DAB 시스템에서 낮은 복잡도와 효율적인 구조를 갖는 FEC 설계

정회원 김 주 병*, 임 영 진**, 이 문 호***, 이 광 재****

FEC design with low complexity and efficient structure for DAB system

Joo-Byoung Kim*, Young-Jin Lim**, Moon-Ho Lee***, Gwang-Jae Lee**** *Regular Members*

요 약

본 논문에서는 DAB 시스템에서 사용하는 FEC(Forward Error Correction) 블록을 하드웨어 크기를 고려하여 효율적인 구조를 갖도록 설계하였다. DAB 시스템의 FEC블록은 크게 스크램블러(에너지분산), 리드-솔로몬 코더, 길쌈 인터리버로 구성된다. RS 디코더 블록 중 키 방정식을 계산해내는 블록과 길쌈 인터리버가 차지하는 하드웨어 비중은 굉장히 크다. 본 논문에서는 스크램블러 부분에서 데이터의 시작을 알려주는 신호의 효율적인 검출 기법을 제안하고, 리드-솔로몬 디코더 블록의 수정 유클리드 알고리즘을 효율적인 하드웨어로 구현하기 위한 새로운 구조와 길쌈 인터리버에서 최적의 메모리 구조를 가지는 효과적인 구조를 제안한다. 제안한 구조에서는 단지 8개의 GF 곱셈기와 4개의 덧셈기만을 가지고 RS 디코더의 수정 유클리드 알고리즘을 구현하였으며, 2 RAM(128)과 4 RAM(256)을 가지고 컨벌루셔널 인터리버를 구현하였다. 제안한 구조로 설계했을 경우 디코더 블록이 Altera-FPGA 칩(FLEX10K)에 모두 들어갈 수 있었다.

ABSTRACT

In this paper, because of FEC(Forward Error Correction) block hardware size in DAB system, is considered to have the benefit of efficient usage. FEC Block in DAB system is composed of Scrambler, Reed-Solomon coder, convolutional interleaver. In RS decoder block, there is a block that calculates Key Equation and convolutional interleaver but they take big part of hardware. In this paper, RS decoder's Key Equation block uses as Modified Euclid Algorithm, therefore, it uses less hardware space but better efficiency in performance. In convolutional interleaver block, recommended for maximum usage of ram with new optimal memory structure for convolutional interleaver. A proposed architecture has only 8 GF-multipliers and 4 GF-adders in RS coder, 2 RAM(128) and 4 RAM(256) in convolutional interleaver. We have designed FEC block for DAB system and have implemented on Altera-FPGA chip(FLEX10K) successfully.

I. 서 론

디지털 오디오 방송(DAB: Digital Audio Broad casting) 시스템은 현재의 AM 방송이나 FM 방송과는 전혀 다른 기술을 이용하여 고품질의 음질을 제공할 수 있으며, 이동체에서도 수신 능력이 강하고, 영상이나 문자와 같은 디지털 데이터를 고속으로

* 전북대학교 정보통신공학과 다차원이동통신연구실, 전북대학교 정보통신연구소(moonho@chonbuk.ac.kr)
** 전북대학교 영상공학과 다차원이동통신연구실(jbkim@chonbuk.ac.kr)
*** 전북대학교 정보통신공학과 다차원이동통신실(yjlim@chonbuk.ac.kr)
**** 한려대학교 정보통신공학과(kjlee@hanlyo.ac.kr)
논문번호 : 00148-0505, 접수일자 : 2000년 5월 5일

송신할 수 있는 특성을 가지고 있다. AM, FM 방송의 단점을 해소하고 고품질의 음질과 새로운 형태의 멀티미디어 서비스를 제공할 수 있는 차세대 방송 방식으로 인정받고 있는 DAB는 범세계적으로 상용화 시기가 임박하고 있다. 이러한 시기에 발맞추어 국내에서도 DAB의 실용화를 위한 연구개발이 시급한 실정이다.

본 논문에서는 이러한 차세대 DAB 시스템의 FEC(Forward Error Correction) 블록을 효율적인 구조를 갖도록 설계하였다. DAB 시스템에서는 동기 획득, 스크램블러, RS(Reed-Solomon) 코더, 길쌈 인터리버의 과정을 거쳐 길쌈 코더 등을 거치게 된다. 본 논문에서는 동기획득 방법과 스크램블러, RS코더, 길쌈 인터리버의 블록을 하나의 칩에 들어가도록 설계하였다. 인코더의 크기는 작으므로 큰 어려움 없이 설계가 가능하지만 디코더 블록에서는 RS 디코더의 큰 계산량과 길쌈 인터리버의 많은 메모리 사용으로 한 칩(FLEX10k)에 들어가도록 작업하기가 쉽지 않다. 특히 RS 디코더에서 키 방정식을 계산하는 블록과 길쌈 인터리버에서 사용되는 하드웨어 비중은 매우 크다. 하드웨어 부담을 최소로 하기 위해서는 RS 디코더의 블록과 길쌈 인터리버의 구조를 최적화시키는 과정이 반드시 필요하다.

본 논문에서는 동기획득을 위한 효율적인 구조와 스크램블러의 구조를 소개하고, RS 디코더의 키 방정식을 계산하는 수정 유클리드 알고리즘을 면적을 고려한 효율적인 구조로 제안하였으며, 최적의 메모리 구조를 갖도록 길쌈 인터리버를 설계하였다.

그림 1은 DAB 시스템에서 사용하는 송신단과 수신단의 FEC 블록도이다. 송신단의 스크램블링은 MPEG-2 데이터를 비트 스트림으로 받아서 수행하고, 외부 채널 부호로 RS 코더(204, 188, t=8)을 사용한다. RS 코더를 거친 데이터는 길쌈 바이트 인터리버를 통과하게 된다. 수신단의 FEC는 송신단의 역과정을 수행한다.

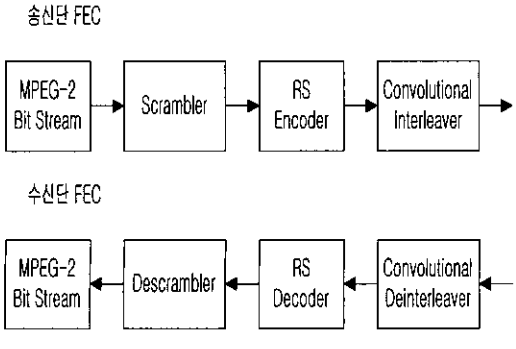


그림 1. 송신단과 수신단 FEC 블록도

설계 방법은 Altera에서 제공하는 Maxplus-2 Tool을 사용하여 VHDL 설계를 하였다. 또한 일반적으로 VHDL로 RAM을 구현하는 것보다 LPM 라이브러리를 사용하는 것이 효율적인 설계가 된다고 알려져 있다.

그러므로 여기에서는 Altera에서 제공하는 LPM (libary of parameterized modules)을 이용한다. LPM은 generic과 parameter를 조정함으로써 사용자가 원하는 RAM을 쉽게 설계할 수 있다.

2장에서는 동기 획득의 효율적인 방법을 소개하고 스크램블러의 동작과 그를 구현하기 위한 구조를 보여준다. 3장에서는 RS 코더의 동작을 소개하고, 특히 디코더의 수정 유클리드 알고리즘을 깊이 분석하고, 하드웨어 면적과 시간을 고려한 효율적인 새로운 구조를 제안한다. 4장에서는 길쌈 인터리버의 동작을 보여주고, 램을 이용하여 구현한 구조를 소개한다. 또한 최적의 구조를 만들기 위한 효과적인 구조를 제안한다. 5장에서는 본 논문의 결론을 맺는다.

Ⅱ. 스크램블러

스크램블러는 크게 두 가지로 나눌 수 있다. 동기획득과정과 PRBS(Pseudo Random Binary Sequence)과정이다.

먼저 MPEG-2 비트 스트림의 한 패킷은 그림 2의 (a)와 같다. 1 바이트의 SYNC(47_{HEX})와 187 바이트의 데이터로 구성되어진다. 이러한 패킷들이 연속해서 비트 스트림으로 밀려들어오기 때문에 스크램블링을 하기 위해서는 반드시 SYNC 신호의 시작점을 찾아야만 한다.

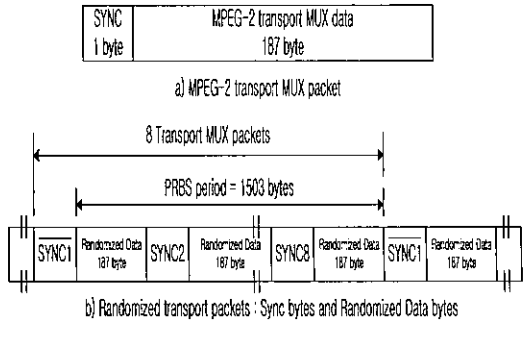


그림 2. MPEG-2 패킷과 에너지 분산된 패킷

SYNC 신호를 찾기 위해서 고려해야 할 것은 실제 데이터내에 SYNC 신호와 똑같은 데이터가 있을 수 있다는 것이다. 이러한 동작을 하기 위해서 여기에서는 쉬프트레지스터를 사용하는 방법을 소개한다. 쉬프트 레지스터를 통해서 한 비트마다 8개의 시작점을 검사한다. 각 시작점의 위치에서 1 바이트를 검사해서 만약 SYNC 신호라면 그 위치부터 카운트를 시작하는 방법을 사용한다. 만약 187 바이트 뒤에 같은 SYNC 신호가 나오면 또 다시 카운트를 시작해서 여러 번 SYNC신호를 검사한다. 즉 3까지 증가하는 카운터와 187까지 증가하는 카운터가 각각 8개씩 필요하게 된다. 그림 3은 쉬프트 레지스터를 이용해서 시작점을 검사하는 그림이다.

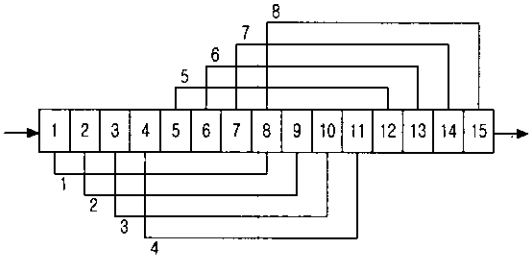


그림 3. 쉬프트 레지스터의 8가지 시작점 검사

동기획득은 SYNC신호를 찾는 것부터 시작된다. 위의 쉬프트 레지스터와 카운터를 이용해 시작점을 찾은 후 해야할 것은 수신단에서 정확한 동기를 맞춰주기 위해서 8 패킷마다 한번씩 SYNC 신호를 인버트 해준다. 즉 $SYNC(47_{HEX})$를 $\overline{SYNC}(B8_{HEX})$로 8번에 한번씩 바꿔주는 동작이 필요하다. 그 동작을 반복하면 그림 2의 (b)와 같은 구조가 된다.

SYNC 신호를 찾은 후에는 SYNC 신호를 제외한 나머지는 모두 Randomized 되어야만 한다. 이 과정에서 PRBS(pseudo random binary sequence)를 이용한다. 여기에서 사용하는 다항식은 $1+x^{14}+x^{15}$ 이다. PRBS 레지스터에서 처음 로드되는 시퀀스는 "100101010000000"이다. 즉 15비트의 쉬프트 레지스터를 사용한다.

그림 4는 PRBS 생성기의 구조를 나타낸다. 이것은 초기화되는 "100101010000000"을 가지고 1503 바이트에 해당하는 시퀀스를 발생시키게 된다. PRBS의 시작하는 위치는 첫 번째 SYNC 신호가 발생한 바로 다음 바이트부터이다. 그리고 그 다음 패킷마다 나오는 SYNC 신호에서는 PRBS 생성이 계속되지만 랜덤마이즈시키지 않는다. 이러

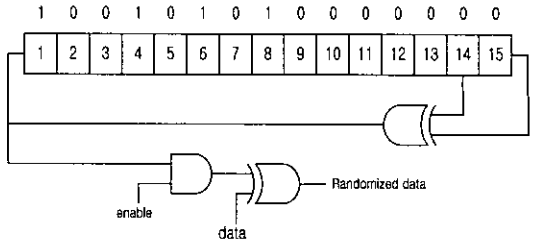


그림 4. PRBS 생성기

한 과정을 8 패킷마다 반복한다. 그러므로 주기가 187+188×7=1503 바이트가 된다. 랜덤마이즈되는 데이터들은 실제 데이터와 PRBS 생성기에서 나오는 비트하고 XOR 연산을 통해서 이루어진다.

디스크램블러의 동작은 스크램블러와 같다. 랜덤마이즈시킨 데이터들을 다시 실제 데이터로 바꿔준다. 동기획득 과정대신 바이트로 들어오는 데이터를 비트단위로 바꿔서 MPEG-2 비트 스트림을 만들어 준다.

Ⅲ. RS(Reed-Solomon) 코더

1. Galois Field 연산자

RS 코드를 코딩할 때 심볼 사이의 모든 연산은 유한 필드(Galois Field)에서 이루어진다. GF에서 덧셈은 XOR를 이용해서 간단하게 수행할 수 있다. 또한 $GF(2^m)$에서는 모든 원소의 덧셈에 대한 역원은 그 자신이므로 $GF(2^m)$에서 덧셈과 뺄셈은 동일한 개념이다. 덧셈 연산은 각 비트를 XOR 함으로써 간단하게 구현된다. GF 곱셈기에 대한 구현은 두 가지의 곱셈 유형으로 구분할 수 있는데, 임의의 두 원소간의 곱셈을 할 경우와 임의의 한 원소와 상수 값과의 곱셈이 그것이다. 두 경우 사이에는 구현에 사용되는 하드웨어 크기가 달라지게 된다. 곱셈은 덧셈과 달리 상당히 복잡한 AND 와 XOR 의 조합 논리로 구성된다. 먼저 $GF(2^m)$에서 임의의 두 원소 A와 B의 곱은 각각의 비트의 곱을 불 대수와 같은 테이블로 정리할 수 있다. 그래서 상당히 복잡한 회로를 가지게 된다. 그러나 상수와의 곱셈은 비교적 간단하게 구현된다. 이는 같은 원소끼리의 덧셈의 결과는 '0'이 되는 $GF(2^m)$의 특성 때문이다. $GF(2^m)$에서 제곱 역시 상수 곱셈기와 마찬가지로 비교적 간단히 구현할 수 있다. 제곱기는 역원 계산 등에 유용하게 사용된다. GF에서의 역원은 나눗셈을 계산하는 회로이다. GF의 특성을 이용하

면 임의의 원소를 β라고 했을 경우 $\beta^{-1}=\beta^{254}$과 같은 관계를 갖고 있음을 알 수 있다. 그림 5는 GF 제곱기와 곱셈기를 사용하여 역원을 계산하는 간단한 블록도이다.

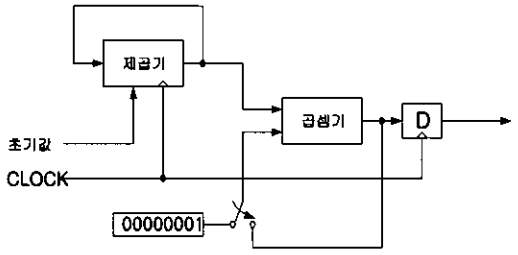


그림 5. 역원 계산기

역원을 계산할 때는 클럭을 이용한 순차 회로를 사용한다. 따라서 역원은 한번에 계산되는 것이 아니라 8번의 클럭이 필요하다. 제곱기에서는 처음 입력받은 초기 값을 계속 제곱을 한다. 처음 초기 값이 B라면 제곱기의 출력은 $B^2 B^4, \cdots$가 되는 것이다. 그리고 곱셈기에서는 제곱기에서 처음으로 출력되는 값과는 1을 곱하고 그 이후로부터는 제곱기의 출력과 바로 전에 갖고 있었던 값과 곱셈을 수행한다. 이와 같은 방법으로 역원의 계산이 완료된다. 그림 6은 $GF(2^8)$에서 A 값의 역원이 차례대로 계산되어지는 타이밍도를 보여준다. 역원 계산 과정은 그림 6에서 보는 것과 같이 8클럭이 필요하다.

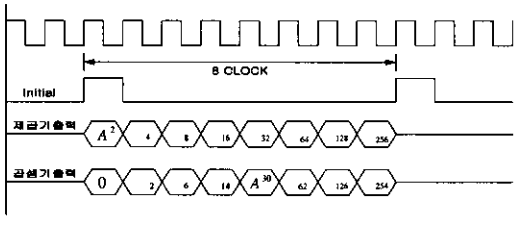


그림 6. 역원 계산기 타이밍도

2. RS 인코더

RS 코드는 q개의 원소를 갖는 $GF(q)$ 상의 블록 코드 계열이다. 본 논문에서 설계한 RS(204,188) 코드는 $q=2^8$ 인 RS(255,239) 코드의 shortened 코드이다. 이들은 모두 8비트로 이루어진 심볼 단위의 오류를 수정하며 이 심볼은 모두 $GF(2^8)$상의 원소이다. 또한 형성된 부호어 사이의 최소 거리가 N-K+1로서 Singleton-bound의 최대값을 갖는 maximum distance 코드이다.

일반적인 순회 부호(cyclic codes)의 인코더와 마찬가지로 RS 코드의 인코더는 정보 데이터 다항식 $d(x)$를 생성 다항식 $g(x)$로 나누는 나눗셈 회로로 구성할 수 있다. 다음 그림 7은 RS(204,188) 코드의 인코더를 보여 준다.

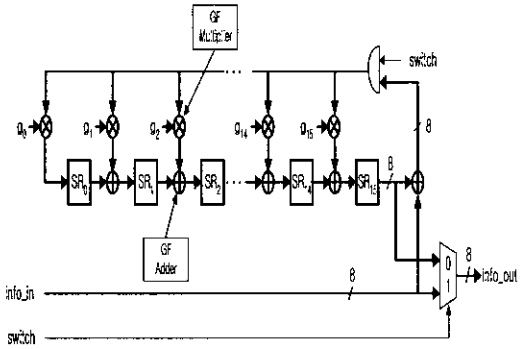


그림 7. RS 인코더 블록

g_i는 앞에서 설명한 생성 다항식의 계수들이며 이 계수들을 이용한 곱셈은 앞에서 설명한 임의의 두 원소와의 곱셈을 하는 방법과 같다.

인코더의 동작 원리는 다음과 같다. 먼저 '패리티'를 생성하기 위해서 $g(x)$로 구성된 나눗셈 회로를 이용하는데 RS(204,188) 코드의 경우 16개의 8비트 레지스터가 필요하다. 처음에 정보 데이터가 입력될 때에는 MUX 를 통해서 입력 데이터가 그대로 출력되어 부호어를 형성함과 동시에 $g(x)$로 나누는 일을 반복적으로 수행한다. 정보 데이터가 모두 입력되면 레지스터에는 나머지에 해당하는 '패리티'가 남아있게 된다. 이후 MUX 는 레지스터의 값을 차례로 출력하여 인코딩을 완료한다. 이 때에 MUX를 제어하는 switch 신호는 앞 단의 포맷팅 블록에서 제공받는데, 188 Byte 동안은 '1'을 유지하다가 '패리티'에 해당되는 부분 동안은 '0'을 유지하게 되므로써 앞서 말한 RS 인코더의 동작을 지원하게 된다.

3. RS 디코더

RS코드의 디코딩은 인코더에서 생성된 부호어는 생성 다항식 $g(x)$로 나누면 나누어 떨어지는 것을 이용하여 오류의 위치와 크기를 찾아내는 과정이다. 디코딩 과정에는 신드롬 계산, 오류 위치 다항식 생성, 오류 위치 계산, 오류 크기 계산, 오류 수정 등의 과정으로 이루어진다.

1) 신드롬(Syndrome) 계산

채널의 불완전한 특성 때문에 수신된 데이터에는 에러가 발생할 확률이 매우 크게 된다. 이를 보상해 주기 위해 수신단에서는 송신단에서와 반대 과정을 거쳐서 에러를 정정하고 원래의 메시지를 얻을 수 있다. RS복호를 하기 위해서 먼저 디인터리버(de-interleaver)에서 출력된 데이터들에 대해 신드롬을 계산한다.

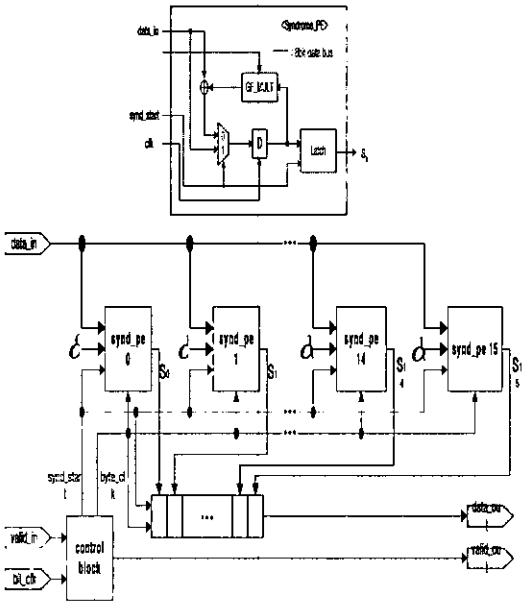


그림 8. 신드롬 계산 블록

신드롬은 블록 부호(block code)의 복호에 필수적으로 요구되는 정보이며 수신된 데이터에 $g(x)$의 근을 대입해 얻어지는 결과이다. 모든 부호어들은 생성 다항식 $g(x)$로 나누면 나누어 떨어진다. 따라서 수신된 데이터에 오류가 없다면 신드롬은 '0'이 된다. 그러나 만일 전송과정에서 오류가 발생했다면 생성다항식보다 한 차수가 낮은 신드롬 다항식이 생성된다. 여기서는 신드롬 다항식의 각 계수들을 산출해 내는 과정을 수행한다.

2) 오류 위치 다항식과 오류 평가 다항식 생성

일반적으로 계산된 신드롬 값들을 바탕으로 에러의 위치와 크기를 찾아내는 방법은 여러가지가 있다. 정정하고자 하는 에러의 수가 1-2개로 작은 경우는 look-up table을 사용한 복호 방식이 사용되며, 그 이상의 에러를 정정하고자하는 경우는 몇 개의 다항식을 풀어서 에러의 위치와 크기를 찾게 된다. 이 때 사용되는 방법은 LFSR (Linear Feedback Shift Register)을 이용하는 Berlekamp- Messy의 방식, 수정 유클리드 알고리듬을 이용하는 방법 등이 있다. 본 논문에서는 수정 유클리드 알고리듬을 이용하여 신드롬 다항식으로부터 오류위치다항식을 생성한다. 수정 유클리드 알고리듬은 처음 네 가지의 다항식을 초기 값으로 하여 이 다항식들이 서로 교차, 조합되면서 오류 위치 다항식(Error Locator Polynomial) 과 오류 평가 다항식(Error Evaluation Polynomial)을 만들어낸다.

신드롬 생성 블록에서 생성된 오류에 관한 정보를 갖고 있는 신드롬 다항식으로부터 오류의 위치를 찾는 과정이 필요하다. 지금까지 RS 코드의 디코딩 중 오류 위치 및 평가 다항식을 구하는 방법에 대해서는 여러 가지가 알려져 있다. Berlekamp-Messy 알고리듬, Peterson -Gorenstein-Zierler 알고리듬, 유한체 Fourier 변환을 이용한 알고리듬, Euclid 알고리듬, 수정 유클리드(Modified Euclid) 알고리듬 등이 그것이다. 본 논문에서는 오류 위치 다항식 $\sigma(x)$와 오류 평가 다항식 $\omega(x)$을 생성하기 위한 알고리듬으로 간단하고 구현이 용이한 수정 유클리드 알고리듬을 사용하였다. 왜냐하면 수정 유

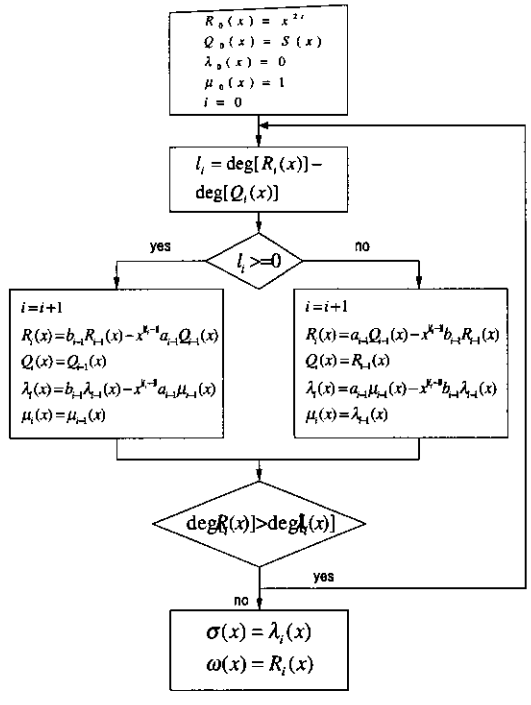


그림 9. 수정 유클리드 알고리즘의 순서도

클리드 알고리듬은 역원 계산이 생략되기 때문이다.

그림 9는 수정 유클리드 알고리듬의 순서도를 나타낸 것이다.

이제는 위의 알고리즘을 하드웨어로 구현하기 위한 방법을 소개하고, 기존의 방법들을 아주 완전히 보완한 새로운 구조의 수정유클리드 알고리즘을 제안한다. 시간과 공간을 절약한 아주 효율적인 구조이다.

알고리듬을 하드웨어적으로 구현하기 위해서는 먼저 R(x)와 Q(x)의 최고차항을 곱하기 위한 곱셈기와 레지스터 그리고 덧셈기 등으로 구성된다. 그림 10에서는 알고리듬의 기능 블록을 보여준다. 이러한 기능을 갖는 블록들이 16개의 직렬로 연결되어 위의 알고리듬을 수행하게 된다.

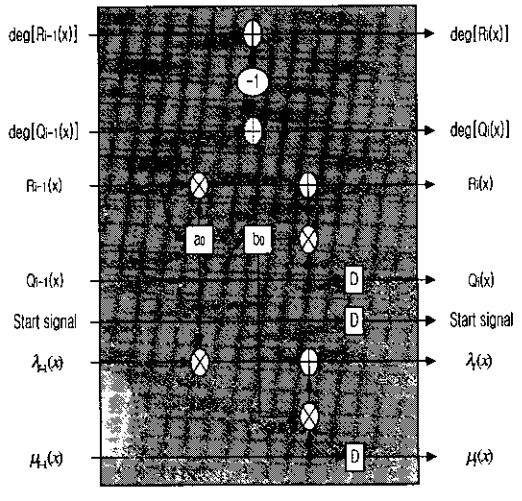


그림 10. Modified Euclid 알고리듬의 기능 블록

그림에서 점선으로 표시한 부분은 R(x)와 Q(x)의 차수에 따라 서로 입력되는 데이터가 교차되는 것을 보여준다. 즉, R(x)의 차수가 Q(x)보다 작을 경우에는 서로 교차되어 입력된다. 그리고 차수는 하나씩 줄어들게 된다. 그리고 차수가 줄어드는 맨 윗부분의 점선은 입력되는 $Q_i(x)$의 최고차항의 계수가 0일 때만 아래쪽에서 1을 빼게 된다. 시작 신호는 처음으로 입력되는 R(x)와 Q(x)의 최고차항의 계수를 로드하기 위해서 사용되어진다. 이렇게 최고차항의 계수를 loading하여 곱하여 더하게 되면 R(x)와 Q(x)중 하나의 처음 항은 반드시 '0'이 되어서 차수가 줄어들게 된다. 위 그림에서 보면 Q(x)는 하나의 지연 후 바로 나오게 되고 R(x)의 경우에는 곱셈기 두 개와 덧셈기 하나를 거쳐서 출력되게 된다. 따라서 두 다항식의 출력에는 서로 시간적인 차이가 생길 수 있게 되고 또 이런 시간적 차이가 누적이 되면 나중에 계산 결과에 중대한 영향을 미칠 수 있으므로 실제로 구현 할 때에는 출력의 끝단에 플립플롭을 하나씩 첨부하여 문제를 해결한다.

수정 유클리드 알고리듬을 구현하기 위한 방법은 크게 두 가지가 있다. 첫 번째 방법은 위 그림 10과 같은 블록 하나를 이용해서 반복적인 계산을 수행하는 방법이다. 두 번째 방법은 위와 같은 블록을 직렬로 계속 연결해서 구현하는 방법이다. 전자의 경우에는 블록 하나만을 사용하므로 하드웨어적인 손실을 줄일 수 있다는 장점이 있다. 그러나 초기값을 입력받기 위해서는 17 개의 직렬 쉬프트 레지스터가 필요하다. 왜냐하면 $R_0(x)$의 값이 x^{16}이기 때문이다. 그리고 모두 16번의 순회 반복 계산이 이루어지므로 모두 272 클록이 필요하게 된다. 그런데 신드롬 다항식은 204 클록마다 계속해서 갱신이 되고 있고 이 신드롬 다항식을 알고리듬에 계속 적용해야하므로 수정 유클리드 알고리듬은 적어도 204 클록 안에 수행되어야 한다. 실제로 신드롬 다항식의 계수를 로드하기 위한 클록까지 생각한다면 그보다 더 적은 클록 시간 안에 해결이 되어야 한다. 또한, 하드웨어 구현에 있어서 그림 10의 구조는 FLEX10K100 라이브러리를 초과하게 된다.

본 논문에서는 위와 같은 문제를 해결하기 위해서 새로운 형태의 수정 유클리드 알고리듬 구조를 제안하였다. 그림 11에 나와있는 제안한 구조에서는 위에서 언급했던 두 가지 문제를 모두 해결할 수 있다. 첫째, 제한된 시간 내에서의 연산을 살펴보면, 그림 11의 구조를 적용하게 되면 17단의 쉬프트 레지스터의 데이터들이 한번 연산을 마치게 되었을 때가 수정 유클리드 알고리듬 순서도의 루프를 두 번 수행한 결과를 얻게 되는데, 따라서 17단 쉬프트 레지스터의 데이터를 8번 연산시키면 오류 위치 다항식과 오류 평가 다항식의 계수 값들을 얻을 수 있다. 이 때에 사용되는 클록을 살펴보면, 쉬프트 레지스터 연산이 한 번씩 끝날 때마다, 0을 삽입시켜줘야 하므로 한 번 루프를 돌 때마다 18클록이 소요되고, 이를 8번 루프를 돌아야하므로 전체 144 클록이 소요된다. 둘째, 하드웨어 크기 면에서 볼 때에도, GF 곱셈기 8개와 GF 덧셈기 4개만을 사용해서 연산을 하므로 하드웨어가 상당히 감소함을 알 수 있다.

그림 11과 같은 구조를 생각해낸 배경은, 그림 9

에서 R(x) 다항식의 차수가 한 차수 떨어지려면 좌·우에 위치한 연산을 한 번씩 수행해야한다는 특성에 있다. 그림 11의 상단에 위치한 R(x), Q(x) 레지스터 블록에서 첫 번째 GF 덧셈기의 출력이 그림 9의 왼쪽 블록 연산에 해당되고, 두 번째 GF 덧셈기의 출력이 그림 9의 오른쪽 블록 연산에 해당된다. 또한, 그림 11에서의 점선은 알고리듬 수행에 필요한 상수 값들을 캡처하기 위한 것을 보여주고 있다.

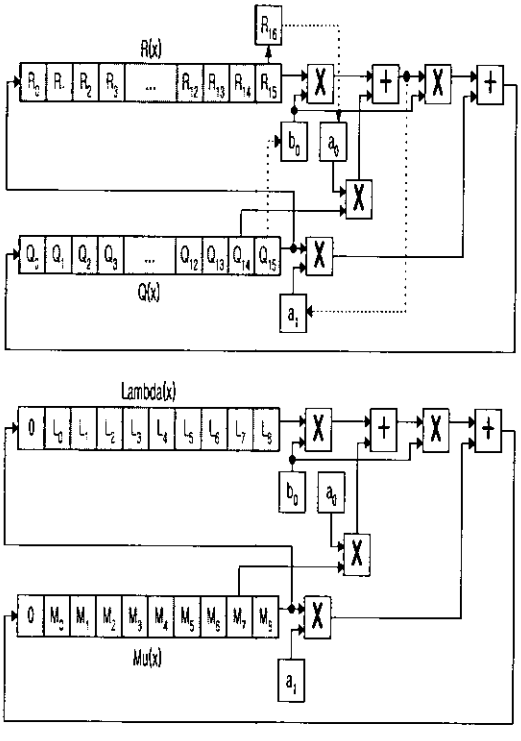


그림 11. 제안한 효율적인 수정 유클리드 연산 블록

지금까지 수정 유클리드 알고리듬을 이용한 오류 위치 다항식과 오류 평가 다항식의 계산을 모두 설명하였다. 여기서 계산된 두 다항식은 오류의 위치와 크기를 파악하는데 중요한 역할을 하게 된다.

Ⅳ. 길쌈 인터리버

DAB 시스템에서 사용하는 길쌈 인터리버는 12개의 가지를 가지고 있으며, FIFO(first-in, first-out) 쉬프트 레지스터 집합으로 구성되어 있다. j(j=0, 1, ..., 11) 번째 가지는 17*j의 깊이를 가지고 있다. 모든 처리는 바이트 단위로 처리한다. 인코딩된 데이터는 스위치를 통해서 차례대로 각각의 레지스터로 들어가고 각각의 레지스터는 쉬프트 동작을 반복한다. 디인터리버는 인터리버에서 배열된 레지스터의 구조가 반대로 되어 있다.

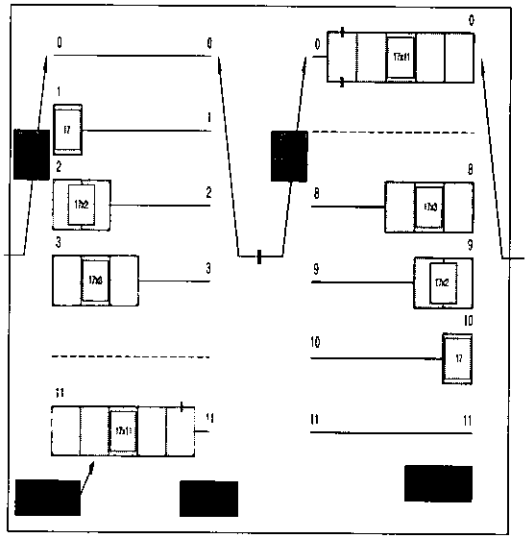


그림 12. 길쌈 인터리버의 구조

1. RAM을 이용한 쉬프트 레지스터의 구현

RAM을 이용한 쉬프트 레지스터의 구현은 간단하다. RAM에 접근할 때, 같은 주소에서 읽어내고, 그 주소에 데이터를 쓰는 동작을 반복하면 된다. 그러한 동작은 FIFO(first-in, first-out)구조를 만족한다.

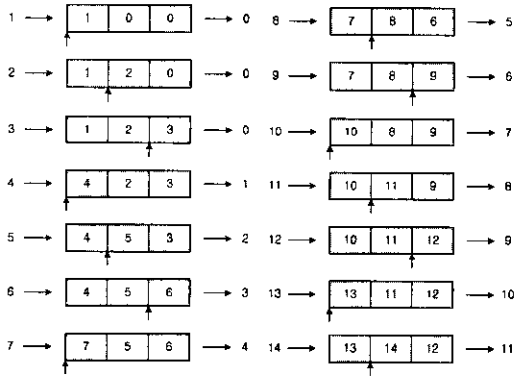


그림 13. RAM을 이용한 쉬프트 레지스터의 동작

그림 13은 RAM을 이용한 쉬프트 레지스터의 동작을 보여준다.

2. 최적의 메모리 구조를 갖는 길쌈인터리버

간단하게 RAM의 크기를 블록 인터리버와 같은 크기로 잡는 방법이 있다. 그러나 이러한 방법은 비효율적임이므로, 고려하지 않는다. 원래 블록 인터리버보다 약 절반정도의 메모리가 필요하다는 것은 이론적으로 알려져 있다. 그러한 취지에서 각 가지마다 각각 크기가 맞는 RAM을 선택하는 방법이 있다. 0번째부터 11번째까지 12개의 가지가 있지만, 0번째 가지는 저장공간이 필요 없으므로 11개의 RAM이 필요하다. j 번째 가지의 RAM의 크기는 $17\times j$보다 큰 2^k중 최소의 k을 찾으면 된다.

이렇게 구현하면 블록 인터리버의 경우보다 약 절반 가량의 메모리를 줄일 수 있었다. 하지만 쓰지 않는 메모리의 양이 너무 많은 문제점이 있다. 8번째부터 11번째까지의 쓰지 않는 메모리가 4번째에서 7번째에 해당되는 필요한 메모리의 양보다 한 바이트 씩 더 많음을 확인할 수 있다. 그림 14는 일반화 된 방법으로 구성된 RAM의 구조이다.

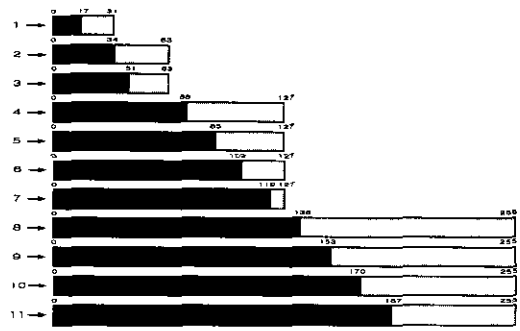


그림 14. 일반화 된 방법으로 구현된 RAM의 구조

기존의 방법에서 문제점은 쓰지 않는 메모리가 너무 많다는 것이었다. 그렇지만 그렇게 밖에 할 수 없었던 것은 하나의 RAM에 두 개의 가지를 넣어서 이중으로 엑세스하기가 너무 복잡하기 때문이었다. 또한 정확한 엑세스 타이밍을 맞추지 못하면, 제대로 된 출력결과를 얻기 힘든 것도 원인이라고 할 수 있다.

본 논문에서는 이러한 방법을 해결하고, 구현함으로써 최적의 메모리를 갖도록 만들었다. 1번째와 3번째, 2번째와 8번째, 4번째와 11번째, 5번째와 10번째, 6번째와 9번째의 가지를 묶어서 하나의 RAM을 사용하여 엑세스 가능하도록 구현하였다. 기존의 방법에서는 32 RAM(1개), 64 RAM (2개), 128 RAM(4개), 256 RAM(4개)이 사용되었지만, 제안한 방법에서는 128 RAM(2개), 256 RAM(4개)를 사용하였다. 그림 15는 제안한 구조로 구현했을 때의 RAM의 구조이다.

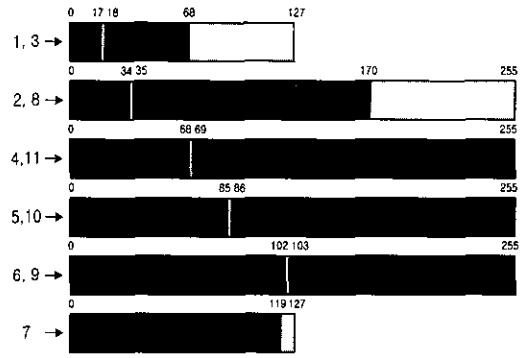


그림 15. 제안한 방법으로 구현한 RAM의 구조

V. 구현 결과

제안한 방법으로 구현했을 경우는 일반화된 방법에서 메모리의 쓰지 않는 메모리들의 크기가 다른 가지들의 크기보다 크게 되는 경우가 많이 발생함을 이용한 것이다. 그림 16에 나와 있는 방법으로 구현한 결과 일반화된 경우의 13,568 비트에 비해 대략 3,300 비트가 줄어든 10,240 비트를 사용하게 됨을 알 수 있다.

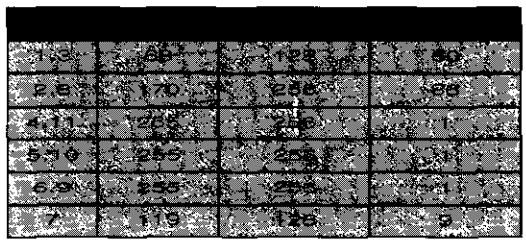


그림 16. 하나의 램에 두 가지를 넣은 구조의 메모리 사용 비트

이 경우 하나의 램에 두 개의 가지를 넣어서 사용하지 않는 메모리를 줄였다. 두 개의 가지를 넣는다는 개념을 적용했기 때문에 1,3번의 가지가 들어가 있는 경우와 2,8번의 가지가 들어가 있는 경우를 살펴보면 아직도 쓰지 않는 메모리의 크기가 다른 가지의 크기보다 더 큰 것이 남아 있음을 알 수 있다. 실제로 이렇게 구성했기 때문에 대략 3,300 비트의 메모리 비트를 줄이게 되었지만, 부가적인 로직 셀들은 증가하게 된다.

이 경우에는 쓰지 않는 메모리의 크기가 실제 사용 가지보다 크게 되는 경우가 없음을 알 수 있다.

지금까지는 메모리의 사용 비트의 관점에서 비교해 보았다. 메모리 사용량은 많이 줄어들었지만 실제로 하나의 램에 여러 개의 가지를 넣기 위해서 추가적인 부가 로직 셀들은 늘어나게 된다. 그럼에도 불구하고 메모리의 비트를 줄여야 하는 문제는 그림 17에서와 같은 경우를 통해 설명할 수 있다.

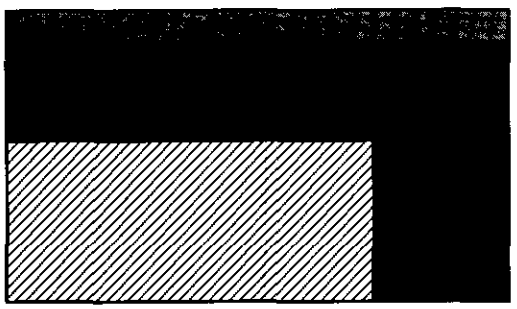


그림 17. LCs와 메모리의 구조

그림 17을 살펴보면 LCs(Logic Cells)가 들어가는 공간은 많이 남는 경우에 반해 메모리의 사용은 많은 경우이다. 길쌈 인터리버의 경우는 이 경우와 같이 실제 사용되는 LCs는 적은데 반해 메모리의 사용량은 굉장히 크다. 그래서 메모리의 사용 비트 수를 줄이는 방법이 반드시 사용되어야 하는 경우이다. 그러나 메모리의 공간이 많이 남고, LCs의 사용량이 큰 경우라면 첫 번째의 설계 방법을 이용해야만 한다. 되도록 간단한 컨트롤 로직을 사용해서 LCs의 개수를 줄이는 것이 좋기 때문이다.

실제로 FLEX10K 계열에서 사용할 수 있는 RAM의 비트 수들을 살펴보면 그림 18과 같다.

	576	1,152	2,880	4,992
	6,144	12,288	20,480	24,576

그림 18. FLEX10K 의 LCs와 RAM 비트 수

그러므로 사용되는 LCs의 개수와 RAM 비트 수를 고려해서 설계해야만 한다. 될 수 있으면 적은 코스트에 설계가 가능하도록 하는 것은 당연하다. 그래서 메모리의 사용 비트 수를 줄이는 방법은 반드시 필요하다.

메모리의 사용 비트 수를 줄이기 위한 방법으로 사용되었던 알고리즘의 특징을 정리해보면 다음과 같다.

첫째, 사용 비트 < 2^m(m은 정수)가 되는 가장 작은 m을 선택한다. 둘째, 사용되지 않는 메모리의 크기를 고려해 작은 가지를 큰 가지 쪽의 사용 램으로 병합한다. 셋째, 한 램의 사용되는 가지들의 순서는 연속되지 않도록 배열한다.

위의 세 가지 특징을 잘 적용하여 램의 구성을 한다면, 같은 타이밍 동안에 적은 메모리 사용을 할 수 있는 효과적인 구조를 얻을 수 있다.

Ⅵ. 결 론

본 논문에서는 DAB 시스템에서 사용하는 FEC 블록 중 SYNC신호를 찾고 데이터를 랜덤마이즈하기 위한 방법을 소개하고, 하드웨어 면적을 줄이기 위해 수정 유클리드 알고리즘과 길쌈 인터리버의 구조를 새로운 구조로 제안했다. RS 복호기의 수정 유클리드 알고리즘의 제안 구조에서는 GF 곱셈기 8개, GF 덧셈기 4개를 이용하여 하드웨어 크기를 효율적으로 설계할 수 있었다. 또한 길쌈 인터리버에서는 쓰지 않는 메모리를 최소로 하기 위해서 하나의 RAM에 두 번의 엑세스를 하는 구조를 선택하였고, 그 결과 128 RAM 2개, 256 RAM 4개만을 사용하여 최적의 메모리 구조를 갖도록 설계하였다.

참 고 문 헌

[1] Bernard Sklar, "Digital Communications fundamentals and applications", Prentice Hall, 1988.

[2] 박세현, "디지털 시스템 설계를 위한 VHDL 기본과 활용", 그린, 1998.

[3] ATSC, A Compilation of Advanced Television Systems Commitee Standards, April, 1996.

[4] S.B Wicker and V.K. Bhargava, Reed-Solomon Codes and Their Applications, New York, IEEE Press, 1994

[5] ETSI, Final draft prETS 300 744 : November 1996.

[6] E.R. Berlekamp, Algebraic Coding Theory, McGraw-Hill, New York, 1968.

[7] S.Lin and D.J. Costello, Error Control Coding : Fundamentals and Applications, Euglewood Cliffs., NJ:Prentice-Hall, 1983.

[8] R.E. Blahut, Theory and Practice of Error Control Codes, Addison-Wdsley, 1983.
[9] Bernard Sklar, Digital Communications fundamentals and applications, Prentice Hall, 1988.

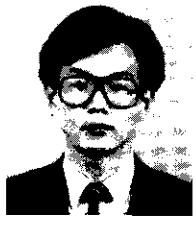
김 주 병(Joo-Byoung Kim)
1999년 2월 : 전북대학교 정보통신공학과 졸업
2001년 2월 : 전북대학교 영상공학과 석사
2001년 1월~4월 : (주)Televiewe 근무
2001년 4월~현재 : (주)ADT 근무
<주관심 분야> 암호학, CAS(Conditional Access System), CI(Common Interface), Embedded OS

임 영 진(Young-Jin Lim)
1998년 2월 : 전북대학교 정보통신공학과 졸업
2000년 2월 : 전북대학교 정보통신공학과 석사
2000년 11월~현재 : LG전자 디지털 네트워크 사업본부 디지털 네트워크 연구소 미디어 통신실 근무
<주관심 분야> 채널코딩, VOCODER

이 문 호(Moon-Ho Lee)

1990년 : 일본 동경대 전자과 공학박사
1983년 : 전남대 전기과 공학박사, 통신기술사
1985년~1986년 : 미국 미네소타 주립대 전기과 포스트닥터
1990년 여름 : 독일 하노버 공대 연구교수
1992년 겨울, 1995년 겨울 : 아흔 공대 연구교수
1998년 여름 : 뮌헨 공대 연구교수
2000년 여름 : 일본나가오가 과학기술대학 연구교수
1970년~1980년 : 남양MBC송신소장
한국통신학회(1986,1997), 대한전자공학회(1987), 전북도(1990)학술논문상 수상.
1997년 : 한국공학원 한림원회원
1997년 : 정보통신부정책심의위원 지상파 디지털 방송 추진위원
(現) 전북대 전자·정보공학부 교수, 정보통신연구소 소장.
<주관심 분야> 이동통신 및 영상통신, 채널코딩, 무선ATM.

이 광 재(Gwang-Jae Lee) 정회원



1986년 2월 : 전북대학교 전자공학과 졸업
1991년 2월 : 전북대학교 전자공학과 석사
1993년 3월~현재 : 전북대학교 전기공학과 박사과정
1995년 3월~현재 : 한려대학교 정보통신공학과 전임강사
<주관심 분야> 전력선통신, 이동통신, RF-ID