광통신 시스템을 위한 Continuously-Interleaved BCH 기반 순방향오류정정 구조 설계

Continuously-Interleaved BCH-based FEC Architecture Design for Optical Communications

인하대학교 대학원
정보공학과
강한길

2011년 2월
광통신 시스템을 위한 Continuously-Interleaved 
BCH 기반 순방향오류정정 구조 설계

Continuously-Interleaved BCH-based FEC Architecture 
Design for Optical Communications

2011년 2월
지도교수 이한호

이 논문은 공학석사학위 논문으로 제출함

인하대학교 대학원
정보공학과
강한길
이 논문을 강한길의 석사학위논문으로 인정함

2011年 2月

주심____________________

부심____________________

위원____________________
본 논문은 차세대 광통신 시스템을 위한 고성능 enhanced FEC (Forward Error Correction)의 구조를 제안한다. 연속 인터리빙 (Continuously-Interleaving) 기술을 이용한 연접 BCH 구조로서 9.3 dB 이상의 높은 오류 정정 성능을 가진다. 상위 레벨 설계를 통해 기존 RS(255,239) 부호와 ITU-T G.975.1에 제안된 연접 부호들과의 성능을 비교 분석하고 제안하는 연속 인터리빙 방법과 그 과정을 살펴본다. 이 후 연속 인터리빙 연접 BCH (CI-BCH) 부호에 적합한 고속 저변적의 효율 적인 하드웨어 구조를 제안한다. 제안하고자 하는 CI-BCH 부호는 ITU-T G.975.1의 Subclause L9을 따르는 광통신 시스템 구조로 64개의 다중 채널을 가지고 있다. 이러한 다중 채널 구조를 위한 저변적 하드웨어 구조를 제안하고, 높은 전송 데이터 응용을 위해 귀환 (feed-back) 루프를 가지고 있는 KES (Key Equation Solver)블록을 파이 프라인드 멀티플렉서 (Multiplexer:MUX)을 삽입하여 독립적인 두 개의 연산을 변환하여 가면서 진행하는 이중 처리 파이프라인드 SiBM (Dual-processing pipelined SiBM) 구조를 제안한다. 제안된 FEC 구조로서 차세대 광통신시스템을 위한 고성능 초고속 enhanced FEC 구조 구현을 위하여 본 연구결과를 계속 발전시켜 연구를 진행할 수 있을 것이다.
Abstract

This paper presents a high performance and high-speed enhanced FEC (Forward error correction) for next generation optical communication systems. As concatenated BCH architecture using the continuously-interleaving method, proposed FEC architecture has a high error correcting capability. And this paper compared the performance of CI-BCH code with RS(255, 239) code currently used as standard and concatenated BCH code proposed by ITU-T G.975.1.

The decoder of CI-BCH code has 64 multi-channel structure. So, this paper presents appropriate hardware architecture for multi-channel structure. To obtain a high throughput and high-speed architecture, the decoder uses the dual-processing pipelined SiBM (Dual-pSiBM) architecture for the KES block. The proposed architecture uses the pipelined Galois-Field \((GF)\) multiplier in the processing element of KES block and appropriate controller because of this architecture has a feed-back loop. As a result the proposed architecture computes 2 error locator polynomial from 2 syndrome value for \(2^t\) clock cycles.

The proposed architecture can be readily applied to the high performance and high-speed enhanced FEC devices for the next generation optical communication systems.
목차

국문요약 IV
Abstract V
목차 VI
그림 목차 VIII
표 목차 IX

1. 서 론 1

2. CI-BCH 부호 3
   2.1. BCH 부호 3
   2.2. 연속 인터리빙 4
   2.3. CI-BCH 부호화 8
       2.3.1 BCH 부호화 8
       2.3.2 CI-BCH 부호화 9
   2.4. CI-BCH 복호화 12
       2.4.1 신드롬 계산 14
       2.4.2 KES (Key Equation Solver) 15
           2.4.2.1 SiBM 알고리즘 15
           2.4.3 Chien-Search 및 오류의 정정 17

3. CI-BCH 부호의 코덱 시뮬레이터 18

4. 광통신 시스템을 위한 CI-BCH FEC 구조 22
   4.1. CI-BCH 부호기 구조 26
4.2. CI-BCH 복호기 구조 27
  4.2.1. 신드롬 계산 복호 구조 27
  4.2.2. KES 복호 구조 29
  4.2.3. Chien-Search 복호 구조 32
  4.3. 인터리버/디인터리버 복호 구조 34

5. 기능 검증 및 성능 분석 36
  5.1. 기능 검증 36
    5.1.1. CI-BCH 복호기의 기능 검증 37
    5.1.2. CI-BCH 복호기의 기능 검증 38
      5.1.2.1 신드롬 계산 복호의 기능 검증 38
      5.1.2.2 KES 복호의 기능 검증 38
      5.1.2.3 Chien-Search 복호의 기능 검증 39
    5.1.2.4 64채널 BCH(1020,988) 복호기의 기능 검증 39
    5.1.2.5 CI-BCH 복호기의 전체 기능 검증 40
  5.1.3. CI-BCH 복호기의 FPGA 검증 41
  5.2. 성능 분석 43

6. 결 론 46

7. 참고 문헌 47
그림 목차

그림 1. G.709 프레임 구조 4
그림 2. 프레임으로부터 64채널을 구성하는 블록도 5
그림 3. BCH(1020, 988)의 코드워드를 구성하는 블록도 6
그림 4. 연속 인터리빙의 예시 7
그림 5. CI-BCH 부호기의 인터리빙/디인터리빙 과정 11
그림 6. CI-BCH 복호기의 인터리빙/디인터리빙 과정 13
그림 7. CI-BCH 부호의 코덱 시뮬레이터 처리 과정 18
그림 8. CI-BCH 코덱 시뮬레이터 실행 화면 19
그림 9. CI-BCH 코덱 시뮬레이터를 통해 얻은 이미지 20
그림 10. BER 성능 그래프 21
그림 11. 6매의 반복 복호를 하는 CI-BCH 기반 FEC의 전체 블록도 22
그림 12. CI-BCH 부호기 23
그림 13. CI-BCH 복호기 24
그림 14. 64채널 BCH(1020,988) 부호기의 타이밍도 25
그림 15. 64채널 BCH(1020,988) 복호기의 타이밍도 25
그림 16. 4-병렬 BCH(1020,988) 부호기의 블록도 26
그림 17. 신드롬 제산 블록의 블록도 28
그림 18. Dual-pSiBM을 사용한 KES 블록의 블록도 29
그림 19. 처리소자 소개 블록도 30
그림 20. 메인 컨트롤 회로의 블록도 31
그림 21. Dual-pSiBM의 타이밍도 31
그림 22. 4-병렬 Chien-Search 블록의 블록도 33
그림 23. 인터리빙/디인터리빙의 블록도 35
그림 24. 검증 결과 36
그림 25. 단일 채널 부호기의 시뮬레이션도 37
그림 26. 64채널 부호기의 시뮬레이션도 37
그림 27. 신드롬 계산 블록의 시뮬레이션도 38
그림 28. Dual-pSiBM을 사용한 KES 블록의 시뮬레이션도 38
그림 29. Chien-Search 블록의 시뮬레이션도 39
그림 30. 64채널 BCH(1020,988) 복호기의 시뮬레이션도 39
그림 31. 6번 반복 복호 CI-BCH 복호기의 전체 시뮬레이션도 40
그림 32. 복호기의 입력력 데이터 40
그림 33. FPGA 보드 테스트 환경 41
그림 34. Chipscope를 이용한 FPGA 시뮬레이션 결과 42

표 목차

표 1. G.975.1 Recommendation Super FEC 코드 2
표 2. 광통신 시스템용 FEC 등의 구현 결과 및 비교 44
표 3. CI-BCH FEC 구조의 FPGA 구현 결과 45

- IX -
1. 서론

표 1. G.975.1 Recommendation Super FEC 코드
Table 1. G.975.1 Recommendation Super FEC codes

<table>
<thead>
<tr>
<th>G.975.1 Subclause</th>
<th>Outer Code</th>
<th>Inner Code</th>
<th>NCG at 10E-15</th>
<th>Redundancy Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>I.2</td>
<td>RS(255,239)</td>
<td>CSOC</td>
<td>8.88 dB</td>
<td>24.48 %</td>
</tr>
<tr>
<td>I.3</td>
<td>BCH(3860,3824)</td>
<td>BCH(2040,1930)</td>
<td>8.99 dB</td>
<td>6.69 %</td>
</tr>
<tr>
<td>I.4</td>
<td>RS(1023,1007)</td>
<td>BCH(2047,1952)</td>
<td>8.67 dB</td>
<td>6.69 %</td>
</tr>
<tr>
<td>I.5</td>
<td>RS(1901,1855)</td>
<td>Hamming product code</td>
<td>8.50 dB</td>
<td>6.69 %</td>
</tr>
<tr>
<td>I.6</td>
<td>LDPC(32640,30592)</td>
<td></td>
<td>8.02 dB</td>
<td>6.69 %</td>
</tr>
<tr>
<td>I.7</td>
<td>Two orthogonally concatenated BCH super FEC code</td>
<td></td>
<td>8.09 dB</td>
<td>6.69 %</td>
</tr>
<tr>
<td>I.8</td>
<td>RS(2720,2550)</td>
<td></td>
<td>8.00 dB</td>
<td>6.69 %</td>
</tr>
<tr>
<td>I.9</td>
<td>BCH(1020,988)</td>
<td></td>
<td>8.70 dB</td>
<td>6.69 %</td>
</tr>
</tbody>
</table>

를 분산시킨다. 위 과정을 인터리빙 (Interleaving)이라 하며 인터리빙 방법에 따라 오류 정정 용이 달라진다. ITU-T G.975.1 Subclause I.9에는 두 개의 BCH(1020,988)를 사용하여 단순 아웃라인을 통해 데이터를 재배열하여 부호화, 복호화 과정을 진행한다. 10번의 반복 복호화를 통해 10E-15에서 8.70 dB의 NCG (Net Coding Gain)를 가진다. 많은 복호 횟수로 인한 높은 하드웨어 복잡도를 보이고, 그에 비해 비교적 낮은 오류 정정 용을 보이기 때문에 하드웨어 복잡도 개선과 성능을 향상시키는 연구가 필요하다고 할 수 있다.

본 논문에서는 ITU-T G.975.1 Subclause I.9의 인터리빙 방법보다 효율적으로 데이터를 재배열하는 연속 인터리빙 방법을 적용하여 10E-15에서 9.3dB 이상의 고성능의 성능을 가지는 CI-BCH (Continuously-Interleaving BCH)부호를 제시한다. 제안한 부호에 대한 상위레벨 설계 및 기존 부호와 성능 분석을 수행하였고, 광통신 시스템의 전송 데이터 용 증가와 다체널 구조에 따른 하드웨어 복잡도와 전력소모의 증가를 해결하기 위해 Dual-pipelined SiBM (Dual-pSiBM)을 이용한 CI-BCH 부호의 하드웨어 구조를 제안하고 이를 Verilog-HDL을 이용하여 RTL 설계를 하고 상용 CAD툴을 이용하여 합성 및 검증을 수행하였다.
2. CI–BCH 부호

2.1 BCH 부호

본 논문에서 제안하고자 하는 CI–BCH 부호는 BCH부호를 연합하여 사용하는 코드이다. BCH 부호는 블록 부호 (Block code)인 순회 부호 (Cyclic code)의 일종으로 매우 강력한 오류 정정 능력을 갖는 부호이다 [6]. 컴퓨터 기억장치, 광디스크, DAT (Digital Audio Tape) 등과 같은 디지털 정보 저장 시스템은 물론 DVB (Digital Video Broadcasting), 위성통신, 우주통신, 압호통신 및 이동 무선전화와 같은 디지털 통신 시스템에 널리 사용되고 있다 [7].

2진 (binary) BCH 부호의 효과적인 복호알고리즘은 1960년 Peterson에 의하여 처음 제안되었으며 이 알고리즘은 1961년 Gorenstein과 Zierler에 의해서 개선되어 비2진 (nonbinary) BCH 부호에까지 적용될 수 있게 일반화되었다 [8][9]. 또한 BCH부호의 복호알고리즘으로 가장 보편적으로 알려져 있는 벨르캄프–매시 (Berlekamp–Massey) 알고리즘과 유클리드 (Euclid) 알고리즘 등의 반복알고리즘이 있다 [10][11].

BCH는 부호의 길이가 n이고 정보의 길이가 k인 \( (n, k) \) 순회부호의 생성다항식을 \( g(x) = g_0 + g_1 x + g_2 x^2 + \cdots + g_{n-k} x^{n-k} \)이라 할 때, \( d(x) = d_0 + d_1 x + d_2 x^2 + \cdots + d_{n-k} x^{n-k} \)를 부호화될 정보다항식이라 하면 부호다항식 \( c(x) \)는 다음의 식 (1)과 같이 표현된다.

\[
c(x) = d(x)g(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_{n-1} x^{n-1}
\]  

\[ (1) \]

\( \text{BCH}(n, k) \) 순회부호의 부호다항식 \( c(x) \)는 다음의 식 (2)와 같이 \( d(x) \)에 \( x^{n-k} \)를 곱하여 \( g(x) \)로 나누면 구할 수 있다.

\[
x^{n-k}d(x) = q(x)g(x) + p(x)
\]  

\[ (2) \]
여기에서 \( q(x) \)는 꼬다항식이고 나머지 \( p(x) \)는 \( (n - k - 1) \)차 이하의 패러티 (parity) 다항식이다. 그리므로 부호다항식은 다음의 식 (3)과 같이 표현된다.

\[
c(x) = d(x)g(x) = p_0 + p_1x + p_2x^2 + \cdots + p_{n-k-1}x^{n-k-1} + d_0x^{n-k} + d_1x^{n-k+1} + \cdots + d_{k-1}x^{n-1}
\]

이것은 다음의 코드워드 (codeword)에 대응한다.

\[
c = (p_1, p_2, \ldots, p_{n-k-1}, d_0, d_1, \ldots, d_{k-1})
\]

식 (4)로부터 코드워드는 \( n - k \)개의 패러티 비트들과 \( k \)개의 정보 비트들로 구성된다.

2.2 연속 인터리빙

본 논문에서 사용한 고성능의 오류 정정 응용을 갖는 CI-BCH부호의 인터리빙 방법은 연속 인터리빙 방법이며 자세한 내용은 [3]에 설명되어 있다. CI-BCH부호는 ITU-T G.975.1에 제안된 두 개의 서로 다른 코드를 연접하여 사용하는 연접 부호의 128 bits만큼 단위 인터리빙 방법과 달리 1 bit 단위 인터리빙 방법을 통하여 각각의 코드워드를 구성한다. 이러한

![그림 1. G.709 프레임 구조](image_url)

Figure 1. G.709 frame structure
1 bit 단위 인터리빙은 Burst 오류를 최소화 시킬 수 있다. 또한 이 연속 인터리빙은 인터리버를 통하여 코드워드를 구성시 이전의 수신된 프레임 의 데이터도 현재의 복호과정에 참여시키 오류 정정 능력을 높이는 방법 이다.

본 논문에서 사용한 CI-BCH 부호는 G.709 프레임 구조를 따르며 그 구조는 그림 1과 같다 [12]. 2048 bits의 패리티 영역과 30,592 bits의 데이터 영역의 총 32,640 bits를 하나의 프레임으로 구성하며 510 bits 로 구성된 64개의 채널로 나누어 복호를 수행한다. 이 때 Burst 오류 정정 효율을 높이기 위하여 프레임으로부터 510 bits 셋 순차적으로 가져오는 것이 아닌 그림 2와 같이 1 bit 셋 각 채널로 할당하게 된다. 즉 프레임의 최상위 64 bits는 각 채널의 최상위 bit가 되게 된다. 이러한 인터리빙을 통해 프레임에 할당된 2,048 bits의 패리티가 64개의 채널로 나누어져 각 채널 당 32 bits 의 패리티 공간을 갖게 된다. 이로써 각 채널은 32 bits

그림 2. 프레임으로부터 64채널을 구성하는 블록도

Figure 2. Configuration of the 64 channel from a frame
의 해리터 공간이 할당된 510 bits의 공간으로 구성되어진다.

이렇게 구성되어진 64개의 채널은 BCH(510, 478)부호를 통해 부호화와 복호화를 진행하는 것이 아닌 그림 3과 같이 이전에 수신된 8개의 프레임으로부터 510 bits의 데이터를 추가로 가지고 와 BCH(1020, 988)의 코드워드를 구성해 부호화 복호화를 진행한다. 즉 현재 수신된 프레임으로부터 생성된 각 채널의 510 bits를 코드워드의 하단에 위치하고 이전에 수신된 8개의 프레임으로부터 각각 63 bits, 64 bits씩 데이터를 가져와 코드워드 상단 510 bits를 구성하게 된다. 이 때 이전의 각 프레임들로부터 데이터를 가져올 때 63 bits, 64 bits씩 그룹으로 데이터를 가져와 할당하는 것이 아니라 Burst 오류를 분산시키기 위해 아래 식 (5)에 의하여 1 bit 단위로 데이터를 각각의 코드워드에 할당하게 된다.

그림 3. BCH(1020, 988)의 코드워드를 구성하는 블록도

Figure 3. Configuration of the BCH(1020,988) codeword
식 (5)에 따라 인터리빙을 하면 가장 오래전에 수신된 2개의 프레임으로부터 63 bits씩, 나머지 6개의 프레임으로부터 64 bits씩 데이터를 가져와 할당하게 된다. 그림 4를 보면 이러한 인터리빙 과정에 관한 예가 나와 있다. 현재 복호되고 있는 프레임을 7번이라 가정하고 31번째 bit 위치와 95번째 bit 위치를 갖는 'A'와 'B'라고 한다면 그림 2의 과정으로 'A'는 31 번 채널의 코드워드의 첫 번째 bit로 할당된다. 이후 64 bits의 bit 위치 차이를 갖는 'B'는 'A'의 바로 옆에 위치하게 되어 복호화, 복호화 과정을 거치게 된다. 7번 프레임의 복호가 끝난 후 다음으로 수신된 6번 프레임의 복호를 진행하게 되면 'A'는 식 (5)에 의해 6번 프레임의 31 번 채널 코드워드의 상위 510bits의 첫 번째 비트인 512 비트 위치로 할당된다. 마찬가지로 'B'는 식 (5)에 의해 다음으로 수신되는 5번 프레임의 31 번 채널 코드워드의 576 비트 위치로 할당된다.

그림 4. 연속 인터리빙의 예시

Figure 4. Example of the continuously-interleaving method
이러한 과정을 통해 32 bits의 패리티 공간을 가진 1020 bits의 코드워드가 생성된다. 이를 BCH(1020, 988) 부호의 부호화 혹은 복호화 과정을 거친 후 다시 역순으로 디너트리밍 되어 원래의 프레임에 재 할당하게 된다. 복호화 과정에서는 이와 같은 방법을 6번 연속 진행시켜 오류의 개수가 오류 정정 능력을 벗어나 정정하지 못했던 이전 프레임의 오류를 정정하고 이는 다시 다음번 복호과정에 반영되어 이전 프레임의 나머지 오류를 정정한다. 그리고 이터리빙과 디너트리밍을 거치면 각각의 프레임은 한 번의 반복 복호 과정동안 2번의 복호과정을 거치게 된다. 즉, 하위 510 bits로 첫 번째 복호 과정을 거치고 상위 510 bits로 두 번째 복호 과정을 거치게 되는 것이다. 이와 같은 과정을 통해 3층 오류 정정 능력의 작은 BCH 부호라도 높은 오류 정정 능력을 보인다.

2.3 CI–BCH 부호화

2.3.1 BCH 부호화

BCH 부호의 부호화 과정은 [13]에서 기술된 방법에 따른다. 부호의 길이는 \( n = 2^m - 1 \)인 2진 \( t \)층 오류정정 BCH부호의 생성다항식 (generator polynomial) \( g(x) \)는 근 \( \alpha, \alpha^2, \alpha^3, \ldots, \alpha^{2t} \)에 대응하는 최소다항식 (minimum polynomial) \( m_i(x), 1 \leq i \leq 2t \)의 곱으로 이루어진다. 여기서 \( \alpha \in GF(2^m) \)는 원시다항식 (primitive polynomial) \( p(x) \)의 근이다. 2진 기호 \{0, 1\}을 계수로 하는 \( c(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_{n-1} x^{n-1} \)을 부호다항식이라 할 때, 부호다항식 \( c(x) \)가 \( \alpha, \alpha^2, \alpha^3, \ldots, \alpha^{2t} \)을 근으로 갖는다면 \( c(x) \)는 이러한 근들에 대응하는 각각의 최소다항식 \( m_1(x), m_2(x), m_3(x), \ldots, m_{2t}(x) \)으로 나누어질 수 t층 오류정정 BCH 부호의 생성다항식 \( g(x) \)는 이들 최소다항식의 최소공배수 (least common multiple)이어야 한다. 즉 \( g(x) \)는 식 (6) 으로 표현된다.

\[
g(x) = LCM(m_1(x), m_2(x), m_3(x), \ldots, m_{2t}(x))
\] (6)
그러나 \( \alpha \)의 모든 짝수 지수는 홀수 지수와 같은 최소다항식을 갖기 때문에 식 (6)은 다음과 같이 식 (7)로 표현할 수 있다.

\[
g(x) = LCM(m_1(x), m_3(x), m_5(x), \cdots, m_{2t-1}(x))
\]  

(7)

이는 부호다항식 \( c(x) \)가 \( \alpha, \alpha^3, \alpha^5, \cdots, \alpha^{2t-1} \)을 근으로 가질 때 한편에서 부호이가 된다는 것을 의미하며 각 최소다항식의 차수는 \( m \)이하이므로 \( g(x) \)의 차수는 \( mt \)이하가 된다. 따라서 BCH 부호는 \( mt \)이하인 패리티 비트 수, 즉 \( n - k \leq mt \)을 갖는다. 그러하여 생성다항식의 차수는 패리티 비트의 수와 같다.

생성다항식의 차수가 \( mt \)이므로, 다항식의 근은 정확히 \( mt \)개의 연속적인 \( \alpha \)의 기듭제곱이어야 한다. BCH 부호로서 순회부호(cyclic code)의 하나이므로 조직적인(systematic) 형태의 부호화는 부호다항식 \( d(x) \)에 \( x^{n-k} \)를 곱하고 패리티다항식 \( p(x) \)을 더하는 형태가 된다. \( p(x) \)는 \( d(x)x^{n-k} \)를 \( g(x) \)로 나눈 나머지이다.

\[
x^{n-k}d(x) = q(x)g(x) + p(x)
\]

(8)

\[
p(x) = x^{n-k}d(x) \mod g(x)
\]

(9)

부호화된 다항식 부호다항식 \( c(x) \)는 다음과 같다.

\[
c(x) = x^{n-k}d(x) + p(x)
\]

(10)

2.3.2 CI–BCH 부호화

CI–BCH의 부호화를 위해선 먼저 그림 3의 연속 인터리빙을 통해 각각의 프레임은 64개의 3중 오류 정정의 BCH(1020, 988)의 코드워드를 구성하게 된다. 각각의 코드워드는 64개의 부호화기를 통하여 2048 bits의 패리티 데이터를 생성하게 된다.

BCH(1020, 988) 부호는 1024개의 원소를 갖는 \( GF(2^{10}) \) 상에서 연산이 이루어지며 원시다항식 \( p(x) = x^{10} + x^3 + 1 \)로부터 아래와 같은 최소다항식
과 생성다항식을 이루게 된다.

\[ m_j(x) = \prod_{j=1}^{10} (x - \alpha^{3j}) \]  

(11)

\[ g(x) = (x^{10} + x^3 + 1)(x^{10} + x^3 + x^2 + x + 1)(x^{10} + x^8 + x^3 + x^2 + x + 1)(x^2 + 1) \]  

(12)

\[ g(x) = x^{32} + x^{28} + x^{25} + x^{19} + x^{18} + x^{16} + x^{14} + x^{12} + x^{10} + x^8 + x^6 + x^4 + x^3 + x^2 + x + 1 \]  

(13)

32차의 생성다항식 \( g(x) \)는 식 (13)으로 식 (9)의 연산을 통해 생성된 \( \text{BCH}(1020, 988) \) 부호의 폐리티를 디인터리빙 과정을 거쳐 프레임에 재할당하게 된다. 디인터리빙 과정은 인터리빙 과정의 역과정으로 각 프레임으로부터 인터리빙을 통해 64개의 코드워드를 구성하였다면 이 64개의 코드워드를 다시 프레임으로 입력하는 일련의 과정을 말한다.

부호화 과정의 경우 988 bits의 정보 데이터 영역은 변하지 않는 값이므로 그림 5와 같이 폐리티 영역 32 bits만 디인터리빙 과정을 통해 원래의 프레임에 삽입된다.
그림 5. 부호기의 인터리빙/디인터리빙 과정

Figure 5. Interleaving and deinterleaving procedure of the encoder
2.4 CI-BCH 복호화

부호화를 통해 얻은 부호어 다항식은 전송하는 동안 잡음에 의해 오염되어 비트단위의 오류로 수신되었다고 가정하면 오류 패턴은 다음과의 식 (14)와 같이 다항식으로 나타낼 수 있다.

\[ e(x) = \sum_{i=0}^{n-1} e_i x^i \]  

(14)

그리면 오류와 함께 수신된 수신 부호어 다항식 \( r(x) \)는 다음과 같이 수신 부호어 다항식과 오류 다항식의 합으로 나타낼 수 있다.

\[ r(x) = e(x) + e(x) \]  

(15)

CI-BCH 부호의 복호는 채널을 통과해 수신된 프레임 데이터 \( r(x) \)를 인터리빙 과정을 통하여 64개의 BCH(1020, 988) 코드워드를 구성해 복호를 한다. 복호화 과정이 끝난 코드워드는 부호화 과정과는 달리 코드워드 내의 오류가 결정된 것을 각각의 프레임에 반영하기 위하여 그림 6과 같이 코드워드의 모든 영역 1020 bits를 인터리빙 과정을 통하여 각각의 프레임에 재 할당한다.

복호화 과정은 먼저 신드롬 계산 (Syndrome Computation) 블록으로부터 신드롬 다항식 (Syndrome Polynomial) \( S(x) \)을 계산한다. 이 후 KES (Key Equation Solver) 블록으로부터 오류 위치 다항식 (Error Locator Polynomial) \( \sigma(x) \)를 찾아낸다. 마지막으로 오류 위치 다항식 \( \sigma(x) \)을 이용하여 Chien-Search 블록에서 해당 위치의 오류를 결정하여 최종적으로 결정된 코드워드를 출력한다.
Figure 6. Interleaving and deinterleaving procedure of the decoder
2.4.1 신드롬 계산

신드롬(syndrome)은 수신된 \( r(x) \)가 오류가 있는 코드워드인지 결정하기 위해 \( r(x) \)에 대해 수행한 페리티 검사의 결과이다. 만약 \( r(x) \)가 오류가 없다면 신드롬 다항식 \( S(x) \)는 ‘0’이 된다. 수행한 \( S(x) \)가 ‘0’이 아니라면 오류가 있다는 것을 나타내며 신드롬 다항식 \( S(x) \)는 \( 2t \)개의 심별 \( S_i \) \((i = 0, 1, 2, \ldots, 2t-1) \)로 이루어진다. 식 (8)-(10)에 의해 오류가 없는 무호 다항식 \( c(x) \)는 생성다항식 \( g(x) \)의 배수임을 알 수 있다. 그러므로 \( g(x) \)의 근은 반드시 \( c(x) \)의 근도 된다.

식 (15)와 같이 \( r(x) = c(x) + e(x) \)이므로 \( r(x) \)에 \( g(x) \)의 각 근을 대입하면, 오류가 없는 코드워드인 경우에만 ‘0’이 된다. 그러나 어떤 오류에 대해서는 계산상 ‘0’이 아닌 결과가 나온다. 신드롬 심별의 계산은 다음의 식 (16)과 같이 표현할 수 있다.

\[
S_i = r(x)|_{x = \alpha^i} = r(\alpha^i) \quad i = 0, 1, 2, \ldots, 2t - 1
\]  (16)

신드롬 다항식 \( S(x) \)는 식 (17)과 같이 표현할 수 있다.

\[
S(x) = S_0 + S_1x + S_2x^2 + S_3x^3 + \cdots + S_{2t-1}x^{2t-1} = \sum_{i=0}^{2t-1} S_i x^i
\]  (17)
2.4.2 Key Equation Solver

KES 블록은 앞의 신드롬 계산 블록으로부터 신드롬 다항식 값을 받아 오류 위치 다항식 \( \sigma(x) \)을 구하는 블록이다. 이를 구하기 위해서 많은 연산량을 필요로 하고 이 때문에 하드웨어 복잡도가 신드롬 계산 블록이나 Chien-Search 블록에 비해 월등히 높고 임계경로지연(Critical Path)을 가지는 블록으로 복호기에 있어서 가장 핵심 블록이라 할 수 있다.


2.4.2.1. SiBM 알고리즘

벨크램프-메시 알고리즘의 하드웨어 복잡도가 낮은 특성으로 인해 많은 연구가 진행되어 RiBM (Reformulated inversionless Berlekamp-Massey) [12]이 제안되었고 이는 기존 벨크램프-메시 알고리즘의 역변환 (Inversion) 연산을 제거하여 연산의 복잡도를 낮추고 정규적인 구조를 갖게 하여 VLSI (Very Large Scale Integrated Circuit) 회로의 레이아웃 (Layout)이 쉬워지는 이점을 가지고 있다.

RiBM 알고리즘은 Galois-Field 상에서 반복연산을 통해 근을 찾아나가는 알고리즘으로 각 반복연산은 뉴턴 (Newton)의 항등식을 만족하는 계수를 갖는 다항식의 최소해와 불합도 (Discrepancy) \( \delta \) 를 식 (18)과 식 (19)을 통해 오류 위치 다항식 \( \sigma(x) \)을 구한다.

\[
\begin{bmatrix}
A^{(r)}(x) \\
B^{(r)}(x)
\end{bmatrix} = \begin{bmatrix}
1 & -\Delta_r x \\
\Delta_r^{-1} \delta_r & \delta_r
\end{bmatrix} \begin{bmatrix}
A^{(r-1)}(x) \\
B^{(r-1)}(x)
\end{bmatrix} \\
\]

\( L_r = \delta_r (r+1-L_{r-1}) + \bar{\delta}_r L_{r-1} \) (19)

이러한 RiBM 알고리즘을 BCH 부호에 바로 적용할 경우 2\( t \) 번의 반복 연산을 통해 오류 위치 다항식을 구할 수 있지만 BCH 부호의
$GF(2^m)$ 상에서는 짝수번째 반복 연산을 수행하는 동안에는 분할도 $\delta$ 가 항상 0을 가진다. 이와 같은 점을 이용하여 SiBM (Simplified inversionless Berlekamp–Massey 알고리즘)은 Look–ahead 기법을 사용하여 $2t$ 번의 반복 연산을 $t$ 번의 연산량으로 줄였다. 또한 BCH 부호는 RS 부호와 달리 오류 값 다항식 (Error Value Polynomial) $\Omega(x)$은 필요로 하지 않기 때문에 오류 값 다항식 $\Omega(x)$을 표현하는 다항식 출력 부분을 삭제하였다. 이는 다음과 같은 의사코드 형태의 알고리즘으로 정리할 수 있다.

Initialization:
\[ \delta_0(0) = 1, \delta_{i}(0) = 0, \delta_{2t-1}(0) = 0; \kappa(0) = 0, \gamma(0) = 1. \]
Input: $S_n$ ($i = 0, 1, 2, ... , 2t-2$)
\[ \delta_i(0) \neq \delta_i(0) = S_n, (i = 0, 1, ..., 2t-2) \]
for $r = 0$ step 1 until $t-1$ do
begin
step SiBM.1 $\delta_i(r+1) = \gamma(r) \cdot \delta_{i+2}(r) - \delta_0 \cdot \delta_{i+1}(r), (i = 0, 1, 2, ..., 2t)$ \[ \delta_{2t+i}(r) = \delta_{2t+i+1}(r) = 0, \delta_{2t+i+2}(r) = 0 \]
step SiBM.2 if $\delta_i(0) \neq 0$ and $\kappa(r) \geq 0$
then
begin
$\delta_i(r+1) = \delta_i(r), (0 \leq i \leq 2t, i \neq 2t - 2 - k, k = 0, 1)$
$\gamma(r+1) = \delta_i(r)$
$k(r+1) = -k(r)$
end
else
begin
$\delta_i(r+1) = \delta_i(r), (0 \leq i \leq 2t, i \neq 2t - 2 - k, k = 0, 1)$
$\gamma(r+1) = \gamma(r)$
$k(r+1) = k(r) + 2$
end
end
Output: $\delta_i(2t) = \delta_i(2t), (i = 0, 1, ..., t)$
이로써 기존의 RiBM에서는 $3t+1$ 개의 GF 덧셈기, $6t+2$ 개의 GF 곱셈기, $6t+2$ 개의 레지스터, $3t+1$ 개의 멀티플렉서를 가졌지만 이를 $2t$ 개의 GF 덧셈기, $4t$ 개의 GF 곱셈기, $2t+1$ 개의 레지스터, $2t$ 개의 멀티플렉서로 하드웨어를 간략화하여 하드웨어 복잡도와 전력소모를 줄이게 된다.

2.4.3. Chien Search 및 오류의 정정

KES 블록에 의해 오류위치항성 $\sigma(x)$을 구한 후, 다항식의 근은 오류의 위치를 나타내는 $x$의 다항식이며 이는 다음의 식 (20)과 같은 형태가 된다.

$$\sigma(x) = \sigma_0 + \sigma_1 x + \sigma_2 x^2 + \ldots + \sigma_{t-1} x^{t-1} + \sigma_t x^t$$  \hspace{1cm} (20)

식 (20)의 오류 위치 다항식 $\sigma(x)$의 근은 부호어의 오류의 위치에 해당하는 값이다. 따라서 오류 위치 다항식 $\sigma(x)$의 근을 찾는 방법 중 가장 적절한 방법은 Chien Search[16]로 밑의 식 (21)과 같이 $n$개의 심벌 원소 $\alpha^i$를 대입하여 $\sigma(\alpha^i) = 0$임을 확인함으로써 가능하다.

$$\sigma(\alpha^0) = \sigma_0 + \sigma_1 + \sigma_2 + \ldots + \sigma_{t-1} + \sigma_t$$
$$\sigma(\alpha^1) = \sigma_0 + \sigma_1 \alpha + \sigma_2 \alpha^2 + \ldots + \sigma_{t-1} \alpha^{t-1} + \sigma_t \alpha^t$$
$$\sigma(\alpha^2) = \sigma_0 + \sigma_1 \alpha^2 + \sigma_2 \alpha^4 + \ldots + \sigma_{t-1} \alpha^{2(t-1)} + \sigma_t \alpha^{2t}$$
$$\ldots$$

$$\sigma(\alpha^{n-1}) = \sigma_0 + \sigma_1 \alpha^{(n-1)} + \sigma_2 \alpha^{2(n-1)} + \ldots + \sigma_{t-1} \alpha^{(t-1)(n-1)} + \sigma_t \alpha^{(n-1)}$$
3. CI-BCH 부호의 코덱 시뮬레이터

CI-BCH 부호의 오류 정정 성능을 명확히 확인하고자 CI-BCH 부호를 사용하여 이미지 (image)를 이용한 코덱 시뮬레이터를 C 언어로 설계하였다. 이미지를 모델링 된 AWGN (Addictive White Gaussian Noise) 채널에 통과시켜 오류가 생긴 이미지를 확인하고 복호기를 통해 정정된 이미지를 출력하는 방식으로 설계하였다. 코덱 시뮬레이터의 처리 절차는 그림 7과 같다. 이미지의 오류를 직접 확인하기 위해 jpeg, png와 같은 압축 (compress) 이미지 형식이 아닌 픽셀 (pixel) 단위 정보로 이미지를 구성하는 Bitmap 이미지를 이용하였다.

그림 8은 코덱 시뮬레이터의 실행 화면이다. 첫 번째 단계에서 이미지를 선택하여 이미지의 사이즈를 파악한 후 프레임 구조에 맞게 랜덤 데이터를 재배열하고 부호화 과정을 거쳐 패리티를 생성한다. 두 번째 단계에서는 AWGN 채널의 SNR (Signal To Noise Ratio) 값을 입력한다. 입력된 SNR 값에 의해 오류의 양이 결정되며 SNR 값이 작을수록 채널상의 오류는 많아진다. 채널을 통과하여 오류가 삽입된 이미지를 출력한 후, 오류 이미지를 CI-BCH 복호기를 통해 정정하게 된다.

그림 9는 코덱 시뮬레이터를 통해 얻은 그림으로 SNR 5.3dB에서 AWGN 채널을 통과시켜 얻은 오류 이미지와 복호화 과정을 거쳐 정정된 이미지를 보여준다. AWGN 채널의 SNR 5.3 dB에서 BER은 4.6E-3를 가지며, 그림 9 (a)의 오류 이미지는 2,893 개의 오류가 생성된 이미지이다.

그림 7. CI-BCH 부호의 코덱 시뮬레이터 처리 과정
Figure 7. Codec simulator procedure of the CI-BCH code
그림 8. CI-BCH 코덱 시뮬레이터 실행 화면
Figure 8. Executive example of the CI-BCH codec simulator

그림 9 (b)는 CI-BCH의 복호화 과정을 통해 그림 9 (a)의 오류 이미지의 오류를 정정한 이미지이다.

CI-BCH 부호의 정확한 오류 정정 능력을 측정하기 위해 CI-BCH 코덱 시뮬레이터를 이용하여 그림 10의 BER 성능 그래프를 작성하였다. 본 논문에서 제안하는 CI-BCH 부호와 기존의 ITU-T G.975.1의 연결 BCH 부호 및 RS(255,239) 부호의 BER 성능을 비교 분석하였다. BER 성능 그래프를 통해 확인할 수 있도록 CI-BCH 부호는 10E-12에서 8.27 dB, 10E-15에서 9.32 dB의 성능을 보여준다. RS(255,239) 부호는 10E-12에서 5.6 dB, 10E-15에서 6.2 dB를 가진다는 점을 볼 때 3 dB 이상의 NCG 성능 향상을 보인다는 것을 확인할 수 있다. 뿐만 아니라 ITU-T G.975.1의 연결 BCH 부호에 비해서도 10E-15에서 0.33 dB의 NCG 향상을 보여 차세대 광통신 표준에 부합하는 성능을 보여준다.
그림 9. CI-BCH 코덱 시뮬레이터를 통해 얻은 (a) 오류 이미지, (b) 정정된 이미지

Figure 9. Output of the CI-BCH codec simulator (a) error image, and (b) corrected image
그림 10. BER 성능 그래프 (a) Eb/N0 0부터 15까지, (b) Eb/N0 4.5부터 9까지

Figure 10. BER performance graph (a) Eb/N0 0 to 15, (b) Eb/N0 4.5 to 9
4. CI-BCH 기반 FEC 구조

광통신 시스템을 위한 CI-BCH 기반 FEC 구조의 전체 블록도는 그림 11과 같다. CI-BCH 부호기는 64개의 부호기와 32640 bits × 11, 즉 11개의 프레임을 저장할 수 있는 메모리를 가진다. 부호화 과정에 필요한 9개의 메모리와 데이터의 입력과 출력을 처리하는 2개의 메모리로 구성된다. 이 메모리에 이전에 수신한 프레임 데이터를 저장한 후 인터리버를 통해 64개의 코드워드를 형성해 부호화를 진행한 후 디인터리버를 통해 메모리에 다시 저장하게 된다. 프레임의 패리티 데이터 영역에 패리티를 추가한 후 채널로 전송되어 복호 과정을 거쳐게 된다.

그림 12는 CI-BCH 부호기의 블록도이다. 인터리버를 통해 메모리의 데이터를 매 클록마다 256 bits씩 4별리 구조의 64개의 부호기로 나뉘어 입력되어 부호화 과정을 거친 후 디인터리버를 통해 메모리로 입력된다. 그림

그림 11. 6번의 반복 복호를 하는 CI-BCH 기반 FEC의 전체 블록도

Figure 11. Block diagram of the 6 iteration CI-BCH based FEC
14는 64채널 BCH(1020,988) 부호기의 타이밍도를 보여준다. 2클록 후부터 247클록동안 정보 데이터를 출력한 후 8클록동안 패리티 데이터를 출력한다. 출력된 데이터는 메모리에 저장되어 진다.

CI-BCH 부호기는 총 6개의 복호기와 32640 bits × 56의 56개의 프레임을 저장할 수 있는 메모리로 이루어져 6번의 반복 복호 과정을 거치게 된다. 복호기의 경우 부호기와 마찬가지로 코드워드를 구성할지 9개의 메모리가 필요하지만 6개의 복호기로 이루어져 있기 때문에 54개의 메모리를 필요로 한다. 그리고 입력과 출력을 처리하는 2개의 메모리로 구성되어 총 56개의 메모리로 구성된다. 부호기와 마찬가지로 인터리버와 디интер리버를 통해 각 채널의 코드워드를 구성한 후 복호하고 메모리에 저장한다.

그림 13은 CI-BCH 복호기의 블록도이다. 부호기와 마찬가지로 각각의 복호기는 4-병렬의 구조를 가지고 있다. 64개의 신드롬 계산블록과, 1개의

그림 12. CI-BCH 부호기

Figure 12. CI-BCH encoder
KES 블록, 64개의 Chien-Search 블록으로 구성되어 있다.

KES 블록은 Dual-pSiBM 구조를 사용한 Time-Multiplexing 방법을 사용하여 한 개의 KES 블록으로 64개의 신드롬 계산 블록과 64개의 Chien-Search 블록을 공유한다. 이러한 방법이 가능한 이유는 신드롬 계산 블록의 경우 255 클록 동안 순차적으로 연산하여 결과를 출력하는데 그 신드롬 계산의 결과는 마지막 255 클록에서 한 번에 나온다. 그리고 Dual-pSiBM을 적용한 KES 블록은 신드롬 블록이나 Chien-Search 블록과 달리 필요한 클록의 수는 $2^t$ 클록으로 단지 6 클록이 소요된다. 즉 Dual-pSiBM을 사용한 3중 오류 정정 BCH(1020, 988) 부호의 경우 6 클록에 2 개의 신드롬 값을 연산하여 오류 위치 다항식을 출력하므로 하나의 신

그림 13. CI-BCH 복호기

Figure 13. CI-BCH decoder
드롬을 계산하는데 필요한 클록의 수는 3 클록으로 64채널의 신드롬 계산 블록에서 생성되는 64개의 신드롬을 처리하는데 192 클록이 소요하게 된다. 이는 신드롬 계산 블록에서 하나의 신드롬이 계산되는 255 클록보다 작으므로 64채널의 신드롬 블록에서 단 하나의 KES 블록을 공유할 수 있다.

그림 13의 신드롬 계산 블록과 KES 블록사이의 Time-Multiplexer는 64개의 신드롬 값을 순차적으로 KES 블록으로 입력해주기 역할을 한다. 이에 대한 타이밍도는 그림 15에 나와 있다.

그림 14. 64채널 BCH(1020,988) 부호기의 타이밍도

Figure 14. Timing diagram of the 64Channel BCH(1020,988) encoder

그림 15. 64채널 BCH(1020,988) 복호기의 타이밍도

Figure 15. Timing diagram of the 64Channel BCH(1020,988) decoder
4.1 CI-BCH 부호기 구조

부호기는 식 (13)에서 정의된 생성다항식 \( g(x) \)을 이용하여 수식 (2)를 연산하여 BCH 정보다항식을 부호화 한다. 광통신 시스템의 경우 한 클록에 128 bits씩 데이터를 전송하고 한 프레임을 구성하는 bits의 수는 32640 bits 이므로 255 클록이 소요된다. 그러므로 부호기는 255 클록 내에 페리티를 생성해 전송해야 한다. 그러므로 CI-BCH 부호에서 사용하는 BCH(1020, 988) 부호기의 경우 1020 bits를 255 클록내에 연산을 해야 하므로 [17]과 [20]의 병렬 구조를 응용한 그림 16의 4-병렬 리턴단지 생성 상태 머신 (Parallel Redundancy Generation State Machine)을 사용한다.

각각의 Combinational Network 에는 생성다항식 \( g(x) \)로 나누는 연산을 하게 되고 이를 4-병렬로 처리하여 247 클록의 반복 연산 후 출력 값을 페리티 데이터 레지스터에 저장 후 8 클록동안 레지스터 컨트롤러에 의해 4 bits씩 출력한다.

그림 16. 4-병렬 BCH(1020,988) 부호기의 블록도
Figure 16. Block diagram of 4-parallel BCH(1020,988) encoder
4.2 CI-BCH 복호기 구조

4.2.1 신드롬 계산 블록

그림 17은 BCH(1020, 988)의 신드롬 계산 블록을 나타낸 블록도이다. 한 개의 프레임이 수신되는 255 클록동안 하나의 신드롬 다항식 \( S(x) \)를 계산해야하므로 무호기와 마찬가지로 1020 bits를 255 클록 내에 생성하기 위해 4-별렬 구조를 가져야 한다. 신드롬 다항식은 수신 다항식 \( r(x) \)에 생성다항식 \( g(x) \)의 근 \( \alpha^i \)를 대입함으로서 구할 수 있다고 앞서 밝혔다. 이를 다음의 식 (22)와 같이 표현할 수 있다. 이는 다시 다음의 식 (23)과 같이 표현할 수 있다.

\[
S(\alpha^i) = r_{1019}\alpha^{1019i} + r_{1018}\alpha^{1018i} + r_{1017}\alpha^{1017i} + \cdots + r_2\alpha^2 + r_1\alpha + r_0
\]  
(22)

\[
S(\alpha^i) = (r_{1019}\alpha^3 + r_{1018}\alpha^2 + r_{1017}\alpha + r_{1016}) \times \alpha^{4 \times 254i} + (r_{1015}\alpha^3 + r_{1014}\alpha^2 + r_{1013}\alpha + r_{1012}) \times \alpha^{4 \times 253i} \\
\quad \cdots + (r_2\alpha^3 + r_1\alpha^2 + r_0\alpha + r_1) \times \alpha^{4 \times 1i} + r_3\alpha^3 + r_2\alpha^2 + r_1\alpha + r_0 \times \alpha^{4 \times 0i}
\]  
(23)

위의 식 (23)를 통해 보면 4개의 요소를 묶어 \( \alpha^i \) \((i = 0, 1, 2, \ldots, 254) \)로 표현이 가능함을 볼 수 있다. 즉 수신 다항식 \( r(x) \)의 계수에 \( \alpha^3, \alpha^2, \alpha, \alpha^0 \)를 각각 곱해준 후 \( \alpha^4 \)의 거듭제곱으로 설계가 가능함을 알 수 있다.

그림 17 (b)의 Bit2Sym 블록은 수신 다항식 \( r(x) \)의 계수에 \( \alpha^3, \alpha^2, \alpha, \alpha^0 \)를 곱해주는 역할을 한다. 이 후 레지스터와 멀티플렉서, 그리고 \( \alpha^4 \)을 곱해주는 \( GF \) 곱셈기를 255 클록동안 반복 시행 신드롬 다항식 \( S(x) \)를 얻게 된다.
그림 17. 신드롬 계산 블록의 블록도 (a) 4-병렬 신드롬 계산 블록의 블록도, (b) Bit2Sym 블록의 블록도
Figure 17. Block diagram of Syndrome computation block (a) 4 parallel Syndrome computation block, and (b) Bit2Sym block
4.2.2 KES 블록 구조

CI-BCH 복호기는 64채널의 구조로 이루어져 있어서 하드웨어 복잡도는 중요한 문제가 된다. 이러한 하드웨어 복잡도 문제를 해결하기 위해 본 논문에서 새롭게 제안하는 이중처리 파이프라인드 SiBM (Dual-processing pipelined SiBM: Dual-pSiBM)을 사용하였으며 Time-Multiplexing 기법을 사용하여 1개의 KES 블록이 64개의 신호를 블록으로부터 계산된 신호를 다항식을 처리한다. Dual-pSiBM 구조는 기존 SiBM의 임계경로지연 (Critical Path)이 긴 문제점을 해결하면서 연산에 필요한 클록의 수를 유지하는 구조이다. 기존 SiBM의 경우 임계경로 지연이 $T_{gmult} + T_{glade}$로 높은 전송 데이타 처리율을 필요로 하는 고속의 광통신 시스템에는 적합하지 못하다. 임계경로지연을 해결하기 위해 파이 프라인드 (Pipelined) GF 구현기를 사용한 구조를 생각해 볼 수 있다. 그러나 SiBM 구조는 귀환 (Feed-back)구조를 가지고 있어 파이프라인드

![Diagram](image)

그림 18. Dual-pSiBM을 사용한 KES 블록의 블록도

Figure 18. Block diagram of the KES block using Dual-pSiBM

- 29 -
$GF$ 꿃셈기를 사용할 경우 연산에 필요한 클록의 수가 기존 $t$ 클록에서 $2t$ 클록으로 늘어나게 된다. 이는 다중 채널 구조에서 Time-Multiplexing 기법을 사용하는 KES 클록에 부적합하다.

Dual-$p$SiBM 구조는 KES 클록 내부의 처리소자 (Processing Element: PE)에 멀티플렉서를 추가하고 이를 제어할 컨트롤 유닛 (Control Unit)을 추가하여 신드롬 2개를 2 클록 연속으로 입력하여 연산할 수 있게 설계하였다. 그림 19와 20은 처리소자와 컨트롤 유닛의 블록도이다. 이로써 각 신드롬은 한 번씩 변가어가며 독립적으로 각각 $t$ 클록동안 연산하게 된다. 즉 $2t$ 클록의 반복 연산 동안 2개의 신드롬을 처리하므로 신드롬 하나당 필요한 클록의 수는 기존 SiBM구조와 동일한 $t$ 클록을 유지한 채 동작 속도를 고속화 하였다. 그림 21의 Dual-$p$SiBM의 타이밍도를 보면 2개의 신드롬이 연산되는 과정을 보여준다.

그림 19. 처리소자 (PE)의 블록도

Figure 19. Block diagram of Processing Element (PE)
그림 20. 메인 컨트롤 회로의 블록도

Figure 20. Block diagram of the main control circuit

그림 21. Dual–pSiBM의 타이밍도

Figure 21. Timing diagram of Dual–pSiBM
4.2.3 Chien–Search 블록 구조

KES 블록으로부터 오류 위치 다항식 \( \sigma(x) \)를 구한다. 이 오류 위치 다항식 \( \sigma(x) \)의 근은 오류의 위치를 나타낸다. Chien–Search 블록에서는 \( GF(2^{10}) \)에 속하는 \( n \)개의 심볼 원소 \( \alpha^i \) \( (i = 0, 1, 2, 3, \ldots, n - 1) \)을 대입하여 \( \sigma(\alpha^i) = 0 \)임을 확인함으로써 가능하다. 식 (21)을 \( 3 \)중 오류 정정 BCH(1020, 988) 부호에 적용하면 다음과 같은 식 (24)를 얻을 수 있다.

\[
\begin{align*}
\sigma(\alpha^0) &= \sigma_0 + \sigma_1 + \sigma_2 + \sigma_3 \\
\sigma(\alpha^1) &= \sigma_0 + \sigma_1 \alpha + \sigma_2 \alpha^2 + \sigma_3 \alpha^3 \\
\sigma(\alpha^2) &= \sigma_0 + \sigma_1 \alpha^2 + \sigma_2 \alpha^4 + \sigma_3 \alpha^6 \\
\sigma(\alpha^{1017}) &= \sigma_0 + \sigma_1 \alpha^{1017} + \sigma_2 \alpha^{2034} + \sigma_3 \alpha^{3051} \\
\sigma(\alpha^{1018}) &= \sigma_0 + \sigma_1 \alpha^{1018} + \sigma_2 \alpha^{2036} + \sigma_3 \alpha^{3054} \\
\sigma(\alpha^{1019}) &= \sigma_0 + \sigma_1 \alpha^{1019} + \sigma_2 \alpha^{2038} + \sigma_3 \alpha^{3057}
\end{align*}
\]

이를 다시 다음의 식 (25)와 같이 표현할 수 있다.

\[
\begin{align*}
\sigma(\alpha^0) &= \sigma_0 + \sigma_1 + \sigma_2 + \sigma_3 \\
\sigma(\alpha^1) &= \sigma_0 + \sigma_1 \alpha + \sigma_2 \alpha^2 + \sigma_3 \alpha^3 \\
\sigma(\alpha^2) &= \sigma_0 + \sigma_1 \alpha^2 + \sigma_2 (\alpha^2)^2 + \sigma_3 (\alpha^3)^2 \\
\sigma(\alpha^{1017}) &= \sigma_0 + \sigma_1 \alpha^{1017} + \sigma_2 (\alpha^2)^{1017} + \sigma_3 (\alpha^3)^{1017} \\
\sigma(\alpha^{1018}) &= \sigma_0 + \sigma_1 \alpha^{1018} + \sigma_2 (\alpha^2)^{1018} + \sigma_3 (\alpha^3)^{1018} \\
\sigma(\alpha^{1019}) &= \sigma_0 + \sigma_1 \alpha^{1019} + \sigma_2 (\alpha^2)^{1019} + \sigma_3 (\alpha^3)^{1019}
\end{align*}
\]

위의 식 (25)의 \( \sigma(x) \)의 계수 \( \sigma_0, \sigma_1, \sigma_2, \sigma_3 \) 별로 연산하고 이를 논문[18]에 서 제안한 하드웨어 구조 설계방법을 응용하여 4-병렬 구조로 하드웨어를 설계하면 그림 22와 같다.
그림 22. 4-병렬 Chien-Search 블록의 블록도

Figure 22. Block diagram of the 4-parallel Chien-Search block
4.3 인터리버/디인터리버 구조

CI-BCH 부호는 입력된 데이터를 순차적으로 메모리에 저장한 후 메모리에 저장된 데이터를 인터리버를 통하여 코드워드를 구성하며 디인터리버를 통해 부호기와 복호기를 통과한 나온 데이터를 다시 메모리로 재배열한다. 인터리버/디인터리버는 데이터를 저장하는 메모리와 인터리버/디인터리버 컨트롤러로 구성되며 그 구조는 그림 23과 같다. 각 프레임 메모리는 64x64 메모리 6개와 64x63 메모리 2개로 구성되어 한 개의 프레임 데이터를 저장한다. 이와 같이 프레임 데이터를 8개의 서브메모리(Sub-memory)로 구성하여 각 메모리의 데이터를 다음 복호화 과정의 코드워드를 구성할 때 부분적으로 참여시킨다. 부호기의 경우 11개의 프레임을 저장할 수 있는 메모리로 구성되어 있고, 복호기의 경우 56개의 프레임을 저장할 수 있는 메모리로 구성되어 있다.

인터리버 컨트롤러는 부호화 혹은 복호화 과정을 진행하는 프레임 번호를 입력받아 코드워드 구성에 필요한 메모리의 주소(Address)를 생성한다. 주소를 입력받은 메모리는 해당 주소의 데이터를 출력하고 인터리버에서 코드워드를 구성한다. 이전 수신 프레임 데이터로부터 상위 510bit를 구성하고 현재 진행하는 프레임의 데이터를 하위 510bit에 위치시키는 역할을 한다. 이 후 비트 인터리버(Bit interleaver)를 통과하여 부호기와 복호기에 입력된다. 비트 인터리버는 데이터의 순서를 재배열하는 역할을 한다. 디인터리버 블록은 인터리버 블록의 반대 과정을 통해 메모리로 데이터를 입력하는 역할을 한다.
Figure 23. Block diagram of the Interleaver/Deinterleaver (a) Interleaver/Deinterleaver of the encoder, and (b) Interleaver/Deinterleaver of the decoder
5. 기능 검증 및 성능 분석

5.1 기능 검증

구조의 기능 검증을 위하여 멘토그래픽스사(Mentor Graphics)의 모델링 6.2i를 이용하여 RTL 코드의 시뮬레이션을 수행하였다. 그림 24는 기능을 검증하기 위한 실험 절차를 보여준다. 그림과 같이 CI-BCH를 상위 레벨 언어인 C 언어로 구현하고 각각의 입력력 데이터를 텍스트 파일로 저장한다. 각각의 입력력 데이터는 C에서 랜덤(Random) 생성한 데이터를 이용하였고, 그 중 복호기의 입력은 C로 모델링(Modeling)된 AWGN (Additive White Gaussian Channel)을 통과시켜 랜덤 오류를 삽입하였다. 그 후 Verilog-HDL로 CI-BCH 부호기와 복호기의 하드웨어를 구현하여, 모델링 시뮬레이션을 이용하여 C 언어로 구현된 CI-BCH로부터 얻은 입력 텍스트를 입력한 후 그 출력을 다시 저장된 텍스트와 비교하는 과정을 거쳐 기능을 검증하였다.

그림 24. 검증 절차

Figure 24. Verification procedure
5.1.1 CI-BCH 부호기의 기능 집중

CI-BCH 부호기는 앞서 살펴본 바와 같이 64채널로 구성되어 있고 각각은 독립적으로 부호화를 진행한다. 인터리버를 통해 이전 수신한 8개의 프로임 데이터와 함께 64개의 코드워드를 형성하여 부호기로 전달하게 된다.

각각의 부호기는 988 bits의 데이터 비트를 한 클록 당 4 bits씩 처리하는 4-병렬 구조를 적용하여 247 클록동안 연산을 수행한다. 이후 8 클록동안 32 bits의 패리티 데이터를 출력으로 보내어 총 1020 bits를 출력하고 255 클록이 소요된다.

그림 25와 26은 CI-BCH 부호기의 모델 및 시뮬레이션도이다. 입력된 데이터가 247 클록동안 부호화 연산과 동시에 출력하고 입력 데이터가 끝난 후 8 클록동안 패리티 데이터를 출력함을 볼 수 있다.

그림 25. 단일 채널 부호기의 시뮬레이션도
Figure 25. Simulation of the single channel encoder

그림 26. 64채널 부호기의 시뮬레이션도
Figure 26. Simulation of the 64 channel encoder
5.1.2 CI-BCH 복호기의 기능 검증

5.1.2.1 신드롬 계산 블록의 기능 검증

신드롬 계산 블록은 부호기에서 소용되는 클록의 수와 같은 255 클록에 연산을 하기 위해 4-병렬 구조로 설계되어 있으며 255 클록 후에 2t-1인 5개의 신드롬 값이 출력된다.

그림 27. 신드롬 계산 블록의 시뮬레이션도

Figure 27. Simulation of the Syndrome computation block

5.1.2.2 KES 블록의 기능 검증

KES 블록은 Dual-pSiBM 구조를 사용하여 구현하였다. 그림 28에서 신드롬 두 개가 순차적으로 KES 블록으로 입력되어 2t 클록인 6 클록동안 연산 후 두 개의 오류 위치 다항식(Error Locator Polynomial : ELP)을 순차적으로 출력함과 동시에 다시 두 개의 신드롬이 입력되어 연산되는 것을 확인 할 수 있다.

그림 28 Dual-pSiBM을 사용한 KES 블록의 시뮬레이션도

Figure 28. Simulation of the KES block using the Dual-pSiBM
5.1.2.3 Chien-Search 블록의 기능 검증

Chien-Search 블록은 앞서 계산된 KES 블록의 오류 위치 다항식을 입력 받아 실제 오류의 위치를 찾아낸다. 그림 29는 Chien-Search 블록의 모델임
시뮬레이션 과정을 보여준다. 입력된 오류 위치 다항식 값에 따라 255 클
록 반복 연산을 하면서 코드워드의 오류의 위치에 Bit Toggle 신호를 보내고 이 Bit Toggle 신호와 FIFO (First In First Out)으로부터 출력된 수신
d데이터와 XOR 연산을 통해 오류를 바로 찾는다.

그림 29. Chien-Search 블록의 시뮬레이션도

Figure 29. Simulation of Chien-search block

5.1.2.4 64채널 BCH(1020,988) 복호기의 기능 검증

그림 30은 64채널 BCH(1020, 988) 복호기의 시뮬레이션도이다. RCode61
과 RCode64에 오류가 포함된 코드워드가 복호기의 입력으로 입력된 후 오
류가 정정된 코드워드 CCod61, CCod64가 복호기의 출력으로 나오는 것
을 확인할 수 있다.

그림 30. 64채널 BCH(1020,988) 복호기의 시뮬레이션도

Figure 30. Simulation of the 64 channel BCH(1020,988) decoder
5.1.2.5 CI-BCH 복호기의 전체 기능 검증

그림 31은 6번의 반복 복호를 하는 CI-BCH 복호기의 전체 시뮬레이션 결과이다. 6개의 64채널 BCH(1020,988) 복호기를 통하여 오류 데이터를 정정하며 입력이 들어간 후 16,830 클록 후 정정된 데이터가 출력한다. 그림 32 (a)는 복호기의 첫 번째 입력 데이터 확인으로 채널을 통과시켜 오류가 생성된 데이터이다. 이러한 오류 데이터는 CI-BCH 복호기에서 6번 반복 복호 후에 그림 32 (b)와 같이 오류가 모두 정정되어 입력으로 사용한 글자인 “Inha University”가 출력되는 것을 확인할 수 있다.

그림 31. 6번 반복 복호 CI-BCH 복호기의 전체 시뮬레이션도

Figure 31. Simulation of the 6 iteration CI–BCH decoder

그림 32. 복호기의 입력/출력 데이터 (a) 오류가 있는 입력 데이터, (b) 복호기 출력 데이터

Figure 32. Input/Output data of the decoder (a) error input data, (b) output data of the decoder
5.1.3 CI-BCH 복호기의 FPGA 견증

Xilinx ISE 10.1과 Xilinx Virtex5 LX330 FPGA 보드를 이용하여 본 논문에서 제안한 CI-BCH 기반 FEC의 기능을 검증하였다. ISE의 ROM IP를 이용하여 C언어로 설계한 CI-BCH를 통해 생성한 입력 데이터를 저장 후 합성(Synthesis) 및 P&R (Place and Route)를 통해 Bit-stream을 생성한 후 Xilinx Chipscope Pro. Analyzer를 이용하여 과정을 분석하였다. 그림 33은 Xilinx Virtex5 LX330 FPGA 보드를 이용한 FPGA 테스트 환경이다. 그림 34는 Xilinx Chipscope Pro. Analyzer를 이용하여 얻은 입력 데이터와 정정된 신호의 출력 과정을 나타낸다.

그림 33. FPGA 보드 테스트 환경

Figure 33. FPGA board test environment
그림 34. Chipscope를 이용한 FPGA 시뮬레이션 결과

Figure 34. FPGA Simulation result using Chipscope
5.2 성능 분석

본 논문에서는 차세대 광통신 시스템을 위한 CI-BCH 기반 FEC구조를 상위레벨 언어로 설계하고 그 성능을 검증하였으며 이를 Verilog-HDL을 이용하여 하드웨어 설계를 하고 멀토그래픽스사의 ModelSim 6.2i를 이용한 시뮬레이션을 통해 기능을 검증하였다. Verilog-HDL을 사용하여 설계한 구조의 결과는 상위레벨 언어를 이용하여 구현한 결과와 일치함을 확인하였 다. 기능의 문제 없음을 확인 후 Synopsys사의 Design-Complier를 사용하 여 로직 게이트 수와 클록 스피드 등을 분석하였다. 그리고 Xilinx ISE와 Virtex5 FPGA를 이용하여 FPGA 시뮬레이션을 진행하여 기능을 검증하고 그 로직 사용량을 분석하였다.

표 2는 RS(255, 239) 부호를 사용한 100 Gb/s급 광통신 시스템용 FEC [19], ITU-T G.975.1의 연결 부호 중 BCH+BCH의 연결 BCH 부호를 사용한 100 Gb/s급 광통신 시스템 FEC [20], Vitesse사의 CI-BCH [3], 본 연구와 같이 진행된 인하대의 H-CBCH [21]와 본 논문에서 연구된 광통신 시스템용 CI-BCH 부호의 성능과 하드웨어 구현 결과를 비교 분석하였다.

표 2의 결과를 통해 알 수 있듯이 [19] [21]에 제안되어 있는 광통신 시스템용 FEC 부호들에 비해 CI-BCH 부호가 높은 NCG 성능을 보여준다. 또한 본 논문에서 제안하는 CI-BCH FEC 구조는 Vitesse사에서 제안하는 CI-BCH FEC 구조에 비해 51%의 낮은 메모리 사용량을 보여주고 있으며 하드웨어 면에서는 Dual-pSiBM구조의 제안으로 47%정도 클록 속도의 향 상과 13%정도 면적 감소의 효과를 보임을 확인할 수 있다.

표 3은 Virtex5 LX330 FPGA와 Xilinx ISE10.1 를 이용하여 합성한 CI-BCH FEC 구조의 FPGA 구현 결과로서 로직 사용량, 클록 속도 및 데이터 처리율과 Block RAM (BRAM)의 사용량 등을 보여준다.
Table 2. Implementation result and comparison of the optical communication FEC architectures

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Redundancy Rate</strong></td>
<td>6.69 %</td>
<td>6.69 %</td>
<td>6.69 %</td>
<td>6.69 %</td>
<td>6.69 %</td>
</tr>
<tr>
<td><strong>Iteration</strong></td>
<td>x</td>
<td>3</td>
<td>6</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td><strong>NCG at 10E−12</strong></td>
<td>5.6 dB</td>
<td>7.98 dB (Theoretically)</td>
<td>8.40 dB (Theoretically)</td>
<td>-</td>
<td>8.37 dB</td>
</tr>
<tr>
<td><strong>NCG at 10E−15</strong></td>
<td>6.2 dB</td>
<td>8.99 dB (Theoretically)</td>
<td>9.35 dB (Theoretically)</td>
<td>9.05 dB</td>
<td>9.32 dB</td>
</tr>
<tr>
<td><strong>Memory Size</strong></td>
<td>7,728</td>
<td>252,576</td>
<td>700,000</td>
<td>244,512</td>
<td>347,136</td>
</tr>
<tr>
<td><strong>Logic Utilization (Virtex5)</strong></td>
<td>-</td>
<td>-</td>
<td>5xRS–8</td>
<td>-</td>
<td>159,898</td>
</tr>
<tr>
<td><strong>Gate Count (IBM90nm)</strong></td>
<td>420,000</td>
<td>2,781,000</td>
<td>5xRS–8</td>
<td>1,866,000</td>
<td>1,839,600 (4xRS–8)</td>
</tr>
<tr>
<td><strong>Clock Rate (Virtex5)</strong></td>
<td>-</td>
<td>-</td>
<td>175</td>
<td>-</td>
<td>258</td>
</tr>
<tr>
<td><strong>Clock Rate (IBM90nm)</strong></td>
<td>620</td>
<td>320</td>
<td>-</td>
<td>430</td>
<td>430</td>
</tr>
<tr>
<td><strong>Latency (Clocks)</strong></td>
<td>161</td>
<td>8,148</td>
<td>-</td>
<td>7,896</td>
<td>16,830</td>
</tr>
<tr>
<td><strong>Throughput (Gb/s)</strong></td>
<td>240</td>
<td>81.9</td>
<td>112.0</td>
<td>110.1</td>
<td>110.1</td>
</tr>
<tr>
<td></td>
<td>Virtex5 LX330</td>
<td>CI-BCH Encoder</td>
<td>CI-BCH Decoder</td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------</td>
<td>----------------</td>
<td>----------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Slice Registers</td>
<td>5,248</td>
<td></td>
<td>140,402</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Slice LUTs</td>
<td>5,632</td>
<td></td>
<td>154,458</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clock Rate (MHz)</td>
<td>312</td>
<td></td>
<td>258</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BRAM Utilization</td>
<td>600</td>
<td></td>
<td>3,360</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Throughput (Gb/s)</td>
<td>79.8</td>
<td></td>
<td>66.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
6. 결론

본 논문에서는 고속의 데이터 전송을 위해 CI-BCH 기반 FEC 구조를 설계하였다. 그리고 CI-BCH 기반 FEC를 상위 레벨 언어인 C 언어로 모델링하고 이미지 코덱 시뮬레이터를 설계함으로써 오류 정정 능력을 확인하였고, BER 성능 그래프를 통해 기존의 표준으로 사용된 RS(255, 239) 부호, ITU-T G.975.1에 제안된 연결 BCH 부호, Vitesse사의 CI-BCH 부호와 성능을 비교하고 분석하였다.

성능 분석결과 10E-15에서 RS(255, 239) 부호와는 약 3.12 dB의 성능 향상을 보였고, 기존의 연결 BCH 부호와는 0.33 dB의 성능 향상을 보여주었다. 1 dB 차이마다 전송거리가 12%정도 증가한다는 것을 생각하면 기존 RS(255,239) 부호보다 40%의 전송거리 향상을 보일 것으로 예상되며, 이는 설치하는 광통신 중계기 개수의 증감 효과를 가져와 장비 비용 절감의 효과를 기대할 수 있다 [22]. 또한 기존의 FEC 부호보다 작은 3중 오류 정정 BCH(1020, 988) 부호를 사용함으로써 작은 하드웨어 사이즈를 가지며 Dual-pSiBM을 사용한 KES 클록을 사용함으로써 클록 스피드를 430 MHz 이상으로 향상시켜 차세대 광통신 시스템의 100Gb/s급 이상의 전송 데이터 처리율을 만족한다.

이러한 점으로 미루어 보아 제안한 CI-BCH 기반 FEC 구조는 높은 오류 정정 능력을 필요로 하는 100Gb/s급 광통신 시스템을 위한 enhanced FEC로서 적용될 수 있을 것이다. 뿐만 아니라 본 논문의 연구 결과를 계속 발전시켜 연구를 진행함으로써 연속 인터리밍 방법과 본 논문에서 제안한 FEC 구조를 이용하여 광통신 시스템뿐만이 아닌 무선 통신 시스템 등의 다른 영역 (Redundancy rate)를 가지는 시스템에 적용하여 시스템의 신뢰도 향상을 기대할 수 있을 것이다.
7. 참고 문헌


