G-PON TC 계층 유료부하 내에서 고속 GEM 프레임 동기회로 구현

종신회원 정 해*, 준회원 권 영 진*

Implementation of a High Speed GEM Frame Synchronization Circuit in the G-PON TC Sublayer Payload

Hae Chung* *Lifelong Member*, Youngjin Kwon* *Associate Member*

요 약

GEM 프레임은 G-PON 시스템에서 가변 사용자 데이터를 전달하는 수단이며 헤더와 유료부하로 구성된다. 헤더의 HEC 필드는 헤더의 내용을 보호하고 동시에 GEM 프레임 동기를 유지할 목적으로 사용된다. 수신 중에 GEM 프레임 동기를 잃어버리면 다시 동기를 획득 할 때까지 프레임들은 폐기되어야 한다. 따라서 손실되는 프레임의 수를 최소화하기 위해서는 고속의 동기모듈이 필요하다. 본 논문에서는 GEM 헤더에 검출이 불가능한 에러가 나타났을 때 발생하는 프레임 손실을 줄이기 위하여 주 상태머신 이외에 부 상태머신의 사용을 제안하고 이를 구현한다. 또한 헤더의 시작점을 찾는데 있어서 고속이며 동시에 효율적인 병렬 구조를 제안한다. 최종적으로, 제안된 방식은 FPGA를 통해 구현하였고 계측기를 이용하여 검증한다.

Key Words : G-PON; FTTH; GEM Frame; Synchronization; HEC.

ABSTRACT

The GEM frame is used a mean to deliver the variable length user data and consists of the header and the payload in the G-PON system. The HEC field of header protects contents of the header and is used to maintain GEM frame synchronization at the same time. When an LCDG (Loss of GEM Channel Delineation) occurs while receiving frames, the receiver have to discard corrupted frames until acquiring the synchronization again. Accordingly, high-speed synchronization method is required to minimize the frame loss. In this paper, we suggest not only a main state machine but a sub-state machine to reduce the frame loss when undetectable errors occurred in the GEM header. Also, we provide a more efficient and fast parallel structure to detect the starting point of the header. Finally, the proposed method is implemented with the FPGA and verified by the logic analyzer.

Ⅰ. 서 론

TPS (Triple Play Service)란 하나의 회선을 통해 영상, 음성, 인터넷 세 가지의 서비스를 모두 공급하는 것을 이야기한다. 하나의 회선에 제공되는 서비스의 종류가 증가하는 만큼 TPS 서비스를 원활히 제공하기 위한 대역폭 또한 증가한다. 특히 IPTV (Internet Protocol Television) 서비스와 같이 고화질 광대역 서비스를 제공하기 위해서는 광대역 채널의 확보가 필수적이다. 이와 같은 광대역 채널

※ 이 논문은 2008년도 정부재원 (교육인적자원부 학술연구조성사업비)으로 한국학술진흥재단의 지원을 받아 연구되었음 (KRF-2008-521-D00317)

* 금오공과대학교 전자통신과 통신망연구실 (hchung@kumoh.ac.kr)

논문번호 : KICS2009-02-079, 접수일자 : 2009년 3월 3일, 최종논문접수일자 : 2009년 4월 24일

을 확보하기 위한 해결책으로 기존의 광통신망을 확대하여 댁내에까지 광케이블을 통해 신호를 전송하는 방식이 제안되었고, 이러한 방식을 FTTH (Fiber to the Home)라고 한다[1].

FTTH를 구성하는 방식은 크게 능동소자를 사용하는 AON (Active Optical Network)과 수동소자를 사용하는 PON (Passive Optical Network)으로 나누어 볼 수 있다. PON은 다시 TDMA (Time Division Multiple Access) 방식과 WDMA (Wavelength Division Multiple Access) 방식으로 나눌 수 있으며, TDMA 방식은 B-PON (Broadband PON), E-PON (Ethernet PON), G-PON (Gigabit capable PON)으로 나눌 수 있다[1],[2]. 그 중 B-PON 방식이 일찍이 시장에 진출하였지만, IP (Internet Protocol) 기반의 서비스가 용이하고 B-PON에 비해 2배 가까운 최대전송률을 가지는 E-PON 방식에 밀려 시장에서 도태되었다. 그에 반해 E-PON 방식은 현재 PON 시장의 주류가 되었다[3].

G-PON 방식은 TDMA 방식 중에서 가장 최근에 규격화가 이루어진 방식이며, GEM (G-PON Encapsulation Method)이라는 구조를 통해 IP 기반의 서비스와 가변길이의 데이터들도 수용이 가능하도록 하여 B-PON에 비해 더욱 보완된 방식이라고 할 수 있다. 또한 E-PON에 비해 2배 이상의 최대전송률을 제공할 수 있기 때문에 E-PON에 비해 가격이 다소 고가인 단점에도 불구하고 북미와 두바이를 비롯한 국내외에 그 시장을 확대해 가고 있다[1],[4],[5].

PON 방식의 가장 큰 수요자인 IPTV 시장은 그 규모와 가능성이 더욱 커지고 있으며, IPTV 시장 선점을 놓고 망사업자간의 경쟁이 더욱 가열되고 있다. 이에 높은 전송대역폭과 다양한 서비스에 유리한 G-PON 시스템에 망사업자들의 관심이 모아지고 있다[6]~[9].

G-PON의 이러한 가능성에도 불구하고 국내 시스템업체들의 문제로 지적되는 점은 G-PON 시스템에 들어가는 핵심 칩의 국산화가 아직 이루어지지 않았다는 것이다. 이에 국내기술로 G-PON TC (Transmission Convergence) 칩을 구현하는 연구가 진행되고 있으며[1], 본 논문에서는 TC 칩의 서브모듈인 GEM HEC (Header Error Control) 모듈의 구현에 대해 논할 것이다.

GEM HEC 모듈이란 GEM 프레임의 헤더를 부호화하여 전송하고 수신된 GEM 프레임의 헤더를 복호화하고 동기 상태를 판별함으로써 정정 가능한 에러를 정정하고 사용자 데이터를 보호하는 모듈이다. 여기서 언급한 GEM 프레임은 G-PON 시스템에서 ATM (Asynchronous Transfer Mode)을 제외한 모든 사용자 데이터에 대하여 일괄적인 처리가 용이하도록 재구성한 프레임이다.

GEM 프레임 헤더의 복호화는 39 비트의 BCH (Bose Chaudhuri Hoequengham)와 1 비트의 패리티 검사를 거쳐 이루어진다. BCH와 패리티 검사를 통해 판별된 값을 이용해 모듈내부의 상태머신은 동기 상태를 유지하거나 변화시킨다. 에러가 일정 수준이상 발생하게 되면 수신단의 GEM HEC 모듈은 동기를 잃어버리게 된다. 이렇게 동기를 잃어버리게 되면, 헤더를 통해 수신된 데이터를 신뢰 할 수 없게 되고 데이터의 손실이 발생한다. 따라서 동기를 잃어버린 시간이 길어지면 길어질수록 손실되는 데이터의 양이 증가하게 되어 채널 효율의 저하로 이어진다.

본 논문의 목적은 GEM 프레임의 헤더에 에러가 발생하여 동기를 잃었을 경우 (LCDG, Loss of GEM Channel Delineation) 최단시간에 동기를 회복할 수 있는 모듈을 제안하고 최소한의 로직소모를 통하여 구현함으로써 G-PON 시스템 상에서의 사용자 데이터 손실을 최소화하고 채널의 효율을 높이는 것이다.

논문의 구성은 2절에서 G-PON 프레임의 구조 및 GEM 프레임 동기에 대해 소개하고, 3절에서는 구현된 GEM 프레임 동기회로에 대해 설명하고, 4절에서는 구현된 동기모듈을 검증한다. 그리고 마지막으로 5절에서 결론을 맺는다.

Ⅱ. G-PON 프레임 구조 및 GEM 프레임 동기의 개요

G-PON 시스템은 대칭 및 비대칭 전송률을 제공한다. 어떠한 전송률을 지원하든 한 프레임은 125 μs의 구조를 가지고 있으며, 동일한 하향프레임 구조를 가지게 된다. 그림 1은 이러한 하향 프레임 구조를 보여준다. 이 프레임은 크게 오버헤더 영역인 PCBd (Physical Control Block downstream)와 유료부하 영역으로 구분된다. 유료부하는 다시 ATM 셀 섹션과 GEM 섹션으로 나뉜다.

GEM 섹션은 GEM 프레임들로 구성되어 있으며 각 GEM 프레임은 헤더와 유료부하로 구성되어 있다. 그림 1에서와 같이 전송할 프레임이 없는 경우에도 수신 측에서 동기를 유지하도록 하기 위하여

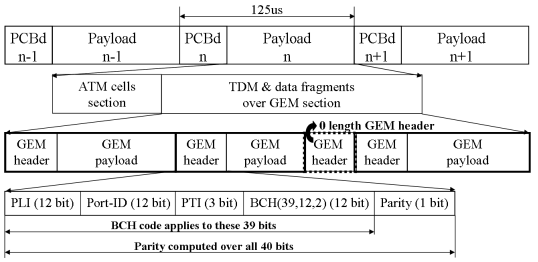

그림 1. G-PON 하향 프레임 구조

Fig. 1. G-PON downstream frame structure.

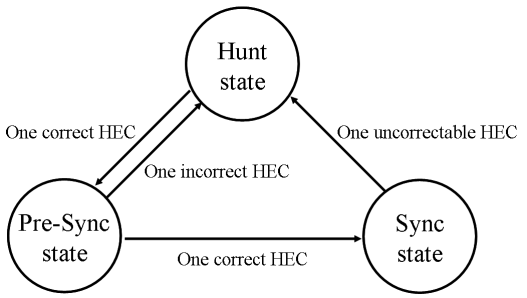

그림 2. GEM 동기 상태머신

Fig. 2. GEM sync state machine.

유료부하의 길이가 0인 프레임을 삽입한다. 이 경우에는 PLI (Payload Length Indicator) 값이 0이 된다. 그림 1의 가장 아래 표시된 블록은 GEM 헤더의 구조와 함께 HEC의 수행 범위를 보여준다. BCH 계산 수행범위는 PLI의 시작부터 39 비트까지이고 패리티 계산 수행범위는 PLI의 시작부터 40 비트까지이다. 이때 패리티 비트는 우수 패리티를 이용한다.

PLI 필드를 읽으면 GEM 프레임의 길이를 알 수 있으며, 다음 GEM 헤더가 나올 위치를 계산할 수 있다. 또한 GEM 프레임의 유료부하 길이가 4095 바이트를 초과하면 프레임을 분할하여 보내는데 수신 측에서는 PTI(Payload Type Indicator)를 통하여 분할된 프레임을 다시 재조립 할 수 있다. 중요한 것은 GEM 헤더를 통해 얻는 정보들은 GEM 동기가 획득되었을 때 그 의미를 가진다는 사실이다.

그림 2는 GEM 동기 상태머신을 나타낸 그림이다[2]. 상태머신은 추적 상태 (Hunt state), 준동기 상태 (Pre-sync state), 동기 상태 (Sync state)의 3 가지 상태를 가지고 특정 조건에 따라 상태가 바뀌게 된다.

수신된 데이터에 에러가 없다면 GEM 프레임은 항상 동기 상태를 유지할 것이다. 그러나 수정 불가능한 헤더오류가 발생하면 동기를 잃었다고(LCDG) 판단하고 추적 상태가 된다. 수정 불가능한 헤더오류는 3 개 이상의 에러가 검출된 경우를 말한다. 추적 상태가 되면 에러가 없는 GEM 헤더가 나올 때까지 헤더의 위치를 추적한다. 에러가 없는 GEM 헤더가 발견 되면 준동기 상태가 되며 다음 헤더의 위치를 예측하게 된다. 예측된 다음 헤더의 위치에서 다시 한 번 에러가 없는 헤더가 검출되면 동기 상태를 회복하게 된다. 그러나 예측된 위치에서 하나 이상의 에러를 가지는 헤더가 발생하면 다시 추적 상태가 되어 올바른 GEM 헤더의 위치를 계속 추적해 나갈 것이다.

GEM 동기를 잃어버리면, GEM 헤더의 PLI 값을 신뢰할 수 없게 되고 다음 헤더의 위치를 예측할 수 없게 된다. 헤더의 위치를 예측할 수 없게 되면 정확한 헤더 해석이 불가능하게 되고, 헤더 값을 기준으로 수행되는 일련의 동작 들을 할 수 없다. 결국 헤더를 해석할 수 없는 GEM 프레임은 버려지고 이것은 전송효율을 크게 떨어뜨리는 요인이 된다.

제안된 GEM 프레임 동기회로는 이와 같이 GEM 동기를 잃어버렸을 때 다시 동기를 회복하는 시간을 최소화하고 검출 불가능한 에러가 발생하였을 때 손실되는 GEM 프레임의 수를 최소화할 목적으로 설계한다.

Ⅲ. GEM 프레임 동기회로

GEM 프레임 동기회로는 GEM 프레임에 대해서만 적용되며, ATM 셀에 대해서는 적용되지 않는다. 또한 ATM 셀의 경우 고정된 길이를 사용하기 때문에 각 ATM 셀의 위치를 예측하는 것이 어렵지 않다. 또한 최근의 규격에서 ATM은 제외되었기 때문에[2], 가변적인 길이를 가지는 GEM 프레임에 대하여 중점적으로 논하기로 한다.

3.1 GEM 프레임 동기회로의 전체 구성

GEM 프레임의 동기회로는 그림 3에서 보는 바와 같이 크게 HEC 계산기, LUT (Look-up Table), 플립플롭, 상태머신으로 구성되어 있다. LUT는 일반적인 ROM (Read Only Memory)으로 구현된다.

수신기의 GEM 프레임 동기회로는 GEM 헤더의 PLI 값을 이용하여 다음 헤더가 나올 위치를 예측한다. 그러나 수신기가 동기를 잃어버리면 다음 헤더의 위치를 예측할 수 없으며 동기를 회복할 때까지 수신되는 모든 프레임들을 폐기해야 한다. 이러한 데이터 손실을 최소화하기 위해서는 최단 시간

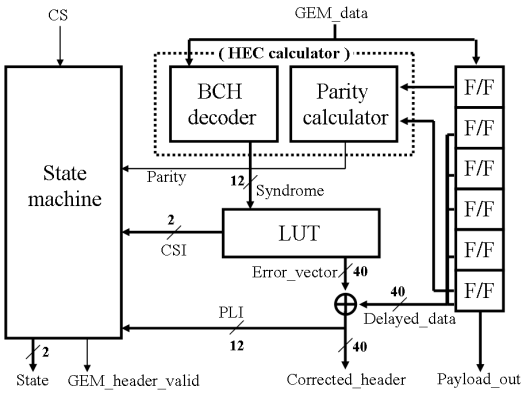

그림 3. GEM 프레임 동기회로도

Fig. 3. Circuit diagram for GEM frame synchronization.

내에 동기를 회복하는 것이 중요하다. 본 논문에서는 이러한 동기회복 시간을 최소화하기 위해 매 바이트 클럭마다 빠짐없이 5 바이트 단위의 검사를 수행 하도록 한다. 단 HEC 모듈에 앞서 전송로에서 입력된 신호가 기가급 트랜시버를 통해 비트동기를 획득하였으며 다시 바이트 동기모듈을 거치면서 바이트 동기까지 획득하였다고 가정한다[1].

그림 3에서 보이는 HEC 계산기는 BCH 복호기와 패리티 계산기로 구성된다. BCH 복호기는 매 클럭마다 5 바이트의 구간에 대해 12 비트의 신드롬 값을 출력하며 패리티 계산기는 1 비트의 패리티 값을 출력한다. 출력된 신드롬 값은 LUT로 입력되며 패리티 결과 값은 상태머신으로 입력된다.

LUT는 입력된 신드롬을 주소로 하여 저장되어있는 40 비트의 에러 벡터 (Error vector)와 정정된 상태 정보를 나타내는 2 비트의 CSI (Correct Status Information)를 출력한다. CSI 신호는 상태머신의 입력으로 들어가며 에러가 없을 경우는 00, 1 개의 에러가 발견된 경우는 01, 2 개의 에러가 발견된 경우는 10, 3 개 이상의 에러가 발견된 경우는 11의 값을 가진다. LUT에서 출력되는 Error_vector는 Delayed_data와 배타적 논리합 (XOR) 연산을 한다. Delayed_data는 입력된 GEM_data를 플립플롭을 이용해 6 클럭 지연시킨 40 비트의 데이터이며 연산 결과는 정정이 끝난 헤더를 의미하는 Corrected_header이다.

전술한 부분 외에도 동기회로에는 상태머신이 함께 구성되어 있다. 상태머신의 역할은 CS (Chip Select), Parity, CSI, PLI 값을 입력으로 받아 다음 상태를 판별하고 판별된 상태를 기준으로 State, GEM_header_valid 출력을 내보내는 것이다.

상태머신의 입력 중 CS 신호는 그림 1에서 보는 것처럼 G-PON 전체 프레임 중에서 GEM 프레임이 존재하는 기간 동안에만 high를 유지하는 1 비트 신호이다. CS 신호가 low가 되면 상태머신은 초기화되며, high가 되면 GEM_data가 입력되기 시작하였다고 판단하고 동작한다. 패리티 신호와 CSI 신호는 HEC 결과를 판별하는데 사용되며 이 결과는 다음 상태를 결정하는 중요한 기준이 된다. PLI 신호는 정정이 끝난 Corrected_header의 상위 12 비트로써 GEM 프레임의 유료부하의 길이를 나타내며 상태머신의 입력으로 사용된다.

상태머신에 출력되는 GEM_header_valid 신호는 Corrected_header를 통해 출력되는 수많은 5 바이트 단위 묶음 중에서 이용 가능한 GEM 헤더를 알려준다. 여기서 이용이 가능한 GEM 헤더의 기준은 ITU-T G.984.3에서 언급하고 있는 GEM의 동기상태에서 얻어지는 헤더 값이다. 그림 1의 G-PON 프레임 구조에서 볼 수 있듯이 GEM 섹션의 시작은 GEM 헤더로 시작하기 때문에 모든 GEM 시작부에서는 동기 상태로 출발한다고 볼 수 있다 [2]. 상태머신의 출력 중 State 신호는 00은 동기 상태, 01은 추적상태, 10은 준동기 상태를 표현한다.

3.2 HEC 부호기

HEC 부호기는 송신부에 필요한 모듈이나 수신부에 사용되는 복호기의 이해를 돕기 위하여 여기서 설명하기로 한다. 이 부호기는 BCH (39, 12, 2) 계산기와 패리티 계산기로 이루어진다. 여기서 39는 보호해야 할 헤더의 길이, 12는 패리티를 제외한 HEC의 길이, 2는 정정 가능한 에러의 개수를 의미한다. 부호기는 PLI 12 비트, Port-ID 12 비트, PTI 3 비트와 zero 13 비트를 입력으로 받아 12 비트의 BCH와 1 비트의 패리티를 포함한 5 바이트 출력을 내보낸다.

송신단의 BCH 계산기는 생성다항식 $g(x)=x^{12}+x^{10}+x^8+x^5+x^4+x^3+1$으로 입력 데이터를 나눈 나머지 값 12 비트를 40 비트의 헤더 중 28 번째부터 39 번째 자리에 삽입하여 출력한다. 송신단의 패리티 계산기는 BCH 계산기를 통해 출력된 5 바이트 출력 값에 대해 짝수 패리티 값을 계산하여 40 번째 자리에 삽입하여 출력한다.

송신단에서 BCH 계산 결과 값이 하위 비트에 더해져 송신이 되면 수신단에서는 동일한 생성다항식을 이용하여 BCH 계산을 수행한다. 이때 송신단에서 BCH 계산 결과를 더하여 송신하였기 때문에

수신 중에 에러가 없었다면 BCH 계산 결과 값은 항상 zero가 된다. 그렇지 않을 경우 BCH 계산 결과는 특정한 신드롬 값을 가지게 되고 신드롬 값과 패리티 값을 이용하여 2 비트 에러까지 정정할 수 있다.

3.3 BCH 계산기

그림 4는 설계된 BCH 계산기를 나타낸 그림이다. BCH 계산기는 8 비트의 입력이 5 회에 걸쳐 (5 클럭) 입력될 때 13 비트의 생성다항식으로 나눗셈을 수행하여 6 번째 클럭에서 신드롬을 출력하는 회로이다. 그리고 이것은 BCH 복호기를 구성하는데 필수적인 모듈이다. 이때, BCH 계산기의 입력 값 39 비트 중 하위 12 비트는 송신단에서 zero로 입력될 것이며 수신단에서는 계산된 BCH 결과 값이 입력될 것이다.

설계된 BCH 계산기의 전체적인 구현과정을 설명하면 다음과 같다. 그림의 BCH 계산기는 39 비트의 입력 데이터에 대해서 계산을 수행하여야 하지만 실제로 입력은 한 클럭에 8 비트씩 입력이 된다. 이에 대해 39 비트의 데이터를 모두 받아들일 때까지 저장하였다가 한 번에 계산을 수행 할 수도 있지만 이렇게 하면 로직의 소모가 커지고 한 클럭 내에 수행하는 연산이 복잡해진다. 따라서 본 논문에서는 현재 클럭에서 입력된 데이터를 제외한 나머지 부분을 zero로 가정하고 먼저 계산을 수행한다. 이에 대해서 다음 클럭에 입력되는 8 비트 데이터는 Ⓐ와 같이 플립플롭의 상위 12 비트와 배타적 논리합을 먼저 취해줌으로써 앞서 다음에 어떤 값이 입력될지 모르는 비트에 대해서 zero로 가정하고 계산한 결과 값에 대해 보상을 해준다.

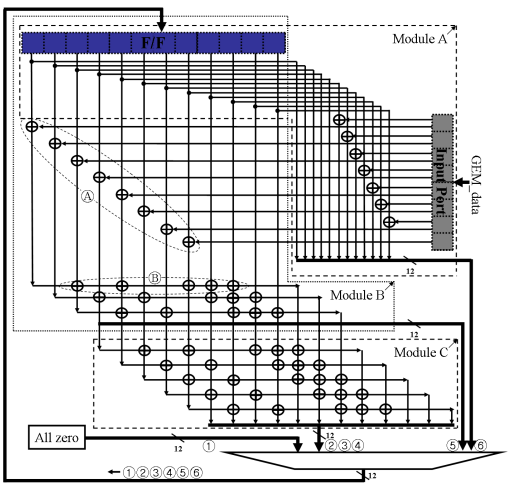

그림 4. BCH 계산기
Fig. 4. BCH calculator.

Ⓑ의 형태는 생성다항식에 의해서 결정되며 생성다항식과 입력된 8 비트 데이터의 각 한 비트 곱에 대해서 mod 연산을 수행하는 부분이다. 그림의 BCH 계산기는 매 클럭 8 비트 데이터에 대한 계산을 수행하기 위해 8 개의 Ⓑ를 가지고 있다. 또한 Ⓑ는 어떤 데이터와 zero를 배타적 논리합을 취한 값은 아무 연산도 하지 않은 값과 같다는 점을 이용하여 최적화한 형태이다.

그림의 BCH 계산기는 12 비트의 플립플롭을 이용하여 계산결과를 저장한다. 매 클럭마다 플립플롭의 출력은 입력 데이터와 연산을 거쳐 다시 플립플롭으로 입력된다. 플립플롭으로 입력되는 값은 6 클럭의 주기를 가지고 그림에서 보이는 ① 번부터 ⑥ 번까지 변한다. 이러한 일련의 과정은 입력되는 39 비트의 데이터를 8 비트 단위로 차례로 연산하기 위한 과정이다. ① 번 입력은 12 비트 플립플롭을 초기화하기 위한 입력이다. ② 번, ③ 번 그리고 ④ 번 입력은 8 비트씩 입력되는 데이터에 대해서 Ⓐ를 거쳐 이전 클럭의 계산결과를 보상해주고 Ⓑ를 8 개 거치면서 8 비트에 대한 mod 연산을 수행하여 나온 결과이다. ⑤ 번 입력은 39 비트의 입력 중 25번째부터 27번째 비트까지의 연산만을 수행하여야 하기 때문에 Ⓑ를 3 번만 거쳐 3 비트에 대한 mod 연산을 수행하였다. 이렇게 39 비트 중 27 비트에 대해서 13 비트의 생성다항식으로 나눗셈을 수행하면 나머지 값은 12 비트가 된다. ⑥ 번 입력은 모듈 A를 거쳐서 출력되는데, 이것은 마지막으로 입력되는 8 비트 데이터는 BCH 계산기의 입력 39 비트 중 하위 7 비트에 해당하는 값이며 마지막 비트는 패리티 값으로 BCH 계산기에서 사용되는 값이 아니기 때문이다. 이렇게 ⑤ 번 입력에서는 8 비트 중 하위 5 비트에 대해서 Ⓑ 연산을 하지 않고 Ⓐ를 통해 배타적 논리합 연산만 수행하였고 ⑥ 번 입력에서는 입력된 8 비트 중 상위 7 비트에 대해서만 배타적 논리합 연산을 수행하였다. 결론적으로 39 비트의 BCH 입력 중 27 비트에 대해서 Ⓑ 연산이 수행되어졌고 나머지 12 비트에 대해서는 플립플롭의 출력과 배타적 논리합 연산만 수행되었다. 이것은 BCH 입력 39 비트 중 상위 27 비트에 대해서 나눗셈을 수행하여 얻은 12 비트 계산 결과를 하위 12 비트에 더해줌으로써 송신단에서는 BCH 12 비트를 얻을 수 있고, 수신단에서는 신드롬 12 비트를 얻을 수 있음을 보여준다.

3.4 BCH 복호기

그림 5는 일반적인 BCH 복호기 구성도를 보여주고 있다. 그림에서 보이는 구조는 앞서 소개한 그림 4의 BCH 계산기 6 개를 병렬로 연결한 형태이다. 5 개가 아닌 6 개의 조합이 필요한 이유는 1 클럭의 초기화 시간이 필요하기 때문이다. 또한 병렬로 설치된 계산기와 다중화기에 적절한 Enable 신호와 Sel_syndrome 신호를 입력해주기 위한 CSG (Control Signal Generator)가 필요하다. CSG는 CS 신호를 입력받아 순차적으로 BCH 계산기들에게 Enable 신호를 입력시키고 그에 따라 BCH 계산기들에게서 출력되는 신드롬에 맞는 Sel_syndrome 신호를 다중화기에 공급하는 역할을 한다. Sel_syndrome을 입력받은 다중화기는 각 BCH 계산기에서 입력되는 신드롬을 순차적으로 출력한다.

그림 6은 본 논문 구현에서 최종적으로 쓰인 형태이며 병렬구성에 더욱 효율적인 BCH 복호기 구성 방법을 보이고 있다. 그림의 BCH 복호기의 경우 각 Step마다 특화된 계산 로직을 사용하여 로직의 효율성을 높인 것이 특징이다. Step$i$에서는 (단, $1 \le i \le 5$) 헤더 5 바이트 중에서 i 번째 바이트에 대한 BCH 계산을 수행한다. Step0을 제외한 각 Step에는 앞서 보인 그림 4의 BCH 계산기의 일부 (모듈 A, B, C)가 포함되는 것을 볼 수 있다. 각 Step에서는 헤더 5 바이트에 대해서가 아니라 매 클럭 자신의 Step에 맞는 바이트에 대해서만 계산을 수행하고 계산 결과를 다음 단에 전달한다. 그로 인하여 그림 6과 같이 GEM_data가 각 Step에 동일하게 1 바이트 단위로 입력된다면 마지막 레지스터에는 매 클럭 헤더 5 바이트에 대한 신드롬이 입력된다. 따라서 그림 6과 같은 구성에서는 그림 5에서와 같이 다중화기와 CSG가 필요하지 않은 장점이 있다.

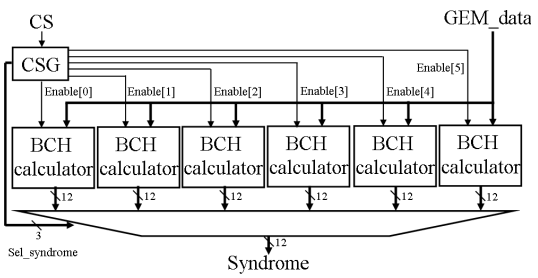

그림 5. 일반적인 BCH 복호기 구조
Fig. 5. Typical BCH decoder architecture.

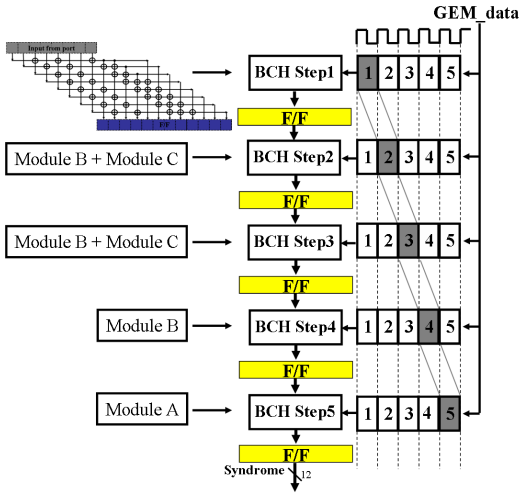

그림 6. 효율적인 BCH 복호기 구조
Fig. 6. Efficient BCH decoder architecture.

3.5 패리티 계산기

그림 7은 HEC 계산기에서 GEM 헤더 40 비트에 대하여 패리티 계산을 수행하는 모듈을 나타낸 것이다. 그림에서 눈금으로 표시한 부분이 패리티 계산기를 나타내는 것이며 그림에 있는 6 바이트의 플립플롭은 그림 3에 주어진 것과 동일한 것이다. 패리티 계산기로 입력되는 데이터는 2 가지이며, 각각 GEM_data가 1 클럭 지연된 것과 6 클럭 지연된 것이다. 이와 같이 5 클럭 차이나는 두 가지 입력과 이전에 계산되어진 패리티 값을 모두 배타적 논리합을 취하게 되면 우측의 그림과 같이 매 클럭 새로운 5 바이트 구간에 대한 패리티 계산 값을 얻을 수 있다. 설계된 패리티 계산기는 전체적으로 1 클럭 지연된 입력을 받는데 이것은 이후에 상태머신으로 CSI 신호와 Parity 신호가 동일한 시점에 입력되도록 하기 위함이다. 그림의 패리티 계산기는 6

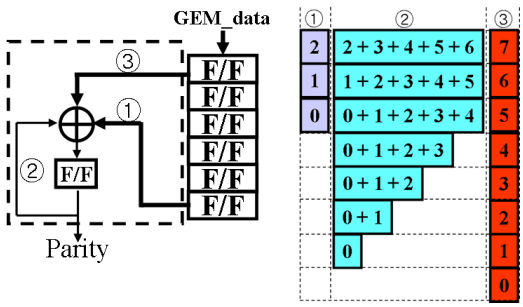

그림 7. 패리티 계산기
Fig. 7. Parity calculator.

바이트의 플립플롭을 필요로 한다. 그러나 본 논문에 설계된 GEM 프레임 동기회로는 에러정정을 위해 같은 플립플롭을 이미 사용하여 로직을 재활용하고 있기 때문에 추가적인 플립플롭 소모 없이 구현이 가능하다.

3.6 상태머신

그림 8은 설계된 동기회로의 상태머신의 구조를 나타낸 것이다. 상태머신은 CSI, CS, Parity, PLI 네 가지 값을 입력으로 받아 State 값과 Header_valid 값을 출력한다. 전체 상태머신 내부에는 두 개의 상태머신이 있으며, 주 상태머신과 부 상태머신으로 나눠 볼 수 있다. 출력 값 Header_valid는 주 상태머신의 Main_header_valid와 부 상태머신의 Sub_header_valid의 합(OR)이 되며, State 출력은 주 상태머신에만 영향을 받는다.

그림 9는 주 상태머신의 순서도를 나타낸 그림이다. 순서도는 클럭이 입력될 때마다 한 번씩 수행된다. GEM 데이터의 존재유무를 알리는 CS 신호가 0 일 때, 상태머신은 초기화된다. 상태는 동기 상태로 초기화되고 PLI_cnt는 구현의 편의상 3으로 초기화 한다. 주 상태머신은 추적 상태, 준동기 상태, 동기 상태의 세 가지 상태를 가지며, 이외의 경우가 발생하면 추적 상태를 디폴트값으로 가진다. 각 상태에서는 주요한 판단시점을 가지는데 추적 상태에서는 매 클럭 정상적인 헤더가 발생하였는지를 관찰하고, 준동기 상태와 동기 상태에서는 PLI_cnt를 기준으로 0x1ffd일 때 HEC 계산결과를 판단하고, 0x1ffc일 때 PLI_cnt 값을 Preset하게 된다. PLI_cnt는 매 클럭마다 1 씩 감소되기 때문에 항상 HEC 계산결과를 판단하는 작업이 먼저 일어나고 그 결과에 따라 PLI_cnt 값을 Preset한다.

상태머신의 출력 Header_valid는 전체 출력 Corrected_header의 값이 이용 가능한 헤더의 값을 가졌을 때 high가 된다. 이때의 주 상태머신의 동작

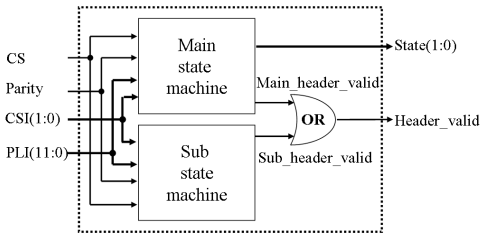

그림 8. 상태머신의 구조
Fig. 8. State machine architecture.

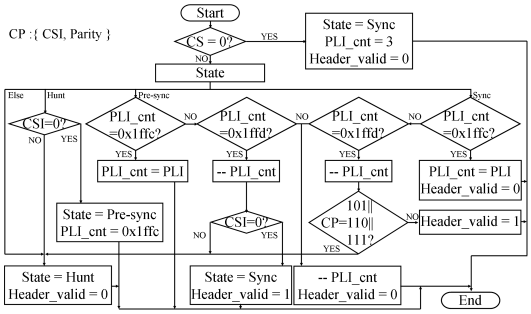

그림 9. 주 상태머신 순서도
Fig. 9. Main state machine flow diagram.

을 살펴보면 PLI_cnt = 0x1ffd 시점에 검사를 수행하여 그 결과에 의해 다음 상태가 동기 상태가 될 때 Header_valid = 1이 되는 것을 볼 수 있다.

그림 10은 GEM 헤더 검출에 실패하는 특수한 경우에 대한 그림이다. 그림과 같이 헤더에 검출되지 않는 에러가 발생하면 GEM 동기회로는 잘못된 PLI 값을 이용해 다음 헤더의 위치를 예측한다. 또한 예측된 다음 헤더의 위치까지 상태머신은 현재 상태를 유지하게 되고 이것은 그 사이에 수신되는 GEM 프레임들의 손실로 이어진다. 만약 에러로 인해 값이 바뀐 잘못된 PLI의 크기가 올바른 PLI의 크기보다 큰 경우에 주 상태머신은 이후 입력되는 올바른 GEM 프레임을 최소 2 개 이상 놓치게 된다. 이런 현상은 그림 10에서 보듯이 이전 상태가 추적 상태, 준동기 상태일 때는 물론 동기 상태일 때도 일어날 수 있다. 특히 동기 구간에서는 우연히 에러가 없는 헤더라고 판별된 경우뿐만 아니라 사실은 에러가 있음에도 수정 가능한 헤더로 판별될

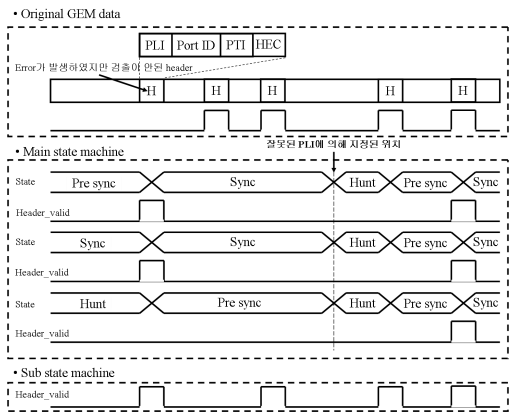

그림 10. GEM 헤더 검출에 실패하는 특수한 경우
Fig. 10. Special case failed to detect the GEM header.

경우에도 이런 상황이 야기될 수 있다.

이렇게 잘못된 PLI에 의해 손실되는 GEM 프레임들을 최소화하기 위해서 본 논문에서는 주 상태머신 이외에 부 상태머신을 추가적으로 삽입하였다. 가장 아래 표시된 Header_valid 신호가 부 상태머신에 의해서 발생된 값이며 주 상태머신이 미처 잡지 못한 GEM 헤더를 찾아내서 표시 해줄 수 있음을 보이고 있다.

그림 11에서는 부 상태머신이 어떻게 주 상태머신이 놓친 GEM 헤더를 찾아내는지 순서도 통해 보이고 있다. 부 상태머신의 특징은 PLI_cnt 값을 기준으로 동작하지 않는다는 것이다. 주 상태머신이 PLI_cnt 값을 통해 헤더의 위치를 예측한 것에 반해 부 상태머신은 HEC 수행결과 에러가 없는 헤더가 검출된 지점을 헤더의 위치로 의심하고 이때의 Sub_PLI_cnt 값을 확인해 봄으로써 PLI 값과 일치하는 곳에서 헤더가 나타났는지 판별하게 된다. 그리고 그 판별한 결과에 따라 Sub_header_valid = 1을 출력하게 되고 이 값은 전체 상태머신의 출력에 OR 게이트로 더해져 출력된다. 부 상태머신에서 Sub_header_valid = 1이 되기 위한 조건은 정상적인 헤더를 검출하고 검출된 PLI 값이 예측하는 위치에서 또 한 번 정상적인 헤더가 검출되는 것이다. 이 조건은 단순해 보이지만 ITU-T G.984.3에 명시하는 상태도를 살펴보면 어느 상태에 상관없이 위의 조건을 만족하면 동기 상태가 되도록 나타나있다[2].

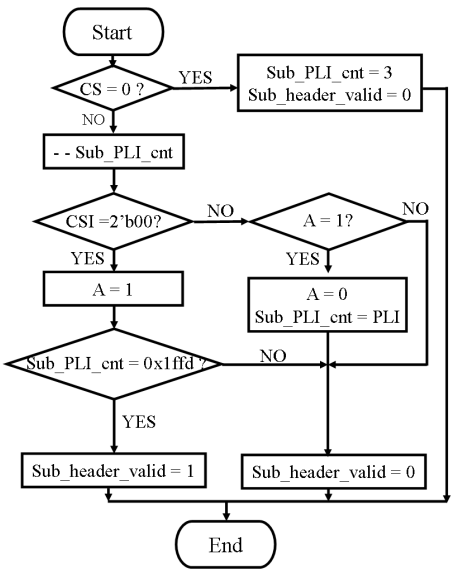

그림 11. 부 상태머신 순서도

Fig. 11. Sub-state machine flow diagram.

주 상태머신 외에 부 상태머신을 추가함으로써 우연히 잘못된 Header_valid = 1을 추가적으로 발생시킬 확률은 극히 낮다. 그 경우는 에러가 나지 않은 정상적인 두 개의 헤더 사이에 우연히 두 개의 5 바이트 데이터가 HEC 검사에 의한 에러가 검출되지 않고 PLI 값과 두 데이터 사이의 간격이 일치되는 경우이다.

그림 10에서 언급한 경우가 우연히 정상적인 헤더가 발생되고 그 PLI 값이 원래의 PLI 값보다 큰 경우인 것과 비교해보면 훨씬 낮은 확률이라고 볼 수 있다. 또한 동기 상태에서는 수정 가능한 헤더오류가 발생되고 그 PLI 값이 원래 PLI 값보다 큰 경우에도 그림 10에서 언급하는 손실이 발생하게 된다. 따라서 부 상태머신의 Sub_header_valid와 주 상태머신의 Main_header_valid를 OR게이트를 통과시켜 Header_valid를 출력하게 하면 성능향상을 기대할 수 있다.

3.7 GEM 프레임 동기회로의 출력

그림 12는 구현된 동기회로 IP (Intellectual Property)의 사용자를 위하여 프레임 동기회로의 출력파형을 나타낸 것이다. 가장 위쪽에 표시된 Payload_out은 입력된 GEM_data를 6 클럭 지연한 후에 그대로 출력한 값이며 8 비트의 폭을 가진다. 반면 중간에 표시된 Corrected_header는 40 비트의 폭을 가지며 에러정정을 거친 데이터이다. 여기서 주의할 점은 Corrected_header 값은 처음 입력된 GEM_data와는 폭 뿐만 아니라 내용까지 전혀 다른 출력이 될 수 있다는 것이다. 마지막으로 GEM_header_valid는 올바른 헤더 값 위치에서 high 신호를 출력한다.

이렇게 출력된 신호는 다음 모듈에서 Corrected_header와 GEM_header_valid 두 개의 신호를 이용해서 헤더를 검출하고, Payload_out을 이용하여 유료부하를 검출하여 최종적으로 GEM 프레임을 구성할 수 있다.

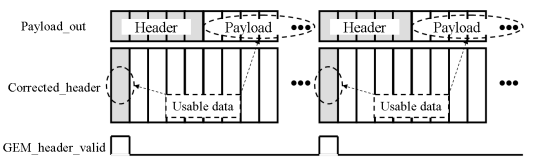

그림 12. GEM 프레임 동기회로의 출력 모형

Fig. 12. Output model of GEM frame synchronization circuit.

Ⅳ. 동기회로 검증

본 논문에서 구현된 GEM 동기모듈은 Xilinx사의 모델명 XC4VFX100의 FPGA칩을 사용하여 Verilog 언어로 구현하였고 칩스코프 (Chipscope)로 검증하였다. 칩스코프는 FPGA 내부의 파형을 PC (Personal Computer) 모니터를 통하여 관찰할 수 있도록 Xilinx사에서 제공하는 툴이다. 실제 구현은 그림 13과 같이 Xilinx ML423 보드를 사용하였다[10]. 이 보드에 있는 FPGA는 MGT (Multi-Gigabit Transceiver)를 내장하고 있고 3 Gbps 이상의 전송속도를 지원한다[11]. 좌측에 주어진 보드는 OLT이고, 우측의 것은 ONU에 해당한다.

그림 14는 HEC 부호기의 칩스코프 결과를 나타낸 것이다. 예시된 부호기의 입력은 5 바이트의 0x528A738000이며, 데이터가 입력되는 5 클럭 동안 Enable 신호가 high로 입력되었다. 부호기는 Enable 신호가 low가 되면 초기화되며 high가 되면 내부 카운터 (Cnt)의 증가와 함께 동작한다. Data_out는 Data_in이 입력되고 2 클럭 이후에 출력되며, 0x528A739F79라는 값을 갖는다. 2 클럭이 지연된 이유는 계산된 12 비트의 값을 입력된 데이터에 실어 8 비트 단위로 출력하기 때문이다. Data_dv는 HEC 결과가 추가된 5 바이트의 Data_out이 존재하는 기간 동안에 high가 되도록 하였다. 그림 14의 부호기는 동작 완료까지 5 클럭의 Enable 신호를 필요로 한다. 따라서 1 바이트의 유료부하를 가지는 최소길이 (6 바이트)의 GEM 프레임에 대해서도 1 클럭의 초기화 할 수 있는 여유를 제공하여 실시간으로 HEC 부호화를 할 수 있는 특징이 있다.

그림 15는 GEM 프레임 동기회로의 입출력을 나타낸 그림이다. 입력 값은 GEM_data와 G-PON 전체 프레임 중에서 GEM 프레임이 존재하는 기간

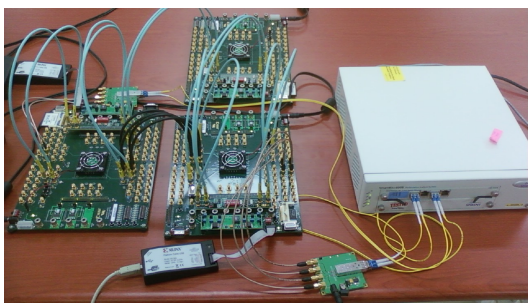

그림 13. ML423 개발 보드
Fig. 13. ML423 development board.

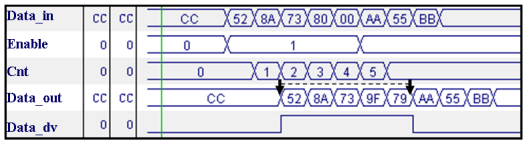

그림 14. HEC 부호기의 칩스코프 결과
Fig. 14. Chipscope view of HEC encoder.

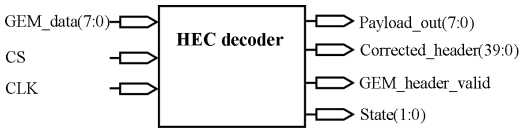

그림 15. GEM 프레임 동기회로의 입력과 출력
Fig. 15. I/O for GEM frame synchronization circuit.

동안 high를 유지하는 CS 신호 그리고 바이트 기준의 CLK이다. 이 세 가지 입력을 받아 출력으로 내보는 값은 Payload_out, Corrected_header, GEM_header_valid 그리고 State가 있으며 그 기능은 앞서 Ⅲ절에서 설명하였다.

그림 16은 주 상태머신이 동기를 잃어버렸다가 다시 동기를 회복하는 과정을 보여주고 있다. 첫 번째 파형은 동기 상태에 있던 동기모듈이 2 개 이상의 에러로 인해 동기 상태에서 추적 상태로 변하는 모습을 보여주고 있다. 주 상태머신은 앞서 입력된 헤더의 PLI값을 이용하여 PLI_cnt = 0x1ffd인 위치

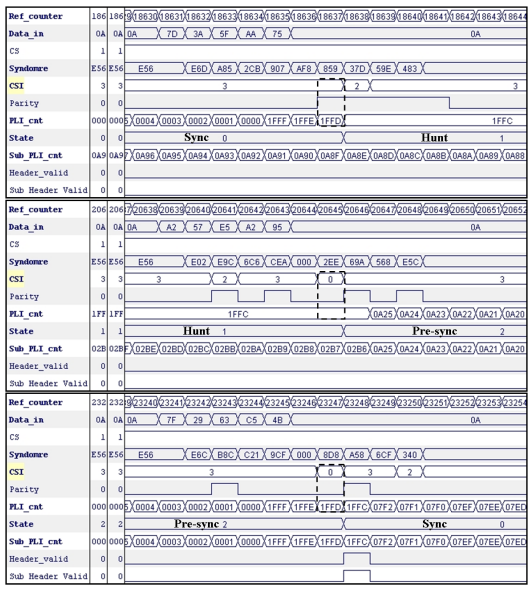

그림 16. 동기손실에서 동기 재획득 과정
Fig. 16. Process of reacquiring synchronization in the loss of synchronization.

에서 다음 헤더의 HEC 결과가 입력될 것이라고 예측하였다. 그리고 해당 HEC 결과에 의해 2 개 이상의 에러가 검출되었고 동기모듈은 동기를 잃고 추적상태가 되었다. 추적상태에 있던 동기모듈은 두 번째 파형에서 에러가 없는 헤더를 발견하여 추적상태에서 준동기 상태로 변하였으며 새로운 헤더의 PLI 값인 0xa25로 PLI_cnt를 Preset하였다. 세 번째 파형은 준동기 상태의 동기모듈이 앞서 0xa25로 Preset 된 이후 PLI_cnt 값을 매 클럭 감산해오다가 PLI_cnt = 0x1ffd 시점에서 올바른 헤더를 의미하는 HEC 계산결과 값을 검출함으로써 잃었던 동기를 재획득하는 모습을 보여준다.

그림 17은 동기 상태에 있는 GEM 동기회로에 정상적인 헤더가 입력되어 연속적인 동기 상태가 된 경우의 칩스코프 결과 파형이다. 입력된 5 바이트 헤더 값은 0xB61925D883이다. 앞서 PLI_cnt = 3에 입력되기 시작한 헤더는 5 클럭이 지난 PLI_cnt = 0x1fff에서 입력이 끝나고 다음 클럭인 PLI_cnt = 0x1ffe에서 신드롬 값이 출력된다. 신드롬 값은 LUT를 거쳐 PLI_cnt = 0x1ffd에서 CSI를 출력하고 이때 패리티 값도 출력된다. 순서도에서 보였던 것처럼 PLI_cnt = 0x1ffd 시점은 CSI와 Parity 값을 비교하여 다음 상태를 판단하는 시점이다. 그림에서는 판단결과 동기 상태를 그대로 유지한 체 PLI_cnt = 0x1ffc 시점으로 넘어가고 따라서 Header_valid는 high가 된다. 또한 입력 받은 5 바이트 헤더에서 상위 12 비트의 값을 새로운 PLI 값으로 인식하여 PLI_cnt는 0x0B61로 Preset된 것을 볼 수 있다.

그림 18은 앞서 동기 상태에 있는 주 상태머신에 PLI 값을 포함한 2 비트 이상의 에러를 가진 헤더가 입력되었지만, 우연히 정정 가능한 헤더로 판별되어 에러를 포함한 PLI 값이 Preset 되었으며, Preset 된 PLI 값이 상당히 큰 값을 가지게 된 경우이다. 이 경우에 주 상태머신은 잘못된 PLI 값이

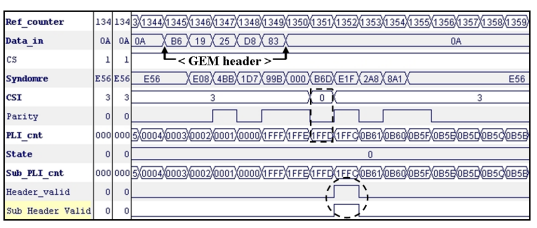

그림 17. 연속적인 동기 상태에서 상태머신의 칩스코프 결과

Fig. 17. Chipscope view of state machine in continuous sync state.

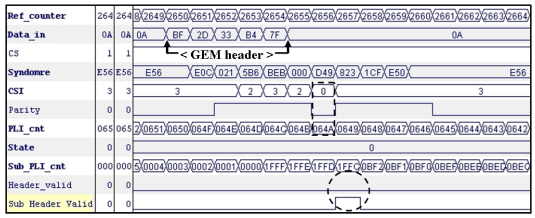

그림 18. 부 상태머신에 의한 추가적인 헤더 검출의 예

Fig. 18. Example of additional header detection by sub-state machine.

모두 감산 되어 0x1ffd가 되기 전까지 수신되는 모든 헤더들의 HEC 결과를 무시하고 상태변화를 시도하지 않을 것이다. 그로 인해 그림 18과 같이 주 상태머신의 출력인 Header_valid 신호가 정상적인 헤더의 위치에서도 high를 출력하지 않는 경우가 발생한다. 주 상태머신이 비록 ITU-T G.984.3에서 명시하는 규격에 합당한 동작을 하더라도 이런 경우에 주 상태머신은 PLI 감산이 끝나기 전까지는 에러를 발견하지 못한다[2].

그러나 부 상태머신에 의해서 출력되는 Sub_header_valid 신호를 보면 두 번째 파형에서 주 상태머신이 놓친 헤더를 추가적으로 검출해내는 것을 볼 수 있다. 부 상태머신은 PLI_cnt와 별개로 HEC 결과에 의해 GEM 헤더를 감지하고 별도의 Sub_PLI_cnt를 이용하여 적절한 위치에 헤더가 위치했는지 판별한다. Sub_PLI_cnt는 HEC 수행결과에 의해 에러가 없다고 판단되는 헤더가 검출될 때마다 검출된 헤더의 PLI 값으로 Preset된다. 결과적으로 그림에서 보이듯이 구현된 부 상태머신은 주 상태머신이 놓쳐버린 GEM 헤더의 위치를 추가적으로 찾아내었고 Sub_header_valid 신호를 이용하여 헤더의 위치를 표시하였다.

지금까지 설명한 바와 같이 본 논문을 통해 제시된 GEM 프레임 동기 회로는 부 상태머신이라는 약간의 추가적인 로직을 소모하여 부 상태머신이 삽입되지 않은 일반적인 프레임 동기회로에 비해 빠른 동기를 획득할 수 있다.

예를 들어, 송신기가 PLI=0x000인 프레임 2개를 연속으로 보냈고, 그 이후 3,583 바이트 시간이 경과한 후에 PLI=0xFFF (4,095) 프레임이 연속으로 2개를 보냈다고 가정하자. 그리고 수신기에서는 최초의 프레임의 PLI에서 검출할 수 없는 에러 (3 비트 이상의 에러)가 발생하여 PLI=0을 PLI=0xE00 (3,584)로 인식하였다고 하자. 이 경우, 주 상태머신만 있는 회로에서는 검출할 수 없는 에러를 지닌

GEM 헤더를 수신한지 3,584 바이트 만에 추적상태가 되고 7,678 바이트 만에 준동기 상태가 되고 11,773 바이트 만에 다시 동기 상태가 된다. 반면, 부 상태머신이 삽입된 동기회로는 위와 같은 경우에 10 바이트 만에 재동기를 잡을 수 있다.

V. 결 론

G-PON에서 GEM 프레임은 가변길이의 데이터를 전달하기 위한 수단이다. 전송 중에 GEM 프레임의 헤더에서 에러가 발생하면 동기를 잃게 되고 이로 인하여 사용자 데이터의 손실을 초래하게 된다. 따라서 사용자 데이터 손실을 최소화하기 위하여 동기회로는 최대한 빨리 동기를 획득할 필요가 있다.

본 논문에서는 프레임 헤더에 에러가 발생하여 동기를 잃었을 경우 최단시간에 동기를 회복할 수 있는 모듈을 제안하였다. 이를 위하여 표준안에서 제시하는 상태머신 이외에 부 상태머신을 가동하여 주 상태머신에서 놓칠 수 있는 헤더를 찾아내어 사용자 데이터의 손실을 최소화하여 링크의 효율을 증진시킬 수 있는 방안을 제시하였다.

구현에 있어서 로직의 효율을 향상시키고 전체적인 구조를 간략화하기 위해 Step 단위의 계산방법을 사용하는 BCH 계산기를 사용하였으며 패리티 계산기 또한 전체 구성에서 이미 사용되고 있는 플립플롭을 재활용하여 6 개의 패리티 계산기 모듈을 하나로 대체하였다.

구현된 회로를 검증하기 위하여 MGT와 천만 게이트를 내장한 FPGA를 사용하였고, 칩스코프를 통하여 동기손실에서 동기 재획득 과정을 보여주었다. 그리고 헤더 에러 발생으로 인하여 주 상태머신에서 간과한 GEM 헤더를 부 상태머신에서 검출하여 사용자 데이터를 복구할 수 있는 과정을 보여주었다.

본 논문에서 구현된 동기회로 IP는 G-PON OLT와 ONU 칩에 적용될 수 있으며, 이를 바탕으로 G-PON 칩의 국산화에 기여하고, 나아가 차세대 액세스 망의 10 기가급 FTTH 칩 개발을 앞당길 수 있을 것으로 판단된다.

참 고 문 헌

〔1〕 정해 외, “ITU-T G.984 기반의 G-PON TC칩 개발,” 정보통신연구진흥원, 최종보고서 2008. 6.

〔2〕 ITU-T Recommendation G984.3, “Gigabit-capable Passive Optical Networks(PON): Transmission Convergence Layer Specification,” March, 2008.

〔3〕 김광옥 외, “GPON 기술 표준 규격 및 개발 동향,” ETRI, 2006. 2. 8.

〔4〕 http://www.neoptek.com/eng/bbs/board.php?bo_table=trends_kor&wr_id=2, 2007. 12. 26.

〔5〕 http://www.dt.co.kr/contents.htm?article_no=2006071302010431693003.

〔6〕 유태환, “IPTV Optical Access Network,” ETRI, 2008. 1. 30.

〔7〕 http://www.ddaily.co.kr/news/news_view.php?uid=34282, 2008. 2. 13.

〔8〕 http://www.ddaily.co.kr/news/news_view.php?uid=34347, 2008. 2. 15.

〔9〕 http://www.eetkorea.com/ART_8800477933_839581_NT_bd400c4f.HTM, 2007. 9. 3.

〔10〕 ML42x User Guide, “Virtex-4FX Rocket IO Characterization Platform,” Xilinx UG087 (V1.2) March 2, 2007.

〔11〕 Virtex-4 User Guide, “Virtex-4 Rocket IO Multi-Gigabit Transceiver,” Xilinx UG076 (V3.2) Sept. 29, 2006.

정 해 (Hae Chung) 종신회원

1987년 2월 한양대학교 전자통신공학과 (학사)
1991년 2월 한국과학기술원 전기 및 전자공학과 (석사)
1996년 2월 한국과학기술원 전기및 전자공학과 (박사)
1995년~1998년 LG정보통신 선임연구원
1998년 8월~현재 금오공대 전자공학부 부교수
2004년 1월~2005년 1월 University of Texas at Dallas 방문교수
<관심분야> FTTH, UBcN, PON, PAN

권 영 진 (Young Jin Kwon) 준회원

2008년 3월 금오공과대학교 전자공학부 졸업
2008년 3월~현재 금오공과대학교 전자통신학과 석사과정
<관심분야> PON, UBcN, PAN