현할 수 있었다. 본 연구 결과를 이용하면, DNP 3.0 프로토콜을 사용할 때 보다 빠른 속도로, 다양한 기능을, CPU의 큰 부담 없이 사용할 수 있을 것이다. 하드웨어 부분이 HDL로 설계되어 IP로 구현되었기 때문에, DNP를 이용한 어떤 응용 제품을 구현한다고 하더라도 바로 적용이 가능하며, 이후 SCADA 시스템의 하드웨어 설계시 탁월한 선택이 될 수 있을 것이다. 향후 과제로는 보다 최적화된 하드웨어를 설계하여야 하며, 데이터 링크 레이어의 data link confirmation 요구시 자동으로 ACK 프레임을 생성하는 기능 등 프로토콜 동작 중 하드웨어로 추가로 구현할 수 있는 부분을 지원해야 할 것이다. 또한, 현재 진행 중인 ASIC이 입고된 후 실제 필드에서의 테스트를 통해 완전한 기능 검증을 수행해야 할 것이다.

참고문헌

[1] 홍승호, "필드버스 전개과정 및 발전전망," 월간 CONTROL, pp. 48-56, Sept., 1996.
[2] Intech: Field Buses Special Issue, ISA Publication, Nov., 1996.
[3] DNP 3.0 Overview, www.dnp.org, 1997.
[4] H8/532 Hardware Manual, 1995.
[5] Xilinx Device Data Book, 2002.
[6] Samir Palnitkar, Verilog HDL, Prentice Hall, the United States of America, 1996.
[7] 최병욱, 김정섭, 이창희, 김종배, 임계영, "EIA-709.1 Control Network Protocol을 이용한 필드버스 시스템의 구현", 제어·자동화·시스템공학 논문지 제6권, 제7호, pp.594-601, 7, 2000.

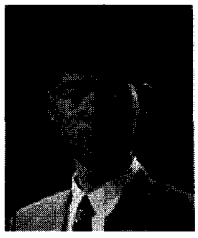


문 전 일

1960년 8월 25일생. 1984년 서울대 기계설계학과 졸업. 1986년 KAIST 생산공학과 졸업 (석사). 1998년 미국 Syracuse대학 산업공학과 졸업 (박사). 1987~ 현재 LG산전 중앙연구소 소장. 관심분야는 Embedded Controller, Networking & Communications, Embedded Linux OS.

김 종 배

1963년 1월 13일생. 1984년 중앙대 전자공학과 졸업. 1986년 동대학원 졸업 (석사). 1988~현재 LG산전 중앙연구소 책임연구원. 관심분야는 필드버스, SoC 및 ASIC.



김 정 섭

1972년 8월 24일생. 1995년 연세대학교 전기공학과 졸업.. 1997년 동대학원 졸업 (석사).2003년 9월~ 미국 Penn State대학 전기공학과 재학 중 (박사 과정). 1997~2003년7월 LG산전 중앙연구소 선임연구원. 관심분야는 필드버스, 실시간 제어 시스템 및 ASIC.

최 병 욱

제어 · 자동화 · 시스템공학논문지 제 4 권 제 4 호 참조

임 계 영

제어 · 자동화 · 시스템공학논문지 제 5 권 제 5 호 참조