http://dx.doi.org/10.5392/JKCA.2016.16.03.091

MLC 낸드 플래시 메모리 오류정정을 위한 고속 병렬 BCH 복호기 설계

Design of High-performance Parallel BCH Decoder for Error Collection in MLC Flash Memory

최원정*, 이제훈**, 성원기**
강원대학교 공학대학 전자정보통신공학과*, 강원대학교 공학대학 전자정보통신공학부**

Won-Jung Choi(che1989@naver.com)*, Je-Hoon Lee(jehoon.lee@kangwon.ac.kr)**,
Won-Ki Sung(sungwk@kangwon.ac.kr)**

요약

본 논문은 MLC 타입 낸드 플래시 메모리의 오류 정정을 위한 병렬 BCH 복호기 설계를 제안한다. 제안된 BCH 복호기는 다중 바이트 병렬 연산을 지원한다. 병렬 계수 증가에 따른 회로 크기 증가폭을 줄이기 위해, LFSR 기반 병렬 신드롬 생성기 구조를 적용하였다. 제안된 BCH 복호기는 VHDL을 이용하여 합성되었고, Xilinx FPGA를 이용하여 동작을 검증하였다. 검증 결과 제안된 신드롬 생성기는 기존 바이트-단위의 병렬 신드롬 생성기에 비해 성능을 2.4배 증가시켰다. GFM 방식의 병렬 신드롬 생성기와 비교하여, 동작 완료에 따른 사이클 수는 동일하나, 회로 크기는 1/3 이하로 감소됨을 확인하였다.

■ 중심어 : | 에러 정정 | BCH 코드 | 병렬 처리 | 신드롬 생성기 |

Abstract

This paper presents the design of new parallel BCH decoder for MLC NAND flash memory. The proposed decoder supports the multi-byte parallel operations to enhance its throughput. In addition, it employs a LFSR-based parallel syndrome generator for compact hardware design. The proposed BCH decoder is synthesized with hardware description language, VHDL and it is verified using Xilinx FPGA board. From the simulation results, the proposed BCH decoder enhances the throughput by 2.4 times than its predecessor employing byte-wise parallel operation. Compared to the other counterpart employing a GFM-based parallel syndrome generator, the proposed BCH decoder requires the same number of cycles to complete the given works but the circuit size is reduced to less than one-third.

■ keyword : | Error Correction | BCH Code | Parallel Processing | Syndrome Generator |

I. 서 론

낸드 플래시 메모리는 비휘발성 반도체 메모리로, 빠른 읽기 및 쓰기 동작, 저전력, 충격에 강한 장점을 갖는다. 이러한 특징으로 인해, 낸드 플래시 메모리는 휴대용 메모리 디바이스와 스마트폰을 비롯한 휴대정보기

* 본 연구는 2014년 강원대학교 학술연구조성비로 연구하였음 (관리번호-220140070). 또한, 본 연구는 한국연구재단의 이공학개인기초연구지원사업으로 수행된 연구 결과임 (No. NRF-2015R1D1A1A01061237).
접수일자 : 2016년 01월 05일 심사완료일 : 2016년 02월 06일
수정일자 : 2016년 02월 01일 교신저자 : 성원기, e-mail : sungwk@kangwon.ac.kr

기의 내장형 메모리로 주로 사용되고 있다. 더 나아가, 한 셀에 여러 비트의 정보를 저장하는 멀티-레벨셀 (MLC, multi-level cell) 기술을 도입하여, 하드디스크를 대체할 정도로 저장 용량을 증가시키고 있다[1][2].

일반적인 메모리 소자는 각 셀당 한 비트를 저장하는 SLC (single level cell) 형태로 구현된다. MLC 구조는 한 개의 셀에 2 비트 이상의 정보를 저장하는 기술을 의미한다. 이는 한 개의 셀이 저장하고 있는 정보를 여러 개의 문턱 전압으로 나누어 각 논리값을 구분한다. 각 셀당 $q \geq 2$개의 비트를 바이진 심벌로 매핑하여 $M=2^q$ 레벨의 전압으로 구분한다[1]. 기존 한 셀당 2비트를 저장하는 MLC 타입에서, 한 셀당 3비트 또는 4비트를 저장할 수 있는 TLC (triple level cell) 또는 QLC (quad level cell)로 발전하고 있다.

MLC 낸드 플래시는 저장 용량은 크게 증가하나, 각종 잡음 및 인접 셀의 간섭에 취약해져 비트 오류가 발생한다. 각 셀당 저장하는 비트 수가 증가할수록 데이터의 안정성과 신뢰성이 낮아지게 된다. 이를 해결하기 위해, MLC 낸드 플래시는 오류 정정 부호 (ECC, error control code)를 채택한다. SLC 플래시 메모리의 경우 단순한 해밍 코드를 이용하여 오류 탐색 및 정정을 수행한다. 반면, MLC 타입 플래시 메모리는 비트 에러 수 증가로 BCH (Bose-Chaudhuri-Hochquenghem) 부호 혹은 RS (Reed-Solomon) 부호와 같이 강력한 오류 정정 부호를 사용한다. 최근 20nm 이하의 초미세공정을 이용한 TLC 혹은 QLC 낸드 플래시 메모리에 더욱 강력한 연판정 에러 정정 (soft-decision error correction)을 적용하려 시도하고 있다[2-4].

기존 낸드 플래시 메모리는 8비트 또는 16비트 인터페이스를 갖는다. 따라서, 8비트 혹은 16비트 폭의 양방향 데이터 버스를 통해 호스트와 낸드 메모리간 양방향으로 데이터가 전송된다[5]. 낸드 플래시용 BCH 복호기는 바이트 단위로 데이터를 수신하고, 이를 병렬로 처리하는 바이트 단위 (byte-wise) 병렬 BCH 복호기가 주로 이용된다[6]. 최근, 낸드 플래시의 읽기 및 쓰기 동작을 빠르게 하기 위해, 16, 32 또는 64비트로 확장된 인터페이스를 채택하고 있다. 따라서, BCH 복호기도 이에 대응하는 병렬 입력과 이를 동시에 처리하는 병렬 연산을 허용해야 한다. BCH 복호기의 병렬 계수를 늘릴수록 회로 크기 증가는 필연적이기 때문에, BCH 복호기의 병렬성을 높이더라도 회로 크기 증가율을 낮추는 회로 설계 기법이 요구된다.

본 연구는 MLC 낸드 플래시 메모리를 위한 병렬 BCH 복호기의 성능 향상 방법을 제안한다. 제안된 BCH 복호기는 여러 바이트의 병렬 입력을 허용하고, 이를 동시에 처리하도록 설계하였다. 병렬성 증대에 따른 회로 크기 증가를 줄이기 위해 일반적인 GFM (Galois filed multiplication) 기반 병렬 신드롬 생성기 대신, 기제안된 바이트 병렬 LFSR (linear field shift register) 기반 신드롬 생성기를 다중 바이트로 확장한 새로운 신드롬 생성기 구조를 제안한다. 제안된 병렬 BCH 복호기는 동일한 병렬 계수를 갖는 기존 BCH 복호기에 비해 적은 회로 크기로 구성 가능하고 보다 높은 클럭 속도로 동작한다.

본 논문은 다음과 같이 구성된다. Ⅱ장에서는 기존의 BCH 복호기에 대해 기술한다. Ⅲ장에서는 제안하는 병렬 BCH (4122,4096,2) 복호기 구조와 q-병렬 신드롬 생성기에 대해 기술한다. Ⅳ장에서는 제안된 BCH 복호기를 RT (register-transfer) 레벨로 합성한 후 실험을 통해 동작 속도와 회로 크기 비교를 기술하고, 마지막으로 Ⅴ장에서 결론을 맺는다.

Ⅱ. 낸드 플래시 메모리 및 BCH 부호

낸드 플래시 메모리는 [그림 1]과 같이 구성된다. 호스트는 컨트롤러를 거쳐 낸드 어레이와 데이터를 송수신한다. 호스트 데이터는 양방향 버스를 통해 메모리에 연결되며, 8비트 혹은 16비트 버스폭을 갖는다. 16비트 모드인 경우, 명령과 어드레스 전송을 위해 하위 8비트를 사용하고, 상위 8비트는 읽기 및 쓰기 동작을 위해 데이터 전달 사이클 동안 이용된다. 낸드 플래시 메모리는 데이터 무결성을 보장하기 위해 ECC를 사용하며, ECC는 하드웨어 혹은 소프트웨어로 구현될 수 있으나, 동작 속도면에서 하드웨어 구현 방식이 유리하다.

호스트 시스템은 페이지 단위로 낸드 메모리의 읽기

및 쓰기 동작을 수행한다. [그림 2]에 나타낸 것처럼, 각 페이지는 여러 개의 섹터를 포함한다. 일례로, 4Gbit 낸드 디바이스는 2,048 블록으로 구성되며, 한 블록에는 64 페이지가 있다. 각 페이지는 4,224 바이트를 갖고, 4096 바이트의 데이터 영역과 128 바이트의 여분의 영역으로 구성된다. 각 페이지는 512 바이트 길이의 섹터들을 포함하며, 섹터별로 오류 정정 코드를 적용하며, 생성된 패리티비트들은 여분의 영역에 저장된다.

낸드 메모리의 읽기 및 쓰기 동작 속도가 빠르고, 한 번에 섹터 단위로 처리하므로, 고속으로 오류 정정 부호의 인코딩 및 디코딩을 수행해야 한다[7]. 특히, 호스트 인터페이스가 지원하는 데이터버스의 전송폭에 해당하는 데이터를 병렬로 수신하고, 이를 한 번에 처리할 수 있도록 높은 병렬성을 가진 BCH 복호기가 요구된다.

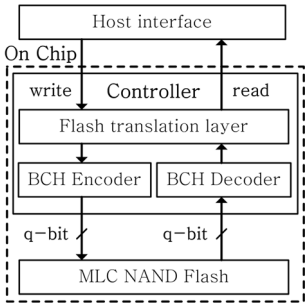


그림 1. NAND 플레쉬 메모리 블록 다이어그램

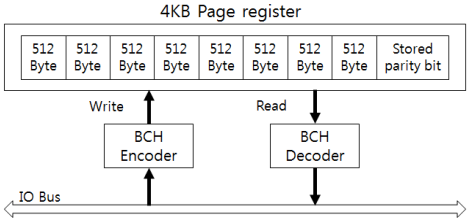


그림 2. MLC NAND 플래시 메모리의 데이터 입출력

BCH 코드는 잘 알려진 순환 오류 정정 부호로, 랜덤 오류 수정에 주로 사용된다. Galois 필드는 기호 GF(2^m)으로 표시되며 최대 2^m-1 길이의 코드워드를 생성한다. BCH 코드는 BCH (n,k,t)로 표기되며, n은 코드워드 길이, k는 메시지 길이, t는 오류 정정 가능 비트 수를 나타낸다. 생성된 패리티비트 수는 (n-k)가 된다[8]. 본 논문에서는 최대 코드 워드의 길이를 메시지 및 정정 가능한 오류의 비트 수에 따르는 Shortened BCH를 사용하였다. 코드워드 길이는 n-[k+(m×t)]로 표기되며, 512 바이트의 섹터별로 복호화하기 위해, BCH (4122,4096,2)가 사용된다[8].

일반적인 BCH 복호기는 [그림 3]과 같이 구성되며, 신드롬 생성기 (syndrome generator), Key equation solver, Chien search 및 오류 정정 (error correction) 등 주요 구성 요소를 갖는다. 수신된 코드워드는 직렬로 BCH 복호기로 입력된다. BCH 복호기는 수신된 코드워드를 통해 오류 위치를 계산하고, 해당 위치의 논리값을 정정한다. 특히, 오류 위치 다항식을 생성하는 신드롬 생성기와 오류 위치를 결정하는 Chien search의 경우 비트-시리얼 (bit-serial) 방식으로 연산할 경우 코드워드의 길이만큼의 지연시간이 발생한다. 결국, 직렬 방식의 BCH 복호기는 회로 크기는 작지만 코드워드의 길이가 길수록 복호화에 필요한 연산 시간이 길다.

고속으로 BCH 복호를 수행하기 위해서는 매 클럭 n개의 다중 비트의 코드워드를 수신하고, 신드롬 생성기 및 Chien search의 경우 병렬로 이를 처리할 수 있어야 한다. 이러한 문제를 해결하기 위해 다양한 연구가 수행되고 있다. Y. Chen은 long BCH 코드를 위한 병렬 Chien search 구조를 제안하였다. 특히, 중복되는 연산을 공유하는 그룹 매칭 기법을 이용하여 회로 크기를 줄이는 방법을 제안하였다[9]. Y. Lee는 병렬 신드롬 생성기 구조를 제안하였고, 특히 하드웨어 복잡도를 줄이기 위한 새로운 최적화 기법을 제안하였다[10].

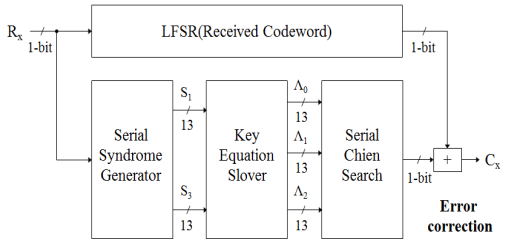


그림 3. 일반적인 직렬 BCH 복호기 구조

Ⅲ. 제안한 병렬 BCH 복호기

본 논문은 낸드 메모리를 위한 고속 병렬 BCH 복호기 구조를 제안한다. 신드롬 생성기의 병렬 계수 증가에 따른 회로 크기 증가폭을 줄이기 위해 기제안된 바이트 단위의 LFSR 기반 병렬 신드롬 생성기에 리프-어헤드 (leap-ahead) 기법을 적용하였다. 이를 통해, 2, 4 또는 8 바이트 이상의 병렬 연산이 가능한 LFSR 기반 병렬 신드롬 생성기 구조를 제안한다. 제안된 병렬 BCH 복호기는 16, 32, 또는 그 이상으로 확장된 인터페이스를 갖는 MLC 플래시 메모리와 글루없이 연결가능하다.

3.1 q-비트 병렬 BCH 복호기

제안한 병렬 BCH 복호기는 [그림 4]와 같이 매 클럭마다 q-비트 데이터 워드를 동시에 수신하고, 이를 병렬로 연산한다. 낸드 플래시의 섹터는 패리티피트를 포함하여 총 4,122 비트가 된다. 병렬 계수가 q인 신드롬 생성기와 Chien search 블록은 수신된 코드워드의 연산을 마치기 위해 각각 ⌈4122/q⌉ 클럭이 필요하다. 병렬 계수, q가 높아질수록 처리 속도가 빨라지기 때문에, 높은 병렬 계수를 갖는 신드롬 생성기와 Chien search 블록은 고속 BCH 복호기 설계에 필수적이다.

일례로, [그림 4]에 나타낸 32-병렬 BCH 복호기는 32 비트 병렬 데이터 전송을 허용한다. 수신된 코드워드, Rx는 매 클럭마다 32비트씩 신드롬 생성기로 전송된다. 신드롬 생성기는 이를 32-비트 단위로 병렬 연산하며, 4,122 비트 길이의 코드워드 연산에 ⌈4122/32⌉, 즉 129 클럭이 필요하다. 신드롬 생성기는 두 개의 신드롬, S_1과 S_3를 생성한 후, 오류 위치 다항식의 원소를 개산하기 위해 Key equation solver로 전송한다. 오류 위치 다항식의 원소인 13비트 길이의 Λ_0, Λ_1 및 Λ_2가 Chien search로 전송된 후, Chien search는 오류 위치 다항식을 32-비트 병렬로 연산하며, 129 클럭 동안 32-비트의 오류 정정 신호를 출력한다. 오류 정정 신호는 LFSR을 통해 지연된 데이터와 XOR 연산을 통해, 오류가 발생한 위치의 비트값을 반전시켜 오류를 정정한다.

32-병렬 BCH 복호기는 32 비트 단위의 데이터 전송을 허용하고, 신드롬 생성기와 Chien search의 32-병렬 연산을 지원함으로써 직렬 BCH 복호기보다 연산에 필요한 클럭 수를 ⌈1/32⌉로 줄일 수 있다. 32-병렬 뿐만 아니라 16 및 64등 플래시 메모리 컨트롤러의 전송폭과 동일한 병렬 계수를 갖는 신드롬 생성기 및 Chien search 블록을 구성 가능하다.

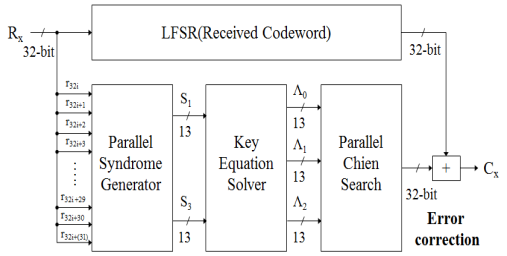


그림 4. 제안된 32-병렬 BCH 복호기 구조

3.2 기존 병렬 신드롬 생성기

신드롬 생성기는 오류위치다항식의 해가 되는 신드롬을 생성한다. 수신된 코드워드, Rx를 최소다항식 m(x)로 나눈 나머지는 오류 위치에 따라서 각각 일정한 독립적인 데이터를 출력하며, 이를 신드롬이라 한다. 오류 위치에 따라서 일정한 값이 발생되며, 이를 통해 오류의 위치를 결정하고 정정할 수 있다.

신드롬 생성기는 오류 정정 가능 비트 수 t에 따라서 2t 개의 신드롬을 생성한다. BCH (4122,4096,2)는 2개의 오류를 수정하므로, S_1, S_2, S_3, S_4 등 4개의 신드롬이 필요하다. 이를 계산하기 위해 식 (1)과 같이 4개의 최소다항식이 필요하다.

$$
\begin{aligned}
m_1(x) &= 1+x+x^3+x^4+x^{13} \\
m_2(x) &= 1+x+x^3+x^4+x^{13} \\
m_3(x) &= 1+x+x^3+x^4+x^5+x^7+x^9+x^{10}+x^{13} \\
m_4(x) &= 1+x+x^3+x^4+x^{13}
\end{aligned} \tag{1}
$$

4개의 최소다항식은 각각 α, α^2, α^3, α^4의 근을 갖는다. 이 때, $m_1(x)$, $m_2(x)$, $m_4(x)$는 최소다항식은 같고 근의 각각 α의 2제곱 및 4제곱 형태이므로 $m_1(x)$을 사용하여 S_1을 연산한 후, S_2, S_4 신드롬은 $S_2=(S_1)^2$, $S_4=(S_2)^2$로 생

성한다. 결국, S_1과 S_3 연산만이 필요하며, 식 (2)와 같이, 수신된 코드워드 r(x)을 최소다항식 $m_1(x)$, $m_3(x)$으로 나누었을 때, 나머지인 $R_1(x)$과 $R_3(x)$로 얻어지며, 이는 식 (3)과 같다.

$$S_1(x) = R_1(x) = \frac{r(x)}{m_1(x)},\ S_3(x) = R_3(x) = \frac{r(x)}{m_3(x)} \quad (2)$$

$$S_1(i) = \sum_{i=0}^{n-1} r_i\alpha^i = r_0\alpha^0 + r_1\alpha^1 + r_2\alpha^2 + \ldots + r_{n-2}\alpha^{n-2} + r_{n-1}\alpha^{n-1} \quad (3)$$

$$S_3(i) = \sum_{i=0}^{n-1} r_i\alpha^i = r_0\alpha^0 + r_1\alpha^3 + r_2\alpha^6 + \ldots + r_{n-2}\alpha^{3(n-2)} + r_{n-1}\alpha^{3(n-1)}$$

GF(2^{13})을 구성하는 2^{13}개의 GF 원소를 벡터로 표기하면 식 (4)와 같다. 유한체에서의 연산을 위해 식 (5)와 같이 다항식으로 나타내며, 이 때, x를 α로 치환하면 식 (6)과 같다.

$$\beta = b_0 b_1 b_2 \ldots b_{12} \quad (4)$$

$$\beta(x) = b_0 + b_1 x + b_2 x^2 + b_2 x^3 + b_2 x^4 + \ldots b_{12} x^{12} \quad (5)$$

$$\beta(\alpha) = b_0 + b_1\alpha + b_2\alpha^2 + b_2\alpha^3 + b_2\alpha^4 + \ldots b_{12}\alpha^{12} \quad (6)$$

식 (6)의 양변에 α를 곱한 후, 식 (7)에 나타낸 것처럼, α^{13}을 치환하여 같은 차수의 α로 정리하면 식 (8)과 같다. 회로 구현시 덧셈은 XOR 연산으로 수행되므로, 식 (9)와 같이 표현된다.

$$\alpha^{13} = 1 + \alpha + \alpha^3 + \alpha^4 \quad (7)$$

$$\alpha\beta(\alpha) = b_{12} + (b_0 + b_{12})\alpha + b_1\alpha^2 + (b_2 + b_{12})\alpha^3 + (b_3 + b_{12})\alpha^4 + b_4\alpha^5 + \ldots + b_{11}\alpha^{12} \quad (8)$$

$$\alpha\beta(\alpha) = b_{12} + (b_0 \oplus b_{12})\alpha + b_1\alpha^2 + (b_2 \oplus b_{12})\alpha^3 + (b_3 \oplus b_{12})\alpha^4 + b_4\alpha^5 + \ldots + b_{11}\alpha^{12} \quad (9)$$

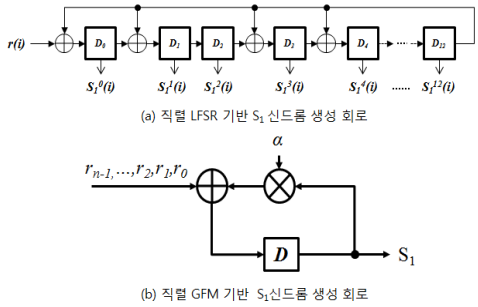
비트 단위로 입력되는 r(x)의 나눗셈 연산은 식 (10)과 같이 연산되며, 이를 통해 신드롬, S_1을 연산한다.

$$S_1(i+1) = (r(i) \oplus b_{12}) + (b_0 \oplus b_{12})\alpha + b_1\alpha^2 + (b_2 \oplus b_{12})\alpha^3 + (b_3 \oplus b_{12})\alpha^4 + b_4\alpha^5 + \ldots + b_{11}\alpha^{12} \quad (10)$$

S_3을 위한 α^3곱셈기 설계를 위해서는, 식 (6)의 β(α)에 α대신 α^3을 곱해주고 S_1을 연산한 것과 동일한 변환 과정을 거치면 식 (11)과 같다.

$$S_3(i+1) = (r(i) \oplus b_{10}) + (b_{10} \oplus b_{11})\alpha + (b_2 \oplus b_{12})\alpha^2 + (b_2 \oplus b_{12} \oplus b_0)\alpha^3 + (b_{10} \oplus b_1 \oplus b_{11})\alpha^4 + (b_{12} \oplus b_2 \oplus b_{11})\alpha^5 + (b_{12} \oplus b_3)\alpha^6 + b_4\alpha^7 + \ldots + b_9\alpha^{12} \quad (11)$$

식 (10) 및 식 (11)과 같이 표현된 신드롬 생성식은 [그림 5]와 같이 LFSR (linear feedback shift register) 혹은 GFM (Glouis field multiplication) 기반의 회로로 구현된다. [그림 5(a)]와 [그림 5(b)]는 각각 직렬 LFSR 기반 및 GFM 기반 S_1 신드롬 생성기를 나타낸다. LFSR 기반 신드롬 생성기는 13비트 레지스터와 4개의 XOR 연산으로 구현된다. GFM 기반 신드롬 생성기는 XOR, GFM 그리고 13비트 레지스터로 구성된다. 두 직렬 신드롬 생성기 모두 매 클럭당 1비트씩 코드워드를 수신하여 연산을 수행한다. 긴 데이터 블록을 오류 정정하기 위해서는 병렬 연산이 필요하다.



(a) 직렬 LFSR 기반 S_1 신드롬 생성 회로

(b) 직렬 GFM 기반 S_1신드롬 생성 회로

그림 5. 직렬 신드롬 생성기. (a) 직렬 LFSR 기반 S_1 신드롬 생성기, (b) 직렬 GFM 기반 S_1 신드롬 생성기

병렬로 신드롬을 생성하기 위해 [그림 6]과 같이 GFM 기반 신드롬 생성기가 주로 사용된다[11]. GFM을 입력 개수만큼 증가시켜 입력 비트별로 병렬로 연산한 후 합산하며, 이 과정을 코드워드가 모두 수신될 때까지 반복한다. q-병렬 GFM 기반 신드롬 생성기는 식 (12)와 같이 신드롬을 생성한다. 매 클럭마다 q-비트씩 병렬로 처리하기 때문에, 직렬 신드롬 생성기에 비해 요구 사이클의 수가 1/q배로 감소한다.

$$S_1(i) = \sum_{i=0}^{128}\left\{\sum_{i=0}^{128} r_{32i} + \sum_{i=0}^{128} r_{32i+1}\alpha + \ldots + \sum_{i=0}^{128} r_{32i+31}\alpha^{32i}\right\} \quad (12)$$

GFM 기반 병렬 신드롬 생성 회로는 병렬 계수가 증가함에 따라, GFM의 수도 이에 비례하여 증가되며, 덧셈기도 커지기 때문에 하드웨어 복잡도가 크게 증가한다. LFSR 구조의 신드롬 생성기는 쉬프트 레지스터와 XOR 연산기들만 필요하기 때문에 GFM 기반 신드롬 생성기보다 하드웨어 복잡도가 크게 감소된다. 그러나, LFSR 방식은 GFM 방식에 비해 불규칙적으로 연산되기 때문에 병렬 구조로 구성하기 어렵다.

이를 해결하기 위해, 본 연구의 선행 연구로, 바이트 단위의 병렬 LFSR 기반 신드롬 생성기를 [그림 7]과 같이 제안하였다[6]. 기제안된 구조는 쉬프트 레지스터에 저장된 초기값을 이용하여, 반복된 루프 연산을 언폴딩하여 연속적으로 수행한다. 따라서, 레지스터의 길이만큼 언폴딩이 가능하며, 기제안된 구조는 한 번에 최대 13 클럭까지 병렬로 연산 가능하다. 또한, 기제안된 바이트 병렬 구조는 13비트의 레지스터와 32개의 XOR 게이트로 구성되므로, 하드웨어 복잡도가 작다.

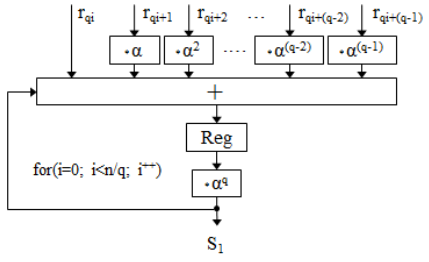


그림 6. q-병렬 GFM 기반 S_1신드롬 생성 회로

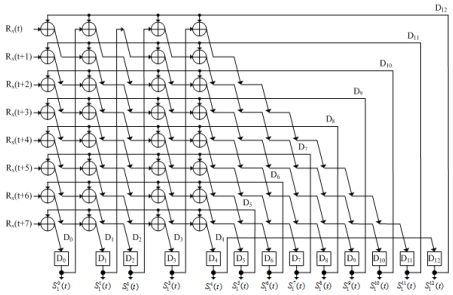


그림 7. 8-병렬 LFSR 기반 S_1 신드롬 생성 회로

BCH 복호기는 신드롬 생성기뿐만 아니라 Chien search 블록도 병렬 구조를 갖는다. Chien search 블록은 병렬 계수를 8뿐만 아니라 16, 32 혹은 그 이상으로 확장 가능하다. GFM 기반의 신드롬 생성기는 병렬 계수를 8뿐만 아니라 그 이상으로 쉽게 확장된다. BCH 복호기의 제어를 단순하게 만들기 위해 신드롬 생성기와 Chien search 블록의 병렬 계수를 동일하게 구성한다. 기제안된 바이트 단위의 병렬 신드롬 생성기는 최대 13 병렬까지만 확장 가능하기 때문에 이에 대응하기 어렵고, 이러한 이유로 GFM 기반 신드롬 생성기가 주로 사용된다.

3.3 제안한 LFSR 기반 병렬 신드롬 생성기

최근 낸드 메모리에 바이트 단위보다 확장된 16, 32 혹은 그 이상의 다중 바이트 전송을 지원한다. 높은 병렬 계수를 갖는 신드롬 생성기가 요구된다. 본 논문은 기존 바이트 단위 병렬 신드롬 생성기에 Leap-ahead 기법을 적용하여 2-바이트 이상의 동시 입력과 병렬 처리를 허용하는 새로운 신드롬 생성기를 제안한다.

일반적으로 LFSR 구조를 채택한 연산기는 식 (13)과 같이 표현된다.

$$X(t+1) = AX(t) \quad (13)$$

X(t)는 현 클럭에서 쉬프트 레지스터를 구성하는 모든 D 플립플롭의 출력이며, X(t+1)은 다음 클럭에서의 모든 D 플립플롭의 출력이다. A는 변환 매트릭스를 나타낸다. 만일 식 (13)을 m 회 반복적으로 연산하면, 모든 D 플립플롭의 m-cycle 이후 출력을 식 (14)와 같이 얻을 수 있다.

$$\begin{aligned} X(t+m) &= AX(t+m-1) \\ &= A(AX(t+m-2)) \\ &= \ldots \\ &= A^m X(t) \end{aligned} \quad (14)$$

식 (14)를 식 (15)와 같이 표현할 수 있다.

$$X'(t+1) = A^m X'(t) \quad (15)$$

만일 새로운 변환 매트릭스 A^m을 이용하여 LFSR 구조를 생성하면, 한 사이클만에 모든 D 플립플롭의 m-사이클 지연 출력, X'(t+1)을 얻을 수 있으며, 이를 Leap-ahead LFSR 구조라 칭한다.

본 논문은 기존 바이트 단위 병렬 신드롬 생성기에 Leap-ahead 기법을 적용하여 2-바이트 이상의 동시 입력과 병렬 처리를 허용하는 새로운 신드롬 생성기를 제안한다. LFSR 기반 신드롬 생성기의 병렬화를 위한 연산식 도출은 아래와 같다. 첫째, 직렬 방식의 신드롬은 식 (16)과 같고, LFSR에 저장되는 초기값은 식 (17)로 표현된다.

$$S_1(i+1) = (r(i) \oplus b_{12}) + (b_0 \oplus b_{12})\alpha + b_1\alpha^2 + (b_2 \oplus b_{12})\alpha^3 + (b_3 \oplus b_{12})\alpha^4 + b_4\alpha^5 + \ldots + b_{11}\alpha^{12} \quad (16)$$

$$S_1(initial) = b_0 + b_1\alpha + b_2\alpha^2 + b_3\alpha^3 + b_4\alpha^4 + \ldots + b_{12}\alpha^{12} \quad (17)$$

신드롬 $S_1(t)$는 t-번째 클럭에서의 신드롬 출력을 나타낸다. 첫번째 클럭 (t=1)에서 생성된 신드롬, $S_1(1)$은 식 (18)와 같이 출력되고, 동시에 이는 다음 클럭에서의 연산을 위해 LFSR에 회귀되어 식 (19)처럼 저장된다.

$$S_1(1) = (r(0) + b_{12}(0)) + (b_0(0) + b_{12}(0))\alpha + b_1(0)\alpha^2 + (b_2(0) + b_{12}(0))\alpha^3 + (b_3(0) + b_{12}(0))\alpha^4 + b_4(0)\alpha^5 + \ldots + b_{11}(0)\alpha^{12} \quad (18)$$

$$S_1(1) = b_0(1) + b_1(1)\alpha + b_2(1)\alpha^2 + b_3(1)\alpha^3 + b_4(1)\alpha^4 + \ldots + b_{12}(1)\alpha^{12} \quad (19)$$

t=2에서 생성된 신드롬은 식 (20)과 같다. 식 (20)을 t=0일 때의 LFSR 초기값을 이용하여 식 (21)과 같이 표현할 수 있다. 이를 식 (22)와 같이 일반화시키면, 한 클럭에 두 사이클 뒤의 출력을 얻을 수 있다.

$$S_1(2) = (r(1) + b_0(1)) + b_1(1)\alpha + b_2(1)\alpha^2 + b_3(1)\alpha^3 + b_4(1)\alpha^4 + \ldots + b_{12}(1)\alpha^{12} \quad (20)$$

$$S_1(2) = \alpha\beta = (Rx(1) + b_{11}(0)) + (Rx(0) + b_{12}(0) + b_{11}(0))\alpha + (b_0(0) + b_{12}(0))\alpha^2 + (b_1(0) + b_{11}(0))\alpha^3 + (b_2(0) + b_{12}(0) + b_{11}(0))\alpha^4 + (b_3(0) + b_2(0))\alpha^5 + b_4(0)\alpha^4 + b_5(0)\alpha^5 + \ldots + b_{11}(0)\alpha^{12} \quad (21)$$

$$S_1(t+2) = (Rx(t+1) + b_{11}(t)) + (Rx(t) + b_{12}(t) + b_{11}(t))\alpha + (b_0(t) + b_{12}(t))\alpha^2 + (b_1(t) + b_{11}(t))\alpha^3 + (b_2(t) + b_{12}(t) + b_{11}(t))\alpha^4 + (b_3(t) + b_{12}(t))\alpha^5 + b_4(t)\alpha^4 + b_5(t)\alpha^5 + \ldots + b_{10}(t)\alpha^{12} \quad (22)$$

식 (22)를 이용하여 BCH(4122,4096,2) 코드를 위한 신드롬 생성 규칙은 [그림 8(a)]와 같이 나타낼 수 있다. 즉, 시간 t에서의 LFSR 저장값을 이용하여 t+2 시간에서의 신드롬 출력, $S_1^0(t+2)$을 연산할 수 있다. 마찬가지 방법으로 [그림 8(b)]에 나타낸 것처럼 현재 클럭 t보다 8 클럭 이후의 신드롬 출력, $S_1^0(t+8)$을 구할 수 있다. 같은 방법으로, 현재 시간, t에서의 LFSR 저장값을 이용하여, t+16 혹은 t+32번째 클럭에서의 직렬 신드롬 생성기의 출력을 단지 한 클럭에 연산할 수 있다.

본 논문에서 제안한 병렬 신드롬 생성기는 기존 8-병렬 LFSR 기반 신드롬 생성기에 Leap-ahead 기법을 적용하여 이의 배수인 16, 32 혹은 그 이상으로 확장한다. 단지 한 클럭만에 16번째, 32번째 클럭에서의 출력을 모두 구할 수 있다. 제안된 신드롬 생성기는 허용되는 병렬 입력 비트 수에 따라 병렬 계수, q를 16, 32 및 64 이상으로 증가시켜, 512 바이트 데이터 블록의 신드롬 생성시 $\lceil 4122/q \rceil$ 사이클만을 필요로 한다.

$$\begin{bmatrix} S_1^0(t+2) \\ S_1^1(t+2) \\ S_1^2(t+2) \\ S_1^3(t+2) \\ S_1^4(t+2) \\ S_1^5(t+2) \\ S_1^6(t+2) \\ S_1^7(t+2) \\ S_1^8(t+2) \\ S_1^9(t+2) \\ S_1^{10}(t+2) \\ S_1^{11}(t+2) \\ S_1^{12}(t+2) \end{bmatrix} = \begin{bmatrix} R_x(t+1) + b_{11}(t) \\ R_x(t) + b_{12}(t) + b_{11}(t) \\ b_0(t) + b_{12}(t) \\ b_1(t) + b_{11}(t) \\ b_2(t) + b_{12}(t) + b_{11}(t) \\ b_3(t) + b_{12}(t) \\ b_4(t) \\ b_5(t) \\ b_6(t) \\ b_7(t) \\ b_8(t) \\ b_9(t) \\ b_{10}(t) \end{bmatrix} \qquad \begin{bmatrix} S_1^0(t+8) \\ S_1^1(t+8) \\ S_1^2(t+8) \\ S_1^3(t+8) \\ S_1^4(t+8) \\ S_1^5(t+8) \\ S_1^6(t+8) \\ S_1^7(t+8) \\ S_1^8(t+8) \\ S_1^9(t+8) \\ S_1^{10}(t+8) \\ S_1^{11}(t+8) \\ S_1^{12}(t+8) \end{bmatrix} = \begin{bmatrix} R_x(t+7) \oplus S_1^2(t) \\ R_x(t+6) \oplus S_1^2(t) \oplus S_1^6(t) \\ R_x(t+5) \oplus S_1^4(t) \oplus S_1^7(t) \\ R_x(t+4) \oplus S_1^2(t) \oplus S_1^7(t) \oplus S_1^8(t) \\ R_x(t+3) \oplus S_1^2(t) \oplus S_1^6(t) \oplus S_1^8(t) \oplus S_1^9(t) \\ R_x(t+2) \oplus S_1^6(t) \oplus S_1^7(t) \oplus S_1^8(t) \oplus S_1^{10}(t) \\ R_x(t+1) \oplus S_1^7(t) \oplus S_1^8(t) \oplus S_1^{10}(t) \oplus S_1^{11}(t) \\ R_x(t) \oplus S_1^8(t) \oplus S_1^9(t) \oplus S_1^{11}(t) \oplus S_1^{12}(t) \\ S_1^6(t) \oplus S_1^9(t) \oplus S_1^{10}(t) \oplus S_1^{12}(t) \\ S_1^1(t) \oplus S_1^{10}(t) \oplus S_1^{11}(t) \\ S_1^2(t) \oplus S_1^{11}(t) \oplus S_1^{12}(t) \\ S_1^3(t) \oplus S_1^{12}(t) \\ S_1^4(t) \end{bmatrix}$$

(a) 2-병렬 S_1 신드롬 생성 공식 (a) 8-병렬 S_1 신드롬 생성 공식

그림 8. 제안된 신드롬 생성식. (a) 2-병렬 S_1 신드롬 생성 공식, (b) 8-병렬 S_1 신드롬 생성 공식

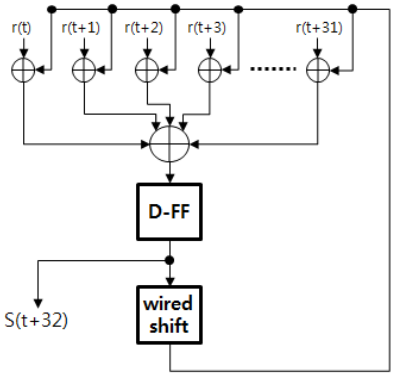


그림 9. 제안된 32-병렬 LFSR 기반 신드롬 생성기

[그림 9]는 제안된 병렬 신드롬 생성기 구조를 나타낸다. 제안된 신드롬 생성기는 매 클럭마다 32-비트의 코드워드를 수신하고, 32-병렬 신드롬 생성기로 연산한다. 병렬 계수가 증가되더라도 Galois 나눗셈 연산을 위한 XOR 연산의 수만 증가하며, 요구 사이클 수는 GFM 기반 신드롬 생성기와 동일하다.

Ⅳ. 시뮬레이션 결과

본 논문에서 제안한 BCH 복호기는 다중 바이트 병렬 연산이 가능한 신드롬 생성기 회로를 포함하여 구현되었다. 제안된 BCH 복호기는 VHDL을 이용하여 회로를 합성하고, Xilinx사의 Zynq XC7Z020 FPGA 개발 환경을 이용하여 동작을 검증하였다.

첫째, 제안된 다중 바이트 병렬 신드롬 생성기 회로의 동작 검증을 수행하였다. [그림 10]은 제안된 신드롬 생성기와 기존 직렬 및 8-병렬 신드롬 생성기와의 동작 비교 결과를 나타낸다. 512 바이트 길이의 섹터를 위한, 4,122비트의 코드워드를 수신한 후 직렬, 8-병렬, 16-병렬, 32-병렬로 연산하여 신드롬을 생성한다. q-병렬 신드롬 생성기는 직렬 신드롬 연산기보다 연산에 필요한 사이클이 ⌈1/q⌉ 만큼 감소한다. 병렬 계수가 8, 16 그리고 32인 병렬 신드롬 생성기는 직렬 방식과 비교하여 각각 ⌈1/8⌉, ⌈1/16⌉ 그리고 ⌈1/32⌉ 만큼 연산 사이클이 감소한다. [그림 10]과 같이, 직렬 신드롬 생성기

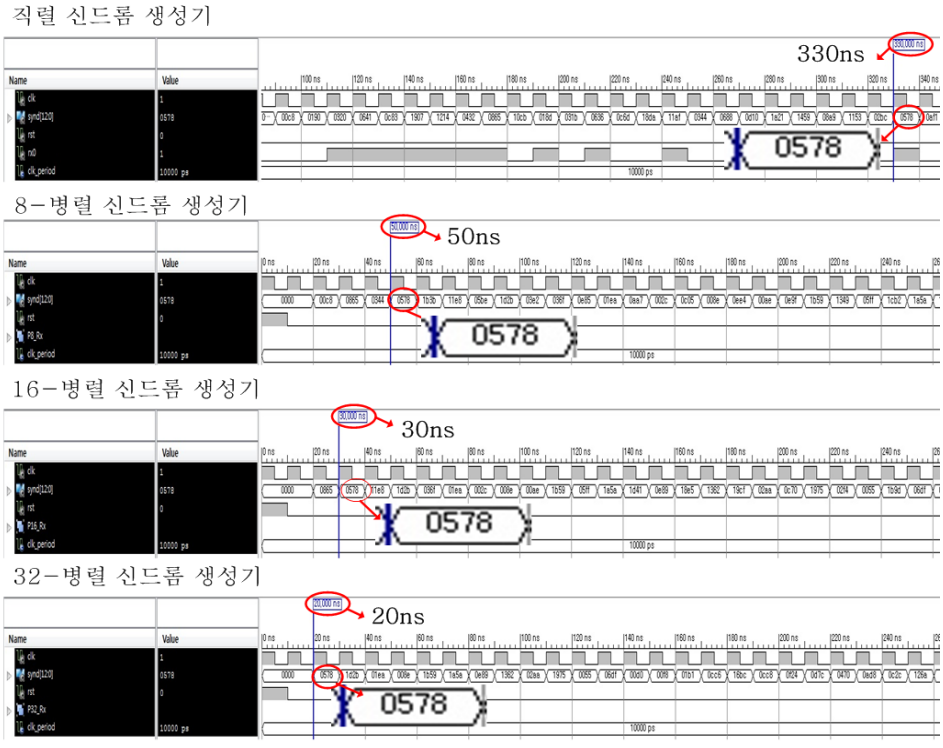


그림 10. Xilinx 환경에서 시뮬레이션 결과 파형

가 32 사이클 연산 후 나온 출력, 0578과 동일한 결과가, 8-병렬, 16-병렬, 그리고 32-병렬 신드롬 생성기는 각각 4, 2, 그리고 1 사이클에 얻어진다. 제안된 다중 바이트 병렬 신드롬 생성기는 여러 개의 바이트를 동시에 수신한 후, 이를 병렬로 처리 가능함을 확인하였다.

[표 1]은 제안된 신드롬 생성기를 이용하여, S_1 및 S_3 신드롬 생성시, 병렬 계수 증가에 따른 처리량 변화를 나타낸다. 일반 직렬 신드롬 생성기 S_1 및 S_3의 경우, 동작 주파수는 약 1.2GHz 및 1.0GHz에서 동작하며, 4,122개의 클럭이 필요하다. 8, 16, 그리고 32 병렬 신드롬 생성기의 요구 클럭 수는 각각 516, 258 및 129로 감소된다. 병렬 계수가 증가할수록, 신드롬 생성기내의 임계경로의 길이가 길어져 클럭 주파수의 주기가 길어진다. 일례로, S_1 신드롬 생성기의 경우, 8, 16 그리고 32병렬로 병렬 계수를 증가시키면, 동작주파수는 1.0GHz, 0.8GHz, 그리고 0.6GHz가 된다. [그림 11]은 병렬 계수 증가에 따른 성능 향상율을 나타낸다. 제안된 다중 바이트 신드롬 생성기는 2-바이트 및 4-바이트 병렬의 경우 약 10배 및 약 15-20배 성능 향상을 얻는다.

[표 2]는 제안한 다중 바이트 병렬 신드롬 생성기와 기존 GFM 기반 병렬 신드롬 생성기[11]와의 회로 크기 비교를 나타낸다. GFM 기반 신드롬 생성기는 병렬 계수의 t에 비례하여 2t-1개의 m-비트 덧셈기와 $2t^2$-t개의 곱셈기와 2t-1개의 레지스터가 추가된다. 제안한 신드롬 생성기는 병렬 계수 t가 증가하더라도 신드롬 연산에 필요한 XOR의 개수만 증가하고 레지스터의 개수가 m개로 고정된다. 따라서, 기존 GFM 기반 병렬 신드롬 생성기보다 회로 크기를 크게 줄일 수 있다.

[표 3]은 기존 신드롬 생성기와 제안한 신드롬 생성기의 회로 크기 비교를 나타낸다. 회로 크기 비교를 위해 제안된 복호기 설계를 SKHynix 0.18μm 공정을 이용하여 합성하였다. W. Liu가 0.25-μm 공정을 이용하여 합성한 결과, 직렬 신드롬 생성기의 크기는 $0.016mm^2$이고 8 병렬 Syn-chien의 경우 $0.091mm^2$의 회로 크기를 갖는다[11]. 기존 8-병렬 LFSR 기반 병렬 신드롬 생성기의 크기는 $0.022mm^2$ 로 제안한 신드롬 생성기와 동일한 크기를 갖는다 [6]. 이는 병렬 계수를 최대 13까지 확장된다는 단점을 갖는다. 본 논문에서 제안된 다중 바이트 병렬 신드롬 생성기는 32-병렬인 경우 회로 크기가 $0.030mm^2$로 구현된다. 따라서, GFM 기반 신드롬 생성기와 비교하여 회로 크기를 1/3 이하로 감소시켰다.

표 1. 제안한 병렬 LFSR 신드롬 생성기의 처리량 증가율

신드롬	병렬 계수	최대 동작주파수	사이클 수
기존 LFSR 기반 S_1 신드롬 생성기[6]	1	1.195Ghz	4,122
	8	0.992Ghz	516
제안한 LFSR 기반 S_1 병렬 신드롬 생성기	16	0.787Ghz	258
	32	0.600Ghz	129
기존 LFSR 기반 S_3 신드롬 생성기[6]	1	1.048Ghz	4,122
	8	0.685Ghz	516
제안한 LFSR 기반 S_3 병렬 신드롬 생성기	16	0.634Ghz	258
	32	0.637Ghz	129

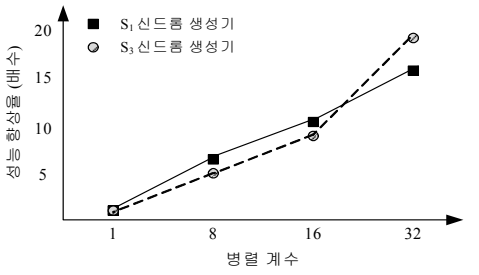


그림 11. 병렬 계수 증대에 따른 성능 향상율

표 2. 기존 GFM 기반 신드롬 생성기와의 회로 크기 비교

단위(개)	병렬계수	m-비트 덧셈기	곱셈기	XOR	레지스터
GFM 기반 병렬 신드롬 생성기[11]	1	1	1	0	1
	8	15	120	0	15
	16	31	496	0	31
	32	63	2016	0	63
제안한 LFSR 기반 병렬 신드롬 생성기	1	–	–	4	13
	8	–	–	32	13
	16	–	–	66	13
	32	–	–	128	13

[표 4]는 제안한 LFSR 기반 다중 바이트 병렬 신드롬 생성기를 채택한 BCH 복호기와 기존 MLC 낸드 플래시 메모리를 위한 BCH 복호기들과의 비교를 나타낸다. 제안된 32-병렬 신드롬 생성기를 채택한 BCH 복호기

를 설계하였다. 제안된 BCH 복호기는 8.6Gb/s의 처리량을 갖고, Y. Lee가 제안한 동일한 32병렬 BCH 복호기에 비해 1.34배 향상된 성능을 나타낸다. 또한, 동작 주파수면에 있어서, K. Lee가 제안한 BCH 복호기가 400MHz 클럭을 사용하는 반면, 제안된 BCH 복호기는 538MHz 클럭으로 1.35배 빠른 클럭을 사용한다. 제안한 BCH 복호기는 병렬 계수가 증가되더라도, 임계 경로의 길이가 크게 증가하지 않았음을 나타낸다.

표 3. 기존 신드롬 생성기와의 회로 크기 비교

신드롬 생성기	병렬계수	회로크기	공정
GFM 기반 병렬 신드롬 생성기[11]	1	0.016mm2	0.25μ m
	32	0.091mm2	
LFSR 기반 병렬 신드롬 생성기[6]	8	0.022mm2	0.18μ m
제안한 LFSR 기반 병렬 신드롬 생성기	8	0.022mm2	0.18μ m
	16	0.026mm2	
	32	0.030mm2	

표 4. 기존 BCH 복호기와 제안한 BCH 복호기 비교

병렬계수	[12]	[13]	[14]	[11]	제안한 BCH 복호기
공정(nm)	130	45	130	250	180
병렬계수	32	16	8	8	32
동작전압(v)	1.2	1.05	-	2.5	1.8
주파수(Mhz)	200	400	125	50	538
코드길이(bit)	8640	9088	4408	4148	4122
오류정정(bit)	32	64	24	4	2
처리량(Gb/s)	6.4	6.4	1	0.4	8.6

V. 결론

본 논문은 MLC 타입의 낸드 플래시 메모리를 위한 고속 병렬 BCH 복호기 구조를 제안한다. 제안된 BCH 복호기는 다중 바이트의 동시 입력을 허용하며, 이를 병렬로 처리한다. 기존 GFM 기반 병렬 신드롬 생성기 구조는 병렬 계수 증가에 따라 회로 크기가 크게 증가한다. 이를 해결하기 위해, GFM 대신 LFSR 기반의 신드롬 생성기 구조를 적용하였다. 기존 바이트 단위의 LFSR 기반 병렬 신드롬 생성기에 Leap-ahead 구조로 구성하여, 2개 혹은 4개 이상의 다중 바이트 입력을 동시에 처리하도록 구성하였다. 제안한 병렬 신드롬 생성기는 기존 GFM 기반 병렬 신드롬 생성기와 비교하여, 동작 속도가 1.3배 빠르며, 회로 크기가 1/3로 감소되었다. 또한, 병렬로 입력되는 다중 바이트 크기에 따라 신드롬 생성기 및 Chien search 블럭의 병렬 계수를 결정하여 전체적인 제어 구조를 단순하게 구성하였다. 따라서, 제안한 구조는 고속 읽기 및 쓰기 동작을 수행하는 다중 바이트의 데이터 버스 폭을 갖는 MLC 타입 플래시 메모리에 직접적으로 적용 가능하다. 또한, 병렬 계수 증가에 따른 회로 크기 증가를 최대한 억제함으로써, 경량 회로 구현이 가능하다. 제안한 병렬 BCH 복호기 구조는 MLC 타입의 낸드 플래시 메모리뿐만 아니라 고속 오류 정정이 필요한 다양한 어플리케이션에의 적용 가능할 것으로 기대된다.

참 고 문 헌

[1] 이동환, 성원용, “멀티 레벨 셀 낸드 플래시 메모리용 적응형 양자화기 설계,” 한국통신학회논문지, Vol.38C, No.6, pp.540-549, 2013(6).

[2] 김성래, 신동준, “멀티 레벨 낸드 플래시 메모리용 연판적 복호를 수행하는 이진 ECC 설계를 위한 EM 알고리즘,” 한국통신학회논문지, Vol.39A, No.3, pp.127-139, 2014(3).

[3] G. Dong, N. Xie, and T. Zhang, “On the use of soft-decision error correction codes in NAND flash memory,” IEEE Trans. Circuits Syst. I, Reg. Papers, Vol.58, No.2, pp.429-439, 2011(2).

[4] C. Yang, Y. Emre, and C. Chakrabarti, “Product code schemes for error correction in MLC NAND flash memories,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst., Vol.20, No.12, pp.2302-2314, 2012(12).

[5] J. Cooke, “Flash memory 101: An introduction to NAND flash,” EE Times., 2006(3). http://www.

eetimes.com/document.asp?doc_id=1272118
[6] 최원정, 이제훈, “센서네트워크 활용을 위한 경량 병렬 BCH 디코더 설계,” 한국센서학회논문지, Vol.24, No.3, pp.188-193, 2015(3).
[7] S. C. Jang, J. H. Lee, W. C. Lee, and K. R. Cho, “Design of a parallel BCH decoder for MLC memory,” Proc. ISOCC ’08, Vol.3, pp.46-47, 2008(11).
[8] S. Lin and D. J. Costello, *Error Control Coding, Upper Saddle River*, NJ: Prentice Hall, 2004.
[9] Y. Chen and K. K. Parhi, “Small Area Parallel Chien Search Architectures for Long BCH Codes,” IEEE Trans. Inform. Theory, Vol.12, No.5, pp.545-549, 2004(5).
[10] Y. J. Lee, H. Y. Yoo, and I. C. Park, “Small-area parallel syndrome calculation for strong BCH decoding,” Proc. of ICASSP 2012, pp.1609-1612, 2012(3).
[11] W. Liu, J. Rho, and W. Sung, “Low-power high-throughput BCH error correction VLSI design for multi-level cell NAND flash memories,” Proc. of SIPS, pp.303-308, 2006.
[12] Y. Lee, H. Yoo, and I. Park, “High- throughput and low-complexity BCH decoding architecture for solid-state drives,” IEEE T. VLSI, Vol.22, No.5, pp.1183-1187, 2013(1).
[13] K. Lee, S. Lim, and J. Kim, “Low-cost, low-power and high-throughput BCH decoder for NAND flash memory,” Proc. of ISCAS, pp.413 - 415, 2012(5).
[14] T. H. Chen, Y. Y. Hsiao, Y. T. Hsing, and C. W. Wu, “An adaptive-rate error correction scheme for NAND flash memory,” Proc. of 27th IEEE VLSI Test Symp., pp.53-58, 2009.

저 자 소 개

최 원 정(Won-Jung Choi) 준회원

- 2014년 2월 : 강원대학교 전자학 전공(공학사)
- 2014년 3월 ~ 현재 : 강원대학교 전자정보통신학과 석사과정

<관심분야> : 디지털 회로 설계, 암호 알고리즘, 회로 및 시스템

이 제 훈(Je-Hoon Lee) 종신회원

- 1998년 8월 : 충북대학교 정보통신공학과(공학사)
- 2001년 2월 : 충북대학교 정보통신공학과(공학석사)
- 2005년 2월 : 충북대학교 정보통신공학과(공학박사)
- 2006년 8월 ~ 2009년 8월 : 충북대학교 BK21충북정보기술사업단 초빙조교수
- 2009년 8월 ~ 현재 : 강원대학교 전자정보통신공학부 부교수

<관심분야> : 컴퓨터구조, 암호이론, 디지털시스템설계 및 임베디드시스템

성 원 기(Won-Ki Sung) 정회원

- 1980년 2월 : 중앙대학교 전자공학과(공학사)
- 1982년 2월 : 중앙대학교 대학원 전자공학과(공학석사)
- 1986년 8월 : 중앙대학교 대학원 전자공학과(공학박사)
- 1986년 3월 ~ 현재 : 강원대학교 전자정보통신공학부 교수

<관심분야> : 제어시스템, 디지털 회로설계