논문 20-45-05-20 The Journal of Korean Institute of Communications and Information Sciences '20-05 Vol.45 No.05
https://doi.org/10.7840/kics.2020.45.5.903

효율적인 SW 기반 BCH 부복호 구현에 관한 연구

이 종 훈◆, 김 수 영°, 송 상 섭*

A Study on Efficient Implementation of SW-Based BCH Encoding and Decoding

Jonghoon Lee◆, Sooyoung Kim°, Sangseob Song*

요 약

본 논문에서는 무선통신인체망 표준으로 지정되어 있는 2비트 오류 정정 BCH 부호에 대한 효율적인 부복호기를 설계하고, 간단한 방법으로 오류 정정 효율을 개선할 수 있는 방법을 제안하였다. 2비트 오류정정이 가능한 BCH 부호는 비교적 간단한 오류정정 부호 방식으로, 부복호 과정의 효율화가 매우 중요하다. 본 논문에서 제안된 방식은 소프트웨어 기반으로 부복호 방식을 구현할 때, 그 효율성이 극대화 될 수 있는 방식이다. 나눗셈 나머지 표를 이용한 부호화와 신드롬-오류위치 표를 이용한 복호 방식을 적용함으로써, 성능이 낮은 처리장치를 가진 시스템에서도 매우 적은 복잡도로 효율적인 부복호가 가능하다. 또한, 제안된 방법은 오류정정 복호 과정에서 수신된 비트의 신뢰도 정보가 낮은 비트를 선택적으로 비트 반전하는 방법을 적용하여, 연판정 복호와 유사한 장점을 취할 수 있도록 하였다. 본 논문에서 제시된 시뮬레이션 결과를 통하여, 제안된 방식이 비트오류율 성능 개선 효과가 있음을 보였으며, 그 효과는 모부호를 단축한 부호에서 더 크다는 것을 알 수 있다.

Key Words : error correction codes, BCH encoding, BCH decoding, shortened code, syndrome

ABSTRACT

In this paper, we design an efficient 2-bit error correction BCH code, which is specified in the WBAN standard, and we also propose a simple method to improve the error correction performance. The 2-bit error correction BCH code is a relatively simple error correction code, and thus the efficiency of the encoding and decoding processes is very important. The efficiency of the proposed method can be maximized when it is implemented in software. By applying the encoding method using the division-remainder table as well as the decoding method using the syndrome-error location table, the proposed method can be efficiently operated with very low computational complexity, even in a low performance device. In addition, the proposed method adopts an efficient performance enhancing method by using a bit inversion process for low-reliability bits, thereby taking advantages similar to soft decision decoding. From the simulation results presented in this paper, we show that the proposed scheme improves a bit error rate performance compared to the conventional scheme with very low complexity, and the advantages are maximized in the shortened codes.

※ 이 논문은 2017년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(No. 2017R1D1A1B03027939).
◆ First Author : Jeonbuk National University Div. of Electronic Eng., jonghoon@jbnu.ac.kr, 학생(박사), 정회원
° Corresponding Author : Jeonbuk National University Div. of Electronic Eng., sookim@jbnu.ac.kr, 정교수, 종신회원
* Jeonbuk National University Div. of Electronic Eng., ssong@jbnu.ac.kr, 명예교수, 종신회원
논문번호 : 202001-013-A-RE, Received January 11, 2020; Revised February 27, 2020; Accepted February 29, 2020

Ⅰ. 서 론

Bose-Chaudhuri-Hocquenghem(BCH) 부호는 선형 순환 블록 부호 중 가장 강력한 부호로 알려져 있으며, 갈로아장(Galois field; GF)을 기반으로 부호화 및 복호가 이루어진다[1]. BCH 부호에 대한 다양한 부호 및 복호 알고리즘이 제안되어 왔으며, 이들은 대부분 대수적인 성질을 이용한 경판정(hard decision) 기반의 알고리즘들이다. 현대의 무선통신 시스템에서는 연판정 정보와 반복적인 복호를 통해서 매우 우수한 성능을 나타내는 터보부호나 저밀도패리티체크 부호들이 연구되고 있지만 비교적 복잡하고 긴 연산이 필요하기 때문에 간단한 오류 정정을 위한 분야에는 적합하지 않다. 반면에 1~2비트의 오류 정정이 필요한 시스템에는 BCH 부호를 적용하는 것이 적합하다.

예를 들어 무선인체통신망(wireless body area network; WBAN)은 인체와 근접해 있는 다양한 휴대 정보 단말기 사이에 통신을 제공하는 근거리통신기술로서 데이터의 오류 정정을 위해 2비트 오류 정정 BCH 부호가 표준으로 적용되고 있다[2,3]. IEEE 802.15.6 WBAN 규격에서는 PLCP(physical layer convergence procedure) 헤더용으로 (31,19) BCH 부호를, PSDU(PHY service data unit) 용으로 (63,51) BCH 부호를 표준으로 정의하고 있다. 또한, PHR(physical layer header) 정보 프레임과 HCS (header check sequence) 용으로는 (40,28) BCH 부호를 정의하고 있다. 이 부호들은 모두 (63,51) BCH 부호와 이를 모부호(mother code)로 하여 축약된 (shortened) 부호들이다.

그리고 최근 저장매체로 널리 활용되고 있는 플래시 메모리의 경우 저장 용량의 확장을 위해서 단일 레벨 셀(single level cell; SLC)에서 멀티 레벨 셀(multi level cell; MLC) 및 트리플 레벨 셀(triple level cell; TLC) 등으로 기술이 확장되어 제조되고 있다. 다중 레벨 셀에 데이터를 저장하게 되면 저장 용량은 증가하지만 저장된 데이터를 읽는 과정에서 레벨 간 간섭에 의한 오류가 발생할 확률이 높아진다. 이러한 문제를 해결하기 위해 비교적 간단한 회로로 구현할 수 있는 BCH 부호를 적용하는 연구가 활발하게 진행되고 있다[4,5].

BCH 부호가 적용되는 분야는 비교적 적은 개수의 비트 오류를 정정하기 위한 분야이기 때문에 간단하고 효율적인 부복호 알고리즘이 필요하다. BCH 부호는 GF에서 정의된 대수적 연산을 기반으로 부복호가 수행된다. 이러한 점을 고려하여, 기존의 연구에서는 BCH 부호에 대한 부복호 과정 중 복잡도 측면에서 비교적 큰 비중을 차지하는 복호 과정에 대한 복잡도를 줄여서 하드웨어를 구현하기 위한 다양한 방법이 제안되고 있다[6]. 이들 연구의 상당수는 BCH 복호에 가장 흔히 사용되는 Berlekamp- Massey algorithm (BMA)의 일부 연산 과정의 복잡도를 줄이는 방식들이며[7], 대부분 소형화된 하드웨어 구현을 목표로 하는 연구들이었다[8,9].

본 논문에서는 위에 기술한 바와 같이 오류 정정 능력이 2비트 이하인 BCH 부호가 적용되는 WBAN과 같은 분야에서 매우 적은 복잡도로 부복호가 가능한 방식을 제안한다. 부호화 과정에서는 기존의 쉬프트 레지스터를 이용한 나눗셈 연산 대신에 간단한 나눗셈 연산의 원리를 적용한 효율적인 룩업테이이블(look up table; LUT)을 사용하는 방법을 적용한다. 복호 과정에서는 기존의 BMA 및 Chien search의 모든 연산을 수행하는 대신에 간단한 신드롬-오류위치 LUT를 이용한다. 특히, 축약된 부호가 적용되는 경우, 의미 있는 신드롬 값을 찾을 수 없는 경우가 모부호에 비해 더 많다는 점에 착안하여, 수신된 비트의 신뢰도가 낮은 비트들의 값을 반전시킴으로써 오류 성능을 개선시킬 수 있는 방식을 제시한다.

본 논문에서는 WBAN의 표준으로 지정된 (63,51) BCH와 그 단축부호인 (40,28) BCH, (31,19) BCH부호를 사용하여, 시뮬레이션 결과를 제시함으로써 제안된 방식이 기존 방식에 비하여 더 적은 복잡도로 우수한 성능을 도출할 수 있음을 보인다. 제안한 방법은 SW를 기반으로 부복호 방식을 구현하는 시스템에 매우 효율적으로 적용이 가능하며, 특히 정보어의 길이를 줄여서 적용하는 단축부호에 효과적으로 적용할 수 있다.

본 서론에 이어서 II 장에서는 기존의 BCH 부호에 대한 부복호 과정에 대해서 기술하고, III 장에서는 복잡한 연산 과정이 필요 없는 제안된 부복호 방식을 제안하고, 신뢰도가 낮은 비트를 선택적으로 반전하여 오류 정정 능력을 향상시키는 방법과 그 결과에 대해서 자세하게 설명한다. 그리고 IV 장에서는 제안한 방법에 대한 성능 모의실험 결과를 기술하고 V 장에서 결론을 맺는다.

Ⅱ. BCH 부호에 대한 기존의 부복호 방식

2.1 부호화 방식

BCH 부호는 다중 오류를 정정할 수 있는 순회 블록 부호(cyclic block code)로서, 정정 가능한 최대 오류

개수에 근거하여 부호의 생성 다항식 $g(x)$를 설계한다. 따라서 BCH 부호에 대한 부복호 알고리즘은 정정 가능한 오류의 개수 t에 비례해서 복잡도가 증가한다.

BCH 부호는 유한체인 GF에서 모든 연산이 이루어지고, t개의 오류를 정정할 수 있도록 설계된 (n,k) BCH 부호의 부호어(codeword) 길이 n은 항상 $n=2^{m-1}, m \geq 3$을 만족한다. 여기서, 정보어의 길이는 k이고, 패리티의 길이는 $n-k$, 최소 거리 $d_{\min} \geq 2t+1$이며, 이진(binary) 및 비이진(non-binary) 부호의 설계가 가능하다[1]. 부호어의 생성, 즉 부호화 과정은 일반적으로 정보어(메시지)를 다항식으로 나타낸 $m(x)$와 생성다항식(generation polynomial) $g(x)$를 연산하여, 부호어 다항식 $c(x)$를 구하는 과정이다. 이때, $m(x)$와 $g(x)$의 차수는 각각 정보어와 부호어의 길이에 의해 결정되기 때문에, $g(x)$의 차수는 $(n-k)$가 된다.

임의의 BCH 부호에 대해서 t와 n 또는 k가 결정되면, 해당되는 GF와 그 원소들, $\alpha^i (i \geq 0)$을 정의하고, GF 내에서의 연산을 통하여 $g(x)$를 설계할 수 있다. 예를 들어, WBAN의 규격으로 지정되어 있는 (63, 51) BCH 부호와 그 축약부호는 GF(2^6)에서 연산이 이루어지는 이진 부호로서, 일반적으로 아래와 같은 $g(x)$를 사용한다[2].

$$g(x) = x^{12}+x^{10}+x^8+x^5+x^4+x^3+1. \quad (1)$$

결과적으로 (63,51) BCH 부호의 부호화 과정은 입력되는 51비트의 메시지 데이터에 12비트를 추가하여 총 63비트로 구성되는 부호어를 생성하는 과정이다. 특히, 부호어의 앞부분에 메시지가 그대로 포함되어 있는 시스터메틱 (n,k) BCH 부호에 대한 부호어 다항식 $c(x)$는 다음과 같이 나타낼 수 있다.

$$c(x) = \sum_{i=0}^{n-1} c_i x^i = x^{n-k} m(x) + r(x), \quad (2)$$

여기서 메시지 다항식 $m(x)$와 패리티 다항식 $r(x)$는 다음과 같다.

$$m(x) = \sum_{i=0}^{k-1} m_i x^i, \quad (3)$$

$$r(x) = \sum_{i=0}^{n-k-1} r_i x^i = x^{n-k} m(x) \bmod g(x). \quad (4)$$

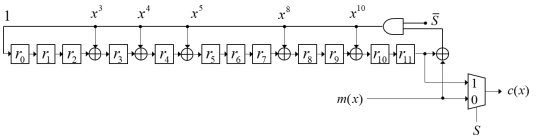

그림 1. 쉬프트 레지스터를 이용한 시스터메틱 (63, 51) BCH 부호기 블럭도
Fig. 1. Block diagram of the systematic (63,51) BCH encoder using a shift register.

위 (4)에서 표현된 것과 같이, $r(x)$는 $x^{n-k}m(x)$를 $g(x)$로 나눈 나머지 값을 다항식으로 표현한 것으로 $(n-k-1)$차 다항식으로 표현된다.

축약 BCH 부호는 부호기에 입력되는 메시지의 길이만 줄이고, 모부호와 동일한 12 비트의 패리티를 추가하여 부호어를 생성하게 된다. 따라서 축약 부호인 (40,28) BCH 및 (31,19) BCH의 부호화 과정은 모부호인 (63,51) BCH 부호와 동일하다. 위 (2)-(4)의 수식을 이용한 부호화 과정은 일반적으로 쉬프트 레지스터(shift register)와 MUX를 이용하여 구현할 수 있다[10]. 위 그림 1은 시스터메틱 (63,51) BCH 부호기의 블록 다이어그램을 나타낸 것이다.

그림 1의 부호기 구조를 참조하면, 부호화 과정에 앞서 모든 레지스터는 '0'으로 초기화되어 있다. 부호기에 51 비트의 메시지 데이터가 순차적으로 모두 입력되면 쉬프트 레지스터에는 12비트로 구성된 $r(x)$의 계수에 해당하는 값들이 남게 된다. 이 과정은 메시지 $m(x)$를 생성다항식 $g(x)$로 나누는 것과 동일한 결과이고, 최종적으로 레지스터에 남아있는 값이 $r(x)$에 해당한다. 쉬프트 레지스터에 저장된 $r(x)$ 12비트는 메시지 데이터의 뒤를 이어서 순차적으로 출력되어 최종적으로 부호어 $c(x)$를 구성한다.

2.2 BMA를 이용한 BCH 부호의 복호

가장 일반적으로 사용되는 BCH 부호의 복호 방법에 사용되는 알고리즘은 BMA로서, 그림 2와 같이 그 전 후로 신드롬 계산, 오류 위치 다항식 결정, 오류 위치 계산 및 오류 정정의 과정을 수반하여 진행된다. 특히 신드롬 계산 후 오류 위치 다항식을 결정하는 BMA와 그 해를 찾는 과정인 Step 2와 Step 3에 가장 많은 연산이 요구되기 때문에 이 과정에서의 복잡도를 줄이는 것이 중요하다.

BMA를 이용하여 오류위치 다항식의 계수를 생성하기 위해서는 반복적인 곱셈과 나눗셈이 필요하다. 비이진 BCH 부호의 경우에, 나눗셈은 결국 역원을 구하는 과정이므로 나눗셈이 필요 없는 Inversion-less BMA를 사용하기도 한다[4]. 또한, 오류 위치 다항식의

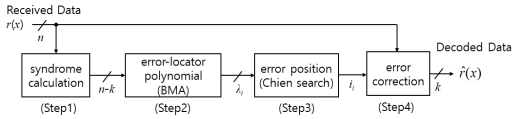

그림 2. BMA에 기반한 BCH 부호의 복호기 블럭도
Fig. 2. Block diagram of the BCH decoder based on BMA.

해를 찾기 위해서 널리 사용되는 Chien search 알고리즘 역시 반복적인 곱셈이 많이 필요하다. BMA 복호는 선형피드백-쉬프트레지스터(LFSR)를 이용하는 것으로, $2t$개의 신드롬 $S_1, S_2, \cdots S_{2t}$와 connection 다항식 $\Lambda(x)$을 LFSR에 입력하여 최종적으로 오류위치다항식 $\Lambda(x)$와 그 해를 찾는다[10]. 다음은 BMA의 의사코드이다.

```
Λ(x) = 1;                    /*connection polynomial*/
T(x) = xΛ(x) = x;            /*correction polynomial*/
L = 0;                       /*L = degΛ(x) */
k = -1;
n = 0;
for (n = 0; n < 2t; n++) {
  Δ = Σ_{i=0}^{L} Λ_i S_{n-i};
  if (Δ == 0) {
    N(x) = Λ(x);             /*if Δ = 0, keep same Λ(x) */
  } else {
    N(x) = Λ(x) + ΔT(x);
    if (L < n - k) {
      L = n - k;
      k = n - L;
      T(x) = Δ^{-1}Λ(x);
    }
  }
  T(x) = xT(x);
  Λ(x) = N(x);
}
```

위 의사코드를 참조하면, BMA는 두 개의 다항식 $\Lambda(x)$와 $T(x)$를 갱신하는 알고리즘이다. 조건에 따라서 각각 $\Lambda(x)+\Delta T(x)$를 $\Lambda(x)$로, $\Delta^{-1}x\Lambda(x)$를 $T(x)$로 갱신한다. 이 과정에서 곱셈 및 나눗셈 연산이 빈번하게 필요하다. 발생한 오류의 개수를 ν라고 할 때 BMA가 종료되면 다음과 같은 오류 위치 다항식을 얻을 수 있다.

$$\Lambda(x) = 1 + \Lambda_1 x + \cdots + \Lambda_\nu x^\nu \tag{5}$$

그림 3은 위 식 (5)의 해를 찾는 방법으로 널리 사용되는 Chien search에 대한 블록다이어그램으로서, α^i가 $\Lambda(x)$의 해가 되는지 알기 위해 가능한 모든 α^i

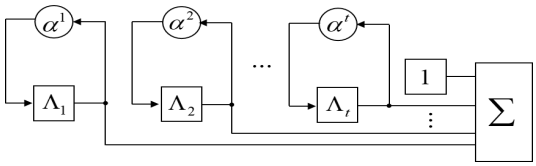

그림 3. Chien search 블록도
Fig. 3. Block diagram of the Chien search

를 대입하여 그 해를 찾는 과정을 구현한 것이다. 이렇게 찾은 α^i로부터 오류의 위치 i를 구할 수 있고, 그림 3과 같이 Chien search로 해를 찾기 위해서는 t개의 블록에서 부호어의 길이만큼의 곱셈이 필요하다.

Ⅲ. 제안된 SW 기반 BCH 부복호 및 성능 향상 기법

3.1 나눗셈 LUT를 이용한 SW 기반 효율적인 부호화

시스터메틱 BCH 부호의 패리티 값인 식 (4)의 $r(x)$는 아래 식 (6)과 같이 표현할 수 있다.

$$r(x) = \sum_{i=0}^{n-k-1} r_i x^i = m(x)\sum_{i=0}^{k} x^{n-i} \bmod g(x) \tag{6}$$

위 식 (6)을 참조하면, $r(x)$는 $m(x)$의 값이 0이 아닌 경우에 대해서 해당 승수에 해당하는 x^{n-i} 값을 생성다항식 $g(x)$로 나눈 값을 모두 더해서 얻을 수 있음을 알 수 있다. 이와 같은 사실에 기반하여, SW 기반의 효율적인 부호화 과정으로 LUT를 이용한 방식을 제안한다. 즉, 부호화 과정을 위하여 그림 1의 쉬프트 레지스터 연산을 수행하지 않고, 부호화에 필요한 $r(x)$값은 x^{n-i}를 $g(x)$로 나눈 나머지의 값, ρ_i를 사전에 모두 계산해 놓고 메시지 데이터 $m(x)$의 i번째 비트가 1인 경우에 해당하는 ρ_i의 값을 모두 더해서 얻을 수 있다. 아래 그림 4는 나눗셈-나머지 LUT를 이용한 부호화기 블럭도를 나타낸 것이다.

나눗셈-나머지 LUT를 이용한 BCH 부호화의 과정은 매우 단순하고 간단하다. 예를 들어 (63,51) BCH

i	ρ_i(HEX)	ρ_i(BIN)
12	539	010100111001
13	A72	101001110010
14	1DD	000111011101
⋮	⋮	⋮
60	2A7	001010100111
61	54E	010101001110
62	A9C	101010011100

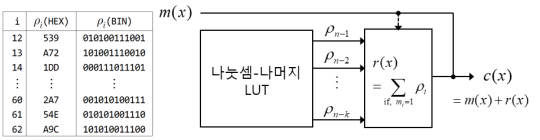

그림 4. 나눗셈-나머지 LUT를 이용한 부호화
Fig. 4. Encoding using division-remainder LUT.

부호의 메시지 데이터가 $m(x)=x^{50}+x+1$이라면 $r(x)=x^{12}m(x)\bmod g(x)$이다. 결과적으로 $r(x)$는 $x^{62}\bmod g(x)$, $x^{13}\bmod g(x)$, $x^{12}\bmod g(x)$의 결과를 모두 더한 것과 같다. 이들 3개의 값은 나눗셈-나머지 LUT에 저장된 ρ_{62}, ρ_{13}, ρ_{12}를 의미하고 그 값은 각각 “101010011100”, “101001110010”, “010100111001”이다. GF에서 덧셈은 XOR 연산으로 구현되고 결과적으로 $r(x)$는 이 값을 모두 더한 “010111010111”이 된다. 같은 방법으로 51비트로 구성된 어떤 메시지 데이터 $m(x)$에 대해서도 최대 51회의 XOR 연산만으로 $r(x)$를 구해서 부호화 과정이 완료된다. 이와 같이 BCH 부호의 부호화는 나눗셈-나머지 LUT를 이용하면 복잡한 쉬프트 레지스터를 이용하지 않고 매우 간단한 XOR 연산만으로 구현이 가능하다.

3.2 신드롬-오류위치 LUT를 이용한 SW 기반 효율적인 복호

복호의 첫 번째 과정인 신드롬 계산은 $g(x)$가 최소다항식의 LCM이라는 관계를 이용하는 것이다[1]. 예를 들어, (63,51) BCH 부호의 경우 최소 다항식의 해가 GF(2^6)에서 생성된 α와 α^3이기 때문에, 부호어 다항식도 α와 α^3을 해로 가져야 한다는 점을 이용하여 해당되는 신드롬 S_1과 S_3를 다음과 같이 구할 수 있다.

$$S_1=c(\alpha)=\alpha^{i_1}+\alpha^{i_2}=X_1+X_2, \tag{7}$$

$$S_3=c(\alpha^3)=\alpha^{3i_1}+\alpha^{3i_2}=X_1^3+X_2^3, \tag{8}$$

여기서 X_1과 X_2는 각각 α^{i_1}과 α^{i_2}이고, i_1과 i_2는 오류의 위치를 의미한다. GF(2^6)에서는 +부호와 -부호는 등가이므로 오류 위치 X_1과 X_2를 각각 근으로 갖는 오류 위치 다항식을 다음 식 (9)와 같이 신드롬 값을 이용하여 표현할 수 있다.

$$\begin{aligned}\Lambda(x)&=(x-X_1)(x-X_2)\\&=x^2+(X_1+X_2)x+X_1X_2\\&=x^2+S_1x+\frac{S_3+S_1^3}{S_1}.\end{aligned} \tag{9}$$

일반적인 BCH 복호 과정에서 오류위치를 찾는 방법은 오류위치다항식 $\Lambda(x)$의 해를 Chien search로 찾는 것이다. 이를 위해서는 매우 많은 횟수의 곱셈 연산이 필요하다. 본 논문에서는 신드롬을 구하고, BMA를 이용하여 오류위치다항식을 생성하고, 오류위치를 찾는 기존의 방법 대신에 계산된 신드롬으로부터 오류위치를 매핑할 수 있는 방법을 제안한다. 오류의 패턴은 매우 다양하게 나타날 수 있지만 이런 오류 패턴으로부터 생성되는 신드롬 값은 한정적인 점을 이용하여 수정이 가능한 모든 오류 패턴에 대해서 신드롬 값을 미리 계산해 놓은 LUT을 이용한다. 예를 들어, 오류 정정 능력 t=2인 (n,k) BCH 부호의 경우에 발생할 수 있는 오류의 개수에 따른 신드롬의 조건을 정리하면 다음 표 1과 같다.

이 경우 부호어에 오류가 없는 경우 1가지와 1비트 오류가 발생한 경우 n가지, 그리고 2비트의 오류가 발생한 경우 ${}_nC_2$ 가지에 대해서는 모든 오류를 정정할 수 있다. 이와 같이 신드롬 패턴이 저장된 신드롬-오류위치 LUT를 사용하면, 수신된 데이터로부터 신드롬 값을 얻은 후에 그 신드롬에 해당하는 오류의 위치가 미리 정의 되어 있는 표를 이용하므로 오류 위치 다항식을 생성하고 그 해를 찾는 복잡한 과정이 필요 없다. 그리고 만약 정정 능력을 초과하는 개수의 오류가 발생한 경우에는 오류 정정 하지 않고 수신된 부호어의 시스터메틱 부분을 그대로 출력한다.

이제 WBAN에서 사용되고 있는 (63,51) BCH 부호 및 그 축약 부호를 이용하여 예를 들어 보면, 이 부호들은 GF(2^6)에서 생성되므로 각 신드롬도 역시 6비트로 구성된다. 따라서 S_1과 S_3는 총 12비트로 구성되므로 모두 2^{12}(=4096) 가지의 서로 다른 신드롬 패턴이 존재할 수 있다. 신드롬-오류위치 LUT는 각각 6비트로 표현되는 신드롬 S_1과 S_3를 주소로 이용하고, 각 주소에는 오류 위치 i_1, i_2에 대한 정보를 저장하고

표 1. 오류 개수에 따른 신드롬 조건
Table 1. Syndrome conditions by the number of errors.

no. of errors	Condition of syndrome	Error location $\{i_1,i_2\}$	no. of cases
0	$S_1=0, S_3=0$	-	1
1	$S_1\neq 0$, $S_3=S_1^3$	{0},{1}, ... ,{n}	n
2	$S_1\neq 0$, $S_3\neq S_1^3$	{0,1},{0,2}, ..., {n, n-2}, {n, n-1}	${}_nC_2$
3 or more	$S_1=0, S_3\neq 0$ $S_1\neq 0$, $S_3\neq S_1^3$	{0,1,2}, {0,1,3}...	$2^{12}-(1+n+{}_nC_2)$

있다. 오류 위치 i_1, i_2는 0과 62사이의 값이므로 하나의 오류 위치를 표현하기 위해서는 최소 6비트가 필요하고 오류 위치 2개를 저장하기 위해서는 최소 12비트가 필요하다. 따라서 12비트로 구성된 주소와 12비트 데이터로 구성된 LUT를 이용하면 2비트 이하로 발생하는 모든 오류에 대응할 수 있다. 신드롬 값으로 생성 가능한 4,096개의 주소 중에서 $(1+n+{}_nC_2)$개는 2비트 이하의 정정 가능한 오류 위치가 저장되어 있고 나머지는 정정 불가능한 오류가 발생한 경우이다. 정정 불가능한 경우나 오류가 없는 경우의 오류 위치 값은 부호어의 길이가 최대 63비트인 점을 고려하면 0~62 범위가 아닌, 63(="111111")을 사용하였다.

(63,51) BCH 부호에 적용할 수 있는 LUT의 내용 중 일부를 아래 표 2에 나타내었다. 2비트 오류가 발생한 것으로 계산된 신드롬을 주소로 한 경우에 오류 위치에는 i_1과 i_2의 값이 각각 하위 6비트와 상위 6비트로 나누어서 저장되어 있다. 1비트 오류가 발생한 경우에는 상위 6비트, 즉 i_2 값에는 63(=111111)이 저장되어 있고 하위 6비트에는 i_1에 해당하는 오류 위치 값이 저장되어 있다. 오류가 전혀 없거나 3비트 이상의 오류가 있는 신드롬의 경우에는 12비트 주소 값이 모두 1로 구성된 값(=111111 111111)이 저장된다. 오류 위치 LUT에는 3비트 이상의 오류가 발생한 경우에도 신드롬 값은 계산되어 주소가 할당되지만, 그 주소에 있는 위치 값이 모두 63을 나타내므로 오류를 정정하는 과정은 진행되지 않는다. 축약 부호인 (40,28) BCH 및 (31,19) BCH의 경우에도 해당 규칙에 따라 적절한 오류 위치가 저장된 LUT를 생성하여 적용한다. 결과적으로 축약 부호의 LUT에는 모부호보다 오류 위치 값으로 "111111(63) 111111(63)" 값이 저장된 주소가 더 늘어나게 된다.

표 2. (63,51) BCH 부호에 대한 신드롬과 오류위치로 구성된 LUT
Table 2. LUT of syndromes and error location for (63,51) BCH code.

no. of errors	Addresses		Error positions	
	S_3	S_1	i_2(number)	i_1(number)
No error (1 case)	000000	000000	111111 (63)	111111 (63)
1 error (63 cases)	100001	111001	111111 (63)	111110 (62)
	010001	110100	111111 (63)	111101 (61)
	⋮	⋮	⋮	⋮
	001000	000010	111111 (63)	000001 (1)
	000001	000001	111111 (63)	000000 (0)
2 errors (1953 cases)	011111	011111	111101 (61)	111110 (62)
	110101	110001	111100 (60)	111110(62)
	⋮	⋮	⋮	⋮
	000110	101001	000001 (1)	111110 (62)
	001111	101010	000000 (0)	111110 (62)
	101010	101110	111100 (60)	111101 (61)
	011110	111001	111011 (59)	111101 (61)
	⋮	⋮	⋮	⋮
	011001	110110	000001 (1)	111101 (61)
	010000	110101	000000 (0)	111101 (61)
	…	…	…	…
	000010	000101	000010 (2)	000000 (0)
	001001	000011	000001 (1)	000000 (0)
3 or more errors (2079 cases)	000001	000000	111111 (63)	111111(63)
	000010	000000	111111 (63)	111111(63)
	⋮	⋮	⋮	⋮
	******	******	111111(63)	111111(63)

위 표 2와 같은 LUT를 이용한 복호는 신드롬 값을 계산한 이후에 바로 적용할 수 있다. 또한 신드롬 값을 주소로 이용하여 저장된 12비트의 데이터에는 상위 및 하위 6비트가 각각 오류가 발생한 비트의 위치 정보를 가지고 있으므로 발생한 오류의 위치와 개수를 바로 알 수 있다. 오류의 위치가 모두 63일 경우는 오류가 없거나 3비트 이상의 오류가 발생한 경우이다. 즉, 2개의 오류의 위치가 모두 63일 때 두 개의 신드롬 S_1 및 S_3가 모두 0인 경우는 오류가 발생하지 않은 것이고, 두 개의 신드롬 값이 모두 0이 아닌 경우는 3비트 이상의 오류가 발생한 것으로 판단할 수 있다. 이 두 가지 경우 모두 수신된 부호어에 대해서 오류를 정정할 필요가 없다.

아래 그림 5는 제안된 LUT를 이용한 복호기의 블럭도를 나타낸 것으로, 오류위치다항식을 생성하고 오류위치를 찾는 복잡한 과정이 필요하지 않다. 신드롬-오류위치 LUT는 복호 과정 전에 미리 생성되어 적용하므로 복호 과정에서는 오직 신드롬 계산만 필요하고 계산된 6비트의 신드롬 S_1과 S_3은 12비트로 구성된 LUT의 주소로 사용된다.

표 2에서 보여준 것과 같이 신드롬 값으로 구성된 주소에 저장된 내용에는 오류위치 2개(i_1,i_2)가 저장

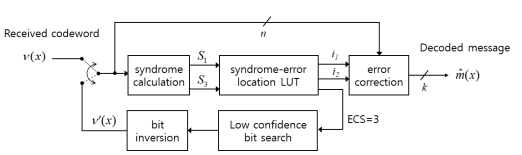

그림 5. 제안된 (n, k) BCH 부호의 복호기 블럭도
Fig. 5. Block diagram of the proposed (n, k) BCH decoder.

되어 있다. 오류위치의 값 i_1과 i_2에는 0에서 62까지의 위치정보가 각각 저장되어 있고 위치정보는 수신된 데이터에서 오류가 발생한 비트의 번호를 의미한다. 만일 오류위치 정보가 "63"일 경우에는 오류의 위치가 지정되지 않은 경우이다. 즉, 오류 위치가 63일 경우는 오류가 없거나, 오류를 정정할 수 없는 경우이다. 따라서 i_1과 i_2의 정보에 따라서 오류수정상태(Error Correction State; ECS)는 3가지 상태에 대한 정보를 출력한다.

ECS가 0일 경우는 수신된 데이터에 오류가 없음을 의미하고, ECS가 1 또는 2인 경우는 각각 1비트 또는 2비트의 오류가 발생한 것으로 판단하고 오류를 정정할 수 있는 상태를 의미한다. ECS가 3인 경우는 계산된 신드롬 값이 오류를 정정할 수 없음을 의미하므로, 오류를 정정하지 않고 시스터메틱 부분을 그대로 출력하는 것이 일반적이다. 이상의 복호 과정은 기존의 BMA를 이용한 복호화 동일한 결과를 얻을 수 있다. 본 논문에서는 위의 LUT를 이용하여 복호 과정의 복잡도를 현저하게 감소시킬 수 있는 방법과 더불어, 신뢰도가 낮은 비트들을 선별하고 신드롬을 재계산함으로서 성능 향상을 도모할 수 있는 방안을 제시한다. 다음 절에서는 제안된 성능 향상 기법에 대한 개념을 설명한다.

3.3 신드롬 재계산을 통한 성능 향상 기법

위 2절에서 제안된 방식에서 설명된 바와 같이, t개 이상의 오류가 발생한 것으로 판단되면 오류 정정 과정 없이 수신된 메시지 데이터를 그대로 출력하게 된다. 이 경우 신드롬이 해당되는 오류위치를 제대로 계산할 수 없음을 의미하는 것인데, 이러한 경우는 축약 부호를 사용할 경우 더 빈번하게 발생하게 된다. 본 논문에서는 그림 5와 같이 정정 능력 이상의 오류가 발생한 것으로 판단될 경우, 수신 비트 중 신뢰도가 낮은 비트에 대하여 선택적으로 비트 반전을 통하여 신드롬을 재계산하는 방식을 통하여 성능 향상 효과를 도모할 수 있는 방법을 제안한다.

제안된 방법은 BCH 부호의 복호 후에도 오류를 정정할 수 없는 것으로 판단된 경우에만 비트 반전과 신드롬 계산 및 재복호를 실행한다. 본 연구에서는 광범위한 시뮬레이션을 통하여 몇 개의 비트에 대한 반전이 가장 효과적인지 또 신뢰도 값에 따른 다양한 반전 기법에 대한 성능을 분석하였다. 이를 통하여 최적의 비트 반전 개수와 신뢰도 값에 따른 반전 유무 판단을 통하여 복잡도의 증가가 거의 없으면서도 가장 효과적인 성능 향상 기법을 적용할 수 있다. 본 연구에서 사용한 (63,51) BCH 부호 및 그 축약 부호들이 가우시안 잡음 채널에서 사용되는 경우 수신 신호 레벨이 평균 수신 신호 레벨의 약 20% 이내에 해당하는 수신 값들만을 선택적으로 반전하여 신드롬을 재계산할 경우 가장 효과적이라는 결과를 얻을 수 있었다. 이에 대한 보다 상세한 분석 내용은 다음 장에서 제시한다.

수신된 부호어에 대하여 계산된 신드롬 값이 ECS 3에 해당할 경우, 신뢰도 값이 특정 문턱 값, *CL* 보다 적은 비트들만을 선별하고, 선택된 비트의 부호를 반전하여 다시 신드롬을 계산한다. 이 경우, 신뢰도 값이 가장 낮은 비트를 선별하여 비트 반전을 순차적으로 수행하는 일종의 블록부호에 대한 연판정 복호 기법을 적용할 수 있으나, 이는 비트 신뢰도를 순차적으로 나열하고 여러 번 신드롬 계산을 하는 복잡도가 증가될 수 있다. 본 연구에서는 광범위한 시뮬레이션을 통하여 매우 낮은 복잡도로 가장 효과적인 성능향상을 이룰 수 있는 방법을 발견하였다. 제안된 알고리즘은 신뢰도 문턱 값, *CL*을 설정하고 이보다 적은 신뢰도를 가지는 비트들을 한 번에 비트 반전하여 추가적으로 한 번 더 신드롬을 계산하는 방식이다. 이 경우 매우 적은 복잡도로 블록부호에 대한 연판정 복호 기법과 거의 유사한 성능 향상 효과를 기대할 수 있다. 다음 장에서는 시뮬레이션 결과를 통하여 제안된 방식이 효과적으로 성능 향상에 기여할 수 있음을 보인다.

Ⅳ. 실험 결과 및 분석

본 장에서는 제안된 방식의 성능 비교를 위하여 이진위상변조(binary phase shift keying; BPSK) 방식을 이용하여 가산성백색가우시간잡음(additive white Gaussian noise; AWGN) 채널에서 (63,51) BCH 부호와 그 단축부호들에 대하여 비트오류율(bit error rate; BER) 성능을 시뮬레이션 결과를 비교 분석한다. 먼저, 다음 그림 6은 AWGN 채널에서 기존의 BMA 복호 방식을 적용하였을 경우에 대하여, 비트에너지대잡음스펙트럼밀도비(bit energy to noise spectral density ratio; E_b/N_0)에 따른 BER 성능을 참고 성능으로 제시하였다.

이제 다음 그림 6의 기존 BCH 부호에 대한 복호 성능을 참고로 하여, 신뢰도가 낮은 비트들을 어떤 방법으로 몇 비트까지 선택하여 신드롬을 재계산하는 것이 가장 효과적인지를 알아보기로 한다. 다음 그림 7과 8은 각각 (63,51) 및 (31,19) BCH 부호에 대하여 신뢰도가 가장 낮은 비트부터 순차적으로 반전시킨

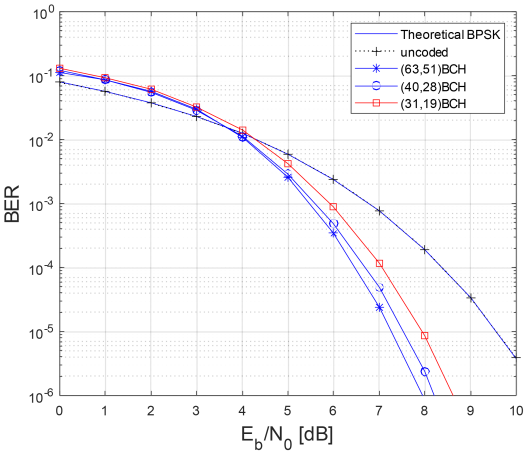

그림 6. (63,51) BCH 부호 및 축약 부호에 대한 기존 복호 방식의 성능 비교
Fig. 6. Performance comparison of the conventional decoding for the (63,51) and its shortened BCH codes

후 복호하는 과정을 최대 9회까지 반복하면서 BER 성능을 시뮬레이션 한 결과를 보여준다. 모부호인 (63,51) BCH 부호보다 그 축약부호인 (31,19) BCH 부호에서 비트 반전 후 복호를 반복했을 때 성능이 더 좋아지는 것을 알 수 있다. (63,51) BCH 부호의 경우 비트 반전 후 복호의 효과가 크게 나타나지 않지만 (31,19) BCH 부호의 경우에는 비트 반전 복호를 1회만 실행해도 BER 값이 10^{-5}에서 0.5 dB 이상의 전력 이득이 발생한다는 것을 알 수 있다. 비트 반전 후 복호를 반복할수록 성능이 점점 향상 되지만 4회 이상 반복했을 경우에는 성능 향상 효과가 미미하거나 오히려 더 악화된다.

다음 그림 7과 8의 비트 반전 개수에 따른 BER 성능 분석 결과, 3회까지 비트 반전 후 복호를 할 경우 성능 향상을 기대할 수 있다는 것을 알 수 있다. 이와 같은 결과를 바탕으로, 1비트 반전 후 신드롬 계산을 3회 반복하는 대신에 한 번에 여러 비트를 동시에 반전하고 신드롬 재계산을 한 번만 실시해 보는 방법을 고려해 볼 수 있다.

다음 그림 9는 1비트씩 최대 3회까지 반전을 반복한 경우와 한 번에 3비트를 동시에 반전한 후 복호 한 결과를 비교한 것이다. 즉 최대 3비트 까지 반전시켜 복호하는 것은 같지만, 신드롬 재계산 과정은 전자는 최대 3회가 필요하고 후자는 1회만 필요하다. 그림 9의 결과는 3비트를 동시에 반전시킨 경우 보다 1비트씩 최대 3회까지 반전해서 복호를 반복한 경우가 오차 범위 내에서 매우 적은 양으로 더 좋은 성능을 보여준다. 후자가 더 좋은 성능을 얻을 수 있지만 연산

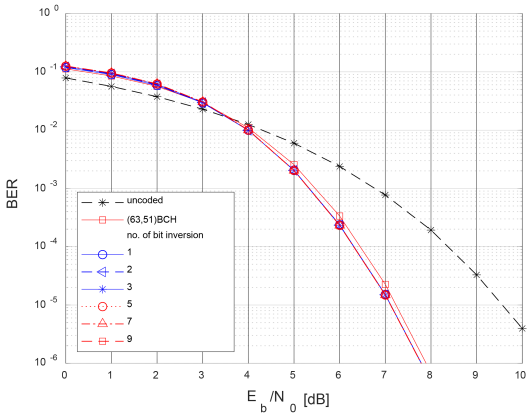

그림 7. (63,51) BCH 부호에 대한 저 신뢰도 비트 반전 개수에 따른 BER 성능 비교
Fig. 7. BER performance comparison of the (63,51) BCH code according to the number of bit inversions.

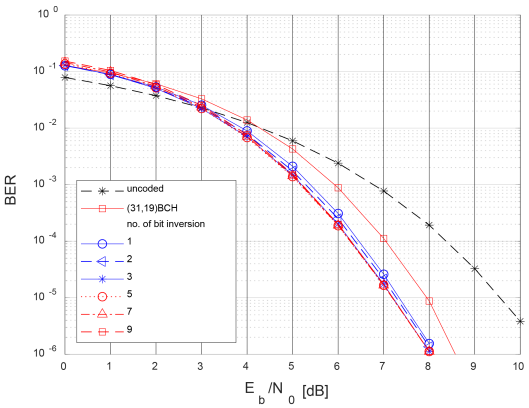

그림 8. (31,19) BCH 부호에 대한 저 신뢰도 비트 반전 개수에 따른 BER 성능 비교
Fig. 8. BER performance comparison of the (31,19) BCH code according to the number of bit inversions.

량이 많은 신드롬 계산이 최대 2회 더 필요하고, LUT를 참조하여 오류를 수정하는 과정도 최대 2회 더 많이 실행된다. 두 가지 방법의 BER 성능 차이는 거의 없으며, 특히 E_b/N_0 값이 커지면 BER 성능 차이는 더 줄어드는 것을 알 수 있다.

위에서 적용한 방법들은 모두 신뢰도가 낮은 비트들을 순차적으로 정렬하는 작업이 필요하다. 실제로 계산된 신드롬이 적절한 오류위치를 지정해 주지 못할 경우, 부호어 내에서의 상대적인 신뢰도 값의 순서가 중요한 것이 아니라 절대적으로 신뢰도 값이 어느 정도인지가 신드롬을 재계산할 필요가 있는지를 판단하는 더 적절한 기준이 될 수 있다. 아래 그림 10은 E_b/N_0가 10 dB 이하인 환경에서 수신 부호어별 신뢰

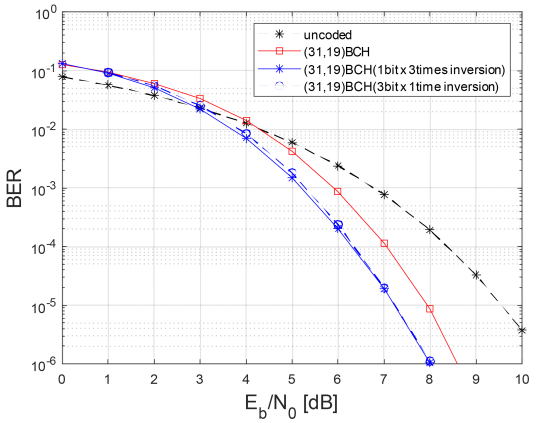

그림 9. 1비트씩 3회 반전한 방식과 1회에 3비트 반전한 방식에 대한 BER 성능 비교
Fig. 9. BER performance comparison with three times of bit by bit inversion and a single time three bit inversion.

도 문턱 값, *CL* 이하에 해당되는 평균 비트의 수를 조사한 결과를 보여준다. 그림 10의 결과를 이용하면, 적절한 *CL* 값을 선택함으로써, 특별한 신뢰도 정렬 과정 없이 특정 개수의 저신뢰도 비트를 선택할 수 있다. 예를 들어, (31,19) BCH 부호의 경우 부호어당 신뢰도 값이 0.2 보다 적인 비트의 개수가 평균적으로 3비트가 된다는 의미이다. 따라서 이 경우 *CL* 값을 0.2로 선택하면 평균적으로 3비트의 저신뢰도 비트가 선택된다는 의미이기도 하다.

위와 같은 결과를 참조하면, 저신뢰도 비트를 선택하기 위해서 신뢰도 문턱 값을 적절하게 설정하는 것은 비트 정렬에 필요한 연산을 줄이는 좋은 방법이 될 수 있음을 알 수 있다. 아래 그림 11은 *CL* 값을 변화시키면서 선택된 저신뢰도 비트를 정렬 과정 없이 모두 한꺼번에 반전한 후 재복호한 성능을 보여준다. 저신뢰도 비트를 정렬해서 순차적으로 반전하지 않고 한 번에 모두 반전시키면 비트를 정렬하는 과정을 생략할 수 있어 연산량을 줄일 수 있는 장점이 있다. *CL* 값을 0.1로 설정 할 경우, 위 그림 8의에서 1~2비트를 반전시킨 결과와 유사한 성능을 보이고 있고, 0.2로 설정 한 경우 3비트를 반전시킨 경우와 유사한 성능을 보여줌으로써, 그림 11의 분석 결과와 정확하게 일치한다는 것을 알 수 있다. 그림 11의 결과로 볼 때, 저신뢰도 비트를 선별하기 위한 정렬 과정 없이, 적절한 *CL* 값을 적용하여 매우 적은 복잡도로 효과적인 성능향상을 얻을 수 있음을 알 수 있다.

본 논문에서 제안한 복호 방법은 신드롬 계산이 완료되면 사전에 구현된 LUT를 이용하여 매우 쉽게 오류를 정정할 수 있는 장점이 있다. 표 3에는 제안된 방법의 계산 복잡도를 기존 BMA와 Chien search를 이용한 방법과 비교하여 나타내었다. 기존 방법의 경우, t^2에 비례하는 복잡도를 가지는 오류위치 다항식 생성 과정과 최대 $t \times n$회의 곱셈이 필요한 Chien search 과정이 필요하지만, 제안 방법의 경우에는 이 모든 과정을 사전에 구현된 LUT로 대체할 수 있다. 또한 성능 개선을 위해서 반복 복호를 수행할 때도 신드롬 계산에 필요한 연산만 추가되기 때문에 기존의 BMA와 Chien search를 이용한 복호 방법과 비교할 때 연산량을 줄일 수 있는 장점이 있다.

저신뢰도 비트 몇 개를 선택하는 방법은 일반적으로 수신된 데이터의 신뢰도를 모두 계산해서 정렬하

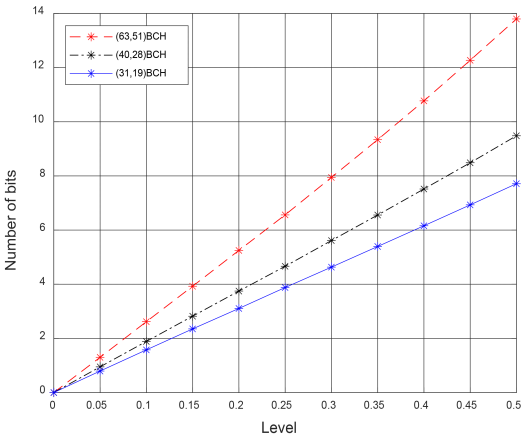

그림 10. 부호어당 신뢰도 문턱 값, CL 이하인 평균 비트의 개수
Fig. 10. Average number of bits with reliability less than CL in a codeword.

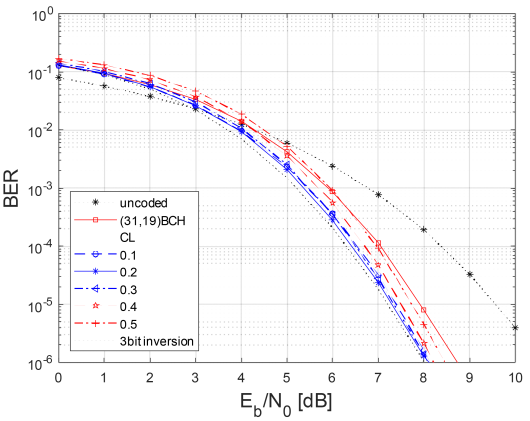

그림 11. (31,19) BCH 부호에 CL을 이용한 비트 반전에 따른 BER 성능
Fig. 11. BER Performance of the (31,19) BCH code according to the bit inversion using CL.

표 3. 복호 수행시 계산 복잡도 비교
Table 3. Computational complexity comparison of the decoding processes.

Decoding process	Proposed method	Conventional method (BMA - Chien search)
(1) syndrome calculation	the same complexity	
(2) error-locator polynomial	no need	BMA complexity: $O(t^2)$
(3) error location search	no need (use LUT)	Chien search complexity: $O(tn)$
Repeat decoding for performance improvement	Requiring only (1)	Requiring all (1)-(3)

는 알고리즘이 필요하다. 그러나 본 논문에서 제안한 신뢰도 문턱 값(*CL*)을 적용하면 복잡한 정렬 알고리즘을 사용하지 않고도 필요한 저신뢰도 비트를 선택할 수 있다. 선택된 저신뢰도 비트를 반전시킨 후 본 논문에서 제안한 복호 기법을 적용하면 연산량을 크게 줄이면서 복호 성능을 개선시킬 수 있다.

V. 결 론

본 논문에서는 2비트 오류정정이 가능한 BCH 부호에 대한 효율적인 부복호기 설계 방법을 제안하였다. 본 논문에서 제안한 방식은 SW를 기반으로 한 부복호기를 구현할 때, 복잡한 연산과정 대신에 매우 간단한 LUT를 이용하여 구현될 수 있도록 한 방식으로 기존의 방식의 부복호 과정에서 요구되는 연산량을 대폭 줄일 수 있는 장점이 있다. 제안된 방식은 메시지 다항식을 생성다항식으로 나눈 나눗셈-나머지 LUT를 이용한 부호화 방법과 신드롬과 오류위치를 미리 계산한 신드롬-오류위치 LUT를 활용한 복호 방법으로 구성되어 있다. 또한 추가적인 복잡도의 증가가 거의 없이, 수신된 데이터 중에서 신뢰도가 낮은 비트를 선별하여 그 값을 반전시킨 후 다시 복호하는 과정을 통하여 기존의 복호 방법으로는 수정할 수 없는 오류도 추가적으로 수정할 수 있는 방법을 제안하였다. 제안된 방법은 본 논문에서 다룬 (63,51) BCH 부호에 국한되지 않고 다른 규격의 BCH 부호에도 유사한 방식으로 적용할 수 있다. 본 논문에서 제안한 BCH 부호의 부복호기와 신뢰도가 낮은 비트를 반전시켜 다시 복호하는 방법은 저전력 저속으로 동작하는 처리장치에서도 소프트웨어적으로 구현하기 쉬울 뿐만 아니라 부복호를 효율적으로 수행할 수 있고 오류 정정 효율도 더 높일 수 있을 것으로 예상된다.

References

[1] Elwyn R. Berlekamp, *Algebraic Coding Theory*(Revised 2015), World Scientific, 2015.

[2] IEEE Std 802.15.6, *IEEE standard for local and metropolitan area networks -part 15. 6: Wireless body area networks*, 2012.

[3] T. W. Yang, B. Shin, S. Lee, and H.-J. Oh, "Design of architecture for (63,51)BCH encoder and decoder," in *Proc. KICS Summer Conf.*, vol. 60, pp. 1358-1359, Jun. 2016.

[4] Arul K. Subbiah and T. Ogunfunmi, "Area-efficient re-encoding scheme for NAND flash memory with multimode BCH error correction," *2018 IEEE ISCAS*, Florence, Italy, May 2018.

[5] H. Kim and S.-R. Kim, "Performance analysis of various coding schemes for storage systems," *J. KICS*, vol. 33, no. 12, pp. 1014-1020, Dec. 2008.

[6] M. Prashanthi, Damarla Paradhasaradhi, and N. Vivek, "An advanced low complexity double error correction of an BCH decoder," *2014 Int. Conf. Green Computing Commun. and Electrical Eng.*, Coimbatore, India, Mar. 2014.

[7] D. Kim, I. Yoo, and I.-C. Park, "Fast low-complexity triple-error- correcting BCH decoding architecture," *IEEE Trans. Cir. and Syst. II*, vol. 65. pp. 764-768, 2018.

[8] C.-H. Yang, T.-Y. Huang, and Y.-L. Ueng, "A 5.4μW soft-decision BCH decoder for wireless body area networks," *IEEE Trans. Cir. and Syst.*, vol. 61, pp. 2721-2729, Sep. 2014.

[9] S.-H. Jeong and B. Moon, "Comparison of error locator polynomial calculation algorithms of BCH code for In-DRAM ECC implementation," in *Proc. KICS Summer Conf.*, pp. 201-201, Jun. 2018.

[10] Stephen B. Wicker, *Error Control Systems for Digital Communication and Storage*, Prentice Hall, ch.5, 1995.

이 종 훈 (Jonghoon Lee)

1996년 2월 : 전북대학교 전자공학과 학사
1998년 2월 : 전북대학교 전자공학과 석사
1999년 3월~현재 : 전북대학교 전자공학과 박사과정
<관심분야> 오류정정부호, 초고속 무선 LAN
[ORCID:0000-0002-8653-6310]

김 수 영 (Sooyoung Kim)

1990년 2월 : 한국과학기술원 전기 및 전자공학과 학사
1990년 2월~1991년 9월 : ETRI 연구원
1992년 10월 : Univ. of Surrey, U.K 공학석사
1995년 2월 : Univ. of Surrey, U.K 공학박사
1994년 11월~1996년 6월 : Research Fellow, Univ. of Surrey, U.K
1996년 8월~2004년 2월 : ETRI 광대역무선전송연구팀장
2004년 3월~현재 : 전북대학교 전자공학부 교수
<관심분야> 오류정정부호화방식, coded MMO, 이동/위성통신
[ORCID:0000-0003-0817-2790]

송 상 섭 (Sangseob Song)

1978년 2월 : 전북대학교 전기공학과 학사
1980년 2월 : KAIST 전기 및 전자공학과 석사
1990년 2월 : Univ. of Manitoba, Electrical & Computer Engineering 공학박사
1981년 3월~현재 : 전북대학교 전자공학부 교수
<관심분야> 부호이론, 오류정정부호
[ORCID:0000-0002-7228-0713]