표준 8-VSB Advanced Television Standard의 개선된 RS Decoder ASIC 설계

정회원 최 진 호*, 전 문 석**

An Advanced ASIC Design of a RS Decoder for the 8-VSB ATV Standard

Jin-ho Choi*, Moon-Seog Jun** *Regular Members*

요 약

본 논문은 8-VSB Advanced Digital TV용으로 사용할수 있도록 ATSC(Advanced Television Standard Committee)의 규약을 만족시키도록 구현한 Reed Solomon 디코더에 대하여 기술한다. 구현된 RS Decoder는 공유된 Tree 구조의 Arithmetic 블록을 사용하여 종래의 기술보다 더 효율적인 연산기 구조를 제안하였으며 빠른 에러 탐지와 정정 시간으로 인한 FIFO의 사용갯수와 Latency Time을 크게 감소시킨 개선된 구조를 제안한다. 일반적으로 2N + A만큼의 Latency Time과 FIFO갯수를 N + A만큼으로 감소시켰다. 이 RS 디코더는 Verilog HDL로 설계되었고 Synopsys Design Compiler에 의해 합성되었다.

ABSTRACT

In this paper, we present a advanced Reed Solomon decoder which uses for 8-VSB Advanced Digital TV and supports fully the ATSC(Advanced Television Standard Committee) specification. The RS decoder presents an advanced architecture which has greatly reduced FIFO space and latency time. Using a shared arithmetic architecture of tree structure, the arithmetic computing time is more fast than the one of conventional architecture and the number of used FIFO is reduced to N + A from 2N + A. This RS decoder is designed by Verilog HDL and synthesized by Synopsys design compiler.

I. 서 론

Digital Television Standard인 VSB(Vestigial sideband) 시스템은 두가지 모드를 지원하는데 8 Level VSB와 16 Level VSB를 갖는다. 이중 지상파 방송 모드인 8 VSB는 NTSC 채널과의 상호 간섭을 최소화하여 현재에는 사용하지 못하는 taboo NTSC 채널을 포함한 모든 NTSC 채널을 통해 6 MHz 채널을 통해 19.28...Mbps의 Payload Data Rate를 제공한다. 이 전송시스템의 입력은 188-byte MPEG 호환 데이타 Packet들로 구성되는 19.39... Mbps 직렬 데이터 스트림이다. 이 Packet들은 187 Bytes의 Payload Data(19.28...Mbps)와 1 Byte의 Sync데이터로 구성된다. 이 입력되는 데이타는 Data Randomizer를 통해 랜덤화되어 FEC (Forward Error Correction)를 위해 Reed-Solomon (RS) 코딩의 형태로 변환된다(각 패킷 187 Bytes에 20 Bytes의 패리티 바이트가 더해진다). 이변환된 RS Coding Form은 1/6 Data Field Interleaving과 2/3 Rate Trellis Coding을 거친후 이 전송을 위한 Data Frame으로 Format되기위해 Data Segment Sync와 Data Field Sync가 더해진다. 그리고 나서 반송파 주파수와 함께 변조되어 안테나를 통해 Broadcasting된다. 앞에 논의된 RS Coding, Trellis Coding,

* LG전자 디지털미디어연구소 선임연구원(jinhochoi@lge.com), ** 숭실대학교 컴퓨터학고 통신연구실(mjun@computing.ssu.ac.kr)
논문번호 : K01101-0313, 접수일자 : 2001년 3월 13일

Data Format등은 모두 ATSC 표준[7]에 자세히 설명된다. 이 전송시스템의 송신부, 수신부역시 표준에 보여주는 것 같이 FEC부와 Pilot Insertion, Equalizer Filter, VSB Modulator, RF단으로 구성되어있다.[7]

본 논문은 수신시스템의 RF단 및 복조부를 거쳐 디지털 VSB신호로 만들어진 Encoding된 Frame을 Decoding하는 Decoding Part(FEC Part)중 RS Decoder 구현 Architecture에 대하여 제안하고 설계를 하였다.

미국의 디지털 TV전송규격인 ATV에서 쓰이는 Reed-Solomon Decoder는 갈로아스 Field GF(2^8)에 근거하는 RS(N,K)로 정의되는 시스템이다. 이 RS(N,K)의 N과 K는 각각 207 과 187이다. 이것의 교정할수 있는 Error수는 10이며 t=10으로 정의한다. Code Generator Polynomial과 Field Generator Polynomial은 다음과 같다[7].

$$g(x) = \prod_{i=0}^{i=2t-1} (x+\alpha^i)$$

$$= x(x+\alpha)(x+\alpha^2) \ldots (x+\alpha^{2t-1}), \quad t=10$$

$$f(x) = x^8+x^4+x^3+x^2+1$$

여기에서 α^i는($i \geq 1$ 의정수, α 는 갈로아스 피일드의 기본값) 갈로아스 피일드의 기본요소이다.

초기의 RS Decoder구현 구조는 System Clock을 사용하여 많은 FIFO와 Latency Time을 갖는 재래적인 RS Decoder를 구현하였고[2][3][4][5][6], 근래에는 ASIC공정의 발전에 따라 Fast Clock을 이용한 구조에 Resource Sharing을 사용한 Advanced Architecture를 제안한 구현이 [1]에 언급된다. 그림 1과 2에 초기의 Symbol Clock을 사용한 Conventional 구조와 Fast Clock을 이용한 구조를 보여준다.

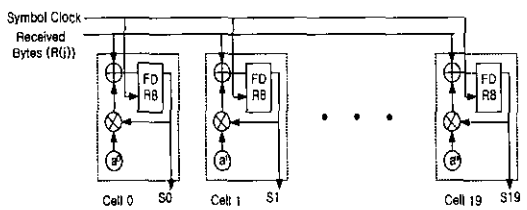

그림 1. System Clock을 사용한 재래적인 RS Decoder 구조의 한 예[2,3,4,5,6].

그림1과 2의 구조의 RS Decoder FIFO수와 Latency Time은 (2N + A)가 되지만 본 논문의 구현 구조에서는 RS Clock을 사용하고, Shared Arithmetic Block을 사용하므로써 (N + A)개의 FIFO, (N + A)T의 Latency Time까지 감소시킨다. 여기에서 N은 208 Bytes 즉, 한 Segment Byte이고, A는 에러위치계산과 기타시간에 대한 기간의 합이며 N보다는 작다.

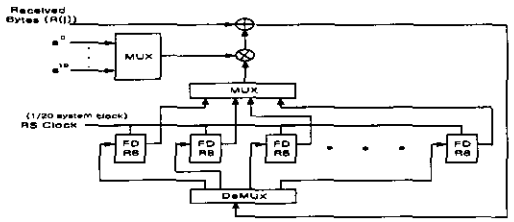

그림 2. RS Clock을 사용한 최근의 RS Decoder 구조의 한 예[1].

우리의 RS Decoder는 다른 구조와 비교하여 다른 특징을 크게 두가지를 갖는다. 우선, 이 구조의 동작 Clock으로 실제 구현가능한 Rs Bit Clock을 사용하는 점이다. Syndrome 계산을 위해 이상적인 RS Bit Clock을 구현하기위해서는, 즉 한 Symbol Clock에서 한 Symbol Data의 계산을 위해서는 1/2t RS Bit Clock이 필요하다[1]. ATV에서 RS Symbol Clock의 시간은 368 nsec이며 RS Bit Clock은 1/2t 배인 18 nsec의 시간이 되며 이의 Duty Cycle은 9 ns이 된다. 여기에서 t는 에러정정가능수 10이다. 실제 ASIC Design의 공정에서의 9 ns은 Critical Timing Delay 때문에 구현되기가 어려울뿐만 아니라, 추가적으로 18 ns에 대한 Clock 발생회로가 따로 필요하며 이 18 ns의 Clock과 Symbol Clock과의 동기문제도 상당한 구현의 어려움을 가지고 있다. 이러한 제약점들은 또한 비용추가의 원인이 된다. 그러므로 이런 두가지 문제를 해결하기위하여 본 구현에서는 1/8 RS Symbol Clock, 46 ns을 사용하였다. 이 Clock은 VSB 수신부의 복조 Part의 Timing Recovery부에서 사용되는 Sampling Clock을 재사용하여 추가적인 클럭 발생회로를 사용하지 않는 장점과 구현상의 Timing Delay를 만족하는 클럭 기간을 가진다.

두 번째 다른 특징을 갖는 구조는, 재래의 방법에서 각각의 Adder와 Multiplier를 두어 계산하는 구조와는 달리 각 알고리즘마다 *Shared Arithmetic* Block을 사용하여 순차적으로 Berlekamp, Error Evaluation, Correction Algorithm을 포함한 대부분의 다항식 계산을 공용으로 사용하며, 또한 갈로아스 피일드의 기본값 Primitive α 와 Inverse α 를

갖는 LUT(look-up Table)를 공용으로 사용한다.

그림 3에 전체 FEC Decoder의 구현 블록 다이어그램과 Decoding Flow를 보여준다. TCM Block은 AWGN 때문에 발생되는 Bit Error를 정정하며 Deinterleaver는 송신측에서의 Interleaved Data를 원래의 Data로 변환한다. 이 변환된 데이터는 RS Decoder에의해 10개의 Byte Error까지 검출하여 정정되어지고 Derandomizer에 의해 최종 복원된 데이터로 출력되어진다.

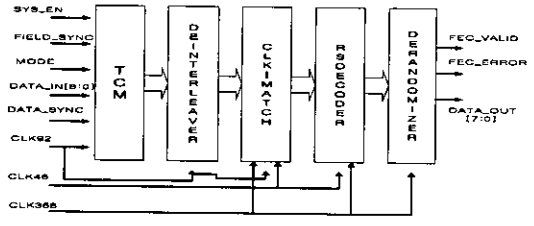

그림 3. Overall Block Diagram of FEC

이 그림3에서 입출력 신호들에 대한 설명은 다음과 같다. Clk46은 Timing Recovery부로부터 발생된 RS Bit Clock이고 Clk92는 TCM Symbol Clock이다. Clk368은 RS Symbol Clock이고 다른 Block들의 Byte Clock이 된다. sys_en은 System Enable이며 demo_ok는 복조 Part에서 Timing Recover의 끝났음을 알려 FEC Decoder의 Decoding 시작을 알리는 플레그이다. fec_error는 한 Segment (207 Bytes)가 교정할수있는수의 Errror들이 초과되었을 때 정정불가능을 나타내는 플레그이다.

Ⅱ. RS Decoder의 ASIC Design

2.1 배경

RS Code는 유한 Field GF(2m)에서 정의된다. Field의 성분은 field generator polynomial이라 하는 Irreducible, Primitive Polynomial f(x)에 의해 정의된다. Code Block은 N($\leq 2^m-1$) Symbol로 구성되며 각 Code Block은 Information과 Parity Symbol을 포함한다. t 개의 Error를 정정하기 위해 2t 개의 Parity Symbol이 부호화 과정 동안 계산되어 Information Symbol에 덧붙여진다. 따라서, RS Code Block은 K-1 차의 Information Polynomial m(x)과 N-K 차의 Code Generator Polynomial g(x)에 의해 N-1 차의 Polynomial c(x)로 정의된다.

$$c(x) = m(x)g(x)$$

여기에서 $g(x) = \prod_{i=k}^{k+2t}(x-\alpha^i)$

위 식에서 α^i는 Field의 Primitive Element이고, k는 Offset Term, t는 Error 정정 가능 한계이다. 모든 유효 Code Block은 Code Generator Polynomial의 배수가 될 수 있다.

전송 Code Block c(x)는 잡음이 존재하는 채널에서 Error가 생길 수 있다. 수신 Symbol $r(x)$는 전송된 c(x)와 Error Polynomial e(x)의 합으로 다음과 같이 나타내어진다.

$$r(x) = c(x)+e(x)$$

복호화 Algorithm의 첫 단계는 Error에 대한 정보를 내포한 Syndrome Polynomial $S(x)$를 계산하는 것이다. Syndrome Polynomial의 각 Coefficient S_i는 다음과 같이 정의된다.

$$S_i = r(\alpha^i) = c(\alpha^i)+e(\alpha^i)$$

위 식에서 Error가 없다면 모든 S_i는 0이 된다. Error를 정정하기 위해 Error의 위치와 크기를 구해야만 한다.

두 번째 단계는 Minimum Degree Error Locator Polynomial $C(x)$를 찾는 것이다. 더구나 Time Domain Decoding이 사용된다면 Error Magnitude Polynomial $\Omega(x)$도 계산되어야 한다. 이들 Polynomial은 Syndrome Polynomial과 다음과 같은 관계를 가지며, 이 식을 key equation이라고 한다.

$$S(x)C(x) = \Omega(x) \mod(x^{2t})$$

마지막 단계는 r(x)에 e(x)를 더해 Error를 정정하는 것이다.

$$\begin{aligned} c'(x) &= r(x)+e(x) \\ &= (c(x)+e(x))+e(x) \\ &= c(x)+(e(x)+e(x)) \\ &= c(x) \end{aligned}$$

e(x)의 Coefficient E_i는 Forney Algorithm에 의해 다음과 같이 구해진다.

$$E_i = -\alpha^i \frac{\Omega(\alpha^{-i})}{C'(\alpha^{-i})}$$

ATV에서 쓰이는 RS Decoder는 GF(28)에 근거한 RS(207, 187)로 정의되는 시스템이다. 이는 10개의 에러까지 정정(t=10)할 수 있다.

여기서 Synch Byte는 RS 부호화 되지 않으며, 여기서 설계한 RS Decoder는 다음과 같은 몇 가지 특징을 가진다. 먼저 이 시스템은 Symbol Clock이 아니라 Bit Clock에 의해 동작한다. 이것에 의해 병

렬 구조를 직렬 구조로 바꾸어 동작할 수 있게 된다. ATV에서는 t=10이므로 이상적으로는 Symbol Clock보다 20 배 빠른 Clock(RS Clock)이 필요하다. 하지만 20 배 빠른 Clk을 만들기 위해서는 RS Symbol Clock(2.69 Mhz)에 20 배 빠른 53.8 Mhz를 만들어주어야 하는 부가적인 상황이 발생한다. 그래서 이 Clk 보다는 Baseband Demodulator의 Timing Clk인 21.52 Mhz(TCM Symbol Clk의 2 배)를 사용한다. 이는 21.52 Mhz로부터 분주하여 TCM Symbol Clk, RS Symbol Clk 모두를 지원할수 있어 Clk 재사용에 큰 장점을 갖는다. 또한, RBA(Recursive Berlekamp Algorithm)을 사용하여 Error Locator Polynomial을 계산하는 것과 Error Magnitude Evaluation계산할때 공통으로 사용할수 있는 Arithmatic Block을 만들어 연산을 할수 있도록 하였다. 이 Arithmatic구조는 트리구조로 구성이 되어 있어 일반 GF 연산은 물론 특정 RS Decoder의 특정 Magnitude값을 계산할때 한번에 반복해야 하는 번거로움을 줄였다. 그리고 가장 획기 적인 부분은 기존의 Recursive한 구조로 사용할때의 FIFO의 Latency Time을 2N + A 에서 N + A로 줄였다는 점이다. 이는 시간의 줄임은 물론 FIFO Space의 큰 절감으로 인한 장점을 갖게 된다. 그림4에 RS Decoder의 전체 다이어그램을 보여준다. 여기에서 A는 에러위치계산과 기타시간에 대한 기간의 합이며 N보다는 작다.

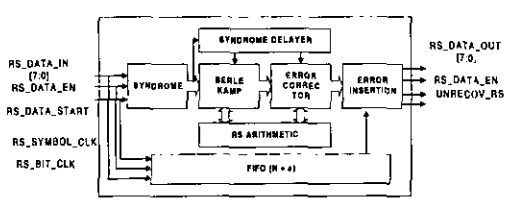

그림 4. RS Decoder의 전체 다이어그램

2.2 The Design of Syndrome Block

Syndrome은 Real-time Implementation을 위해 다음과 같이 Recursive하게 계산된다[2][3][6].

$$S_i = \sum_{j=0}^{N-1} r_j \alpha^{ij}$$
$$= \{..[(r_{N-1}\alpha^i + r_{N-2})\alpha^i + r_{N-3}]\alpha^i + .. + r_1\}\alpha^i + r_0$$

하나의 Byte값을 계산하기 위해서, 이상적이라면 *1/2t Symbol Clk*, 즉 RS Symbol Clk을 1/20 분주하여 사용한다면 하나의 Multiplier와 하나의 Adder만 있으면 가능 하겠지만 1/8로 분주한 Clk을 사용하므로서 병렬로 3 개의 Multiplier와 3개의 Adder들이 필요하다. 그리고 8번을 분주하는 *Counter*가 필요할것이다. 이들의 동작은 다음과 같다. 각 S1부터 S20은 각각의 Count 순번에 맞추어 Addition과 Multiply를 하고 Shift를 한다.

여기에서, rs_clk(Rs Bit clk)으로 동작하는 Count 군은 순서적으로 0번째 Clock{S19,S15,S7}, 1번째 Clock{S18,S14,S6}, . . . , 7번째 Clock{S8,S0}순으로 각각 일치되며 Alpha값 또한 0{alpha^16, alpha^8, alpha^0}, 1{alpha^17,alpha^9,alpha^1},.., 7{alpha^15,alpha^7} 순으로 전달되어 Shift된 S0 (Synd0), S8(Synd8), S16(Synd16)값과 계산된다는 것이다. 결국 한 Symbol Clk일때 하나의 rs_data_in (r(x))값에 대한 Syndrome 값이 rs_clk에 맞추어 순차적으로 S0 - S19까지 계산된다는 것이고 이 작업을 207만큼 반복될때 Syndrome결과값이 얻어지게 된다. 그림5에 블록 다이어그램을 보여준다.

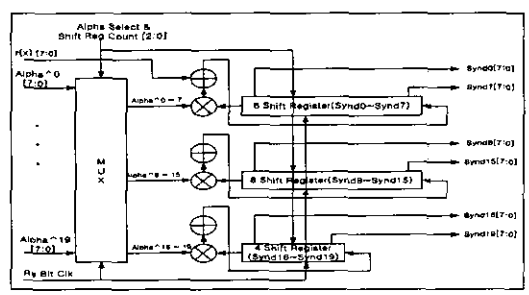

그림 5. Syndrome 계산의 블럭 다이어그램

2.3 Berlekamp Block의 설계

Berlekamp Algorithm(BA)는 Error Locator Polynomial C(x)를 얻기 위한 것이다. 이 BA Block은 초기화 단계, Discrepancy d_n의 계산, C_n(x)의 갱신과 같이 3가지 단계를 가진다. Error Location Polynomial C(x) 와 그것의 구현 알고리즘은 아래와 같다. [2][3][6]

$$C(x) = (1+\alpha^{i_1}x)(1+\alpha^{i_2}x)....(1+\alpha^{i_v}x)$$

구현에서 BA 블럭은 크게 3부분으로 구분된다. 즉, Syndrome결과 값을 Latching하는 Part와 Sn값을 Shift하여 Sn의 n-1,n-2,...를 저장하는 Part, dn, Dn, Cn을 구하는 Part 그리고 n,k,L,n-k를 구하는 Part로 구분된다. 이들을 구하는 것은 앞에서 설명한 알고리즘에 따르며 알고리즘 설명은 Algorithm 1에 보여준다. 또한 구현 Block Diagram이 그림 6에 보여준다.

그림 6의 Block Diagram은 Algorithm 1의 흐름대로 설명할 수가 있다. Syndrome에서 계산된 Synd0 - Synd19까지의 데이터가 입력으로 들어오면 Latch Sn Block에서 이 값들을 Latch하여 보관을 하게된다. Sn Shifter Block은 dn을 계산할 때 필요한 Sn값을 Shift시켜주는 기능을 한다. 만일 Syndrome Delayer Block으로 부터 입력되는 no_error값이 High Active를 가지면 Segment는 에러가 없으므로 BA 기능은 종료되고 다음 Segment를 기다리게 된다. BA는 Algorithm 2을 수행하기 위해 3 State FSM을 갖는다. 알고리즘1의 Program4에서 보는바와같이 2t, 즉 최대 20번의 반복(N = 20)을 하는데 각 한번의 BA 알고리즘을 수행하기 위해서는 7 Clock이 필요하게 된다. 7 State라함은 000 ~ 111까지의 7번의 Rs Bit Clock을 의미한다. BA Block이 기능을 시작하면 3 State FSM Block이 Enable되어 State Counter가 시작한다. 000일 때 모든 BA Block이 Program1과 같이 초기화가 되며 Syndrome이 끝났음을 알리는 Flag가 입력되면 State가 001로 바뀐다. 001에서는 Dn과 dn의 계수값을 Multiply계산을 위해 Arith Block으로 주고, State 010에서 그 결과값을 받는다. 그리고 New Cn값을 구하기위해 이 결과 값과 Cn값을 Addition을 위해 Arith Block으로 주게되면 Arith Block의 11개의 Adder 출력에는 Cn의 결과값을 가지게된다. 이러한 Flow로 n=1일 때, n=2, ... , n=19때까지 Cn, Dn, dn을 계산하며 BA Algorithm을 수행하게 된다. 마지막에 dn을 계산할때는 현재의 n에서 계산된 Cn과 현재 n을 기준으로 Sn, Sn-1, Sn-2,...의 Shift된 값을 Arith Block에게 주면 Arith Block에서 그 결과 값이 발생될것이고 2.7의 Arith Block의 설명과 같이 Adder들의 Tree 구조의 최종 출력으로 Summation된 값을 최종 출력으로 받는다. Sn Shifter의 예를 들면, Program2의 Sn-i와 같이 n=6일 경우 S_0th는 Synd6가 되고 S_1st는 Synd5, ... ,S_6th는 Synd0가 되도록 Shift된다. 이러한 구조로 설계되었기 때문에 20번의 n을 수행하기 위해서는 초기화 단계(n = 0)를 뺀 19 * 7 Clocks이 소요된다. 즉 133 Rs Bit Clock이므로 8배의 RS Symbol Clock으로 환산하면 17T이면 BA 결과가 모두 얻어질수가 있다. BA계산이 끝나면 Arith Block은 다음 Error Correction기능을 위해 신호를 기다릴 것이다.

Algorithm1. BA구현을 위한 알고리즘.

Program 1. Initial condition

n = 0,L = 0,k = -1,C(z) = 1, D(z) = z

Program 2. Calculation *d*

$$d_n = \sum_{i=0}^{L} C_i S_{n-i}$$

Program 3. Update

step 1 : if $d_n = 0$ then go to step6

else next step

step 2 : $C_{temp}(z) = C_n(D) - d_n D_n(z)$

step 3 : if L ≥ n-k then goto step5

else next step

step 4 : L_temp = n-k k = n - L

$$D(z) = \frac{C(z)}{d} z$$

L = L_temp

step 5 : $C(z) = C_{temp}(z)$

step 6 : D(z) = D(z) z

step 7 : n = n + 1

Program 4. If N=2t then STOP else goto Program 2

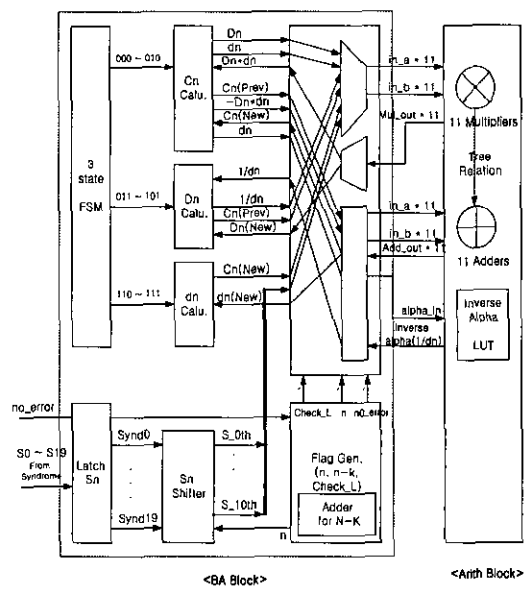

그림 6. BA 계산의 블럭 다이어그램

2.4 Syndrome Delayer

이 블록은 Syndrome Values $S_1, \ldots, S_{20}$ 이 최종적으로 계산이 끝나면 이값들을 Latch하고 에러위치와 에러정정을 위해 BA Block과 Error Correction Block에 제공이 된다. 그리고 만일 이 값들이 모두 Zero값을 가지면 즉 에러가 없으면 no_error라는 Flag를 발생시킨다.

2.5 Error Correction Block

BA에서 Error의 위치를 알아낸 후 아래와 같이 에러의 크기 Ei를 구해야 한다[4][5][8].

$$E_i = -\alpha^i \frac{\Omega(\alpha^{-i})}{C'(\alpha^{-i})}$$

Polynomial Ω(x) 의 x 에 α^{-i}를 넣어서 값을 구하는 과정을 evaluation 이라 한다. Ei를 구하기 위해, Ω(x)와 C'(x)의 계수를 알아야 한다. 그 이후, evaluation을 하게 된다. C(x)로부터 C'(x)의 계수가 추출되는 동안, Ω(x)의 계수는 새로 계산되어야 한다.

$$\Omega(x) = 1 + (S_1 + C_1)x + (S_2 + C_1S_1 + C_2)x^2 + \cdots + (S_t + C_1S_{t-1} + \cdots + C_t)x^t$$

$$\begin{aligned} C(\alpha^0) &= C_0 + C_1 + C_2 + \cdots + C_{10} \\ C'(\alpha^0) &= C_1 + C_3 + C_5 + C_7 + C_9 \\ &= Odd[C(\alpha^0)] \\ C(\alpha^{-1}) &= C_0 + C_1\alpha^{-1} + C_2\alpha^{-2} + \cdots + C_{10}\alpha^{-10} \\ C(\alpha^{-1}) &= C_1 + C_3\alpha^{-2} + C_5\alpha^{-4} + C_7\alpha^{-6} + C_9\alpha^{-8} \\ &= \alpha Odd[C(\alpha^{-1})] \\ &\vdots \\ C(\alpha^{-(N-1)}) &= C_0 + C_1\alpha^{-(N-1)} + C_2\alpha^{-2\times(N-1)} + \cdots + C_{10}\alpha^{-10\times(N-1)} \\ C(\alpha^{-(N-1)}) &= C_1 + C_3\alpha^{-2\times(N-1)} + C_5\alpha^{-4\times(N-1)} + C_7\alpha^{-6\times(N-1)} + C_9\alpha^{-8\times(N-1)} \\ &= \alpha Odd[C(\alpha^{-(N-1)})] \end{aligned}$$

위의 수식들을 구현하기 위하여 그림 7에 Error Correction Block에 대한 블록 다이어그램을 보여준다. 이 Error Correction 블럭도 BA와 마찬가지로 4 Bit State로 구성된 13 상태의 FSM으로 구성이 된다. 이들 각각을 Step으로 나타낸다면 다음의 Step Flow에 따른다.

Step0(0000) : Berlekamp가 끝났음을 알리는 Step_end신호가 입력되면 각 Registers을 Reset한다.
step1(0001) : Ω(x)(Bn으로도 표기하자)를 구하기위해 Sn과 Cn의 값들을 가용한 Arith Block의 Multiplier와 Adder에 따라 가용한 갯수씩 계산한다. 여기에서 Bn_coef10, Bn_coef1이 계산된다. 수식 Ω(x)에 t=10을 대입하여 전개하면 Mutiply들과 Addition들로 구성이 되고 이 전개된 수식들중 Multiply와 Addtion갯수를 세어 Arith Block의 11개의 Multiplier를 최대한 이용하여 계산할 수 있는 적절한 Bn을 결정하여 Arith Block의 Multiplier 입력들로 인가한다. 이 State에서의 예를 든다면, Arith Block의 가용한 11개의 Multiplier와 이들의 출력을 입력으로 받는 Tree구조의 11개의 Adder들을 통과하면 Arith Block의 최종 상위 Adder에 Bn10과 Bn1에 대한 계수값이 남을 것이다. 그림7의 Block 다이어그램에서, Omega(x) 계산에서 Cn과 Sn은 매 State마다 각기 다른 Bn을 구하기 위해 공용으로 사용된다. Arith Block의 구조는 2.7에 설명된다.

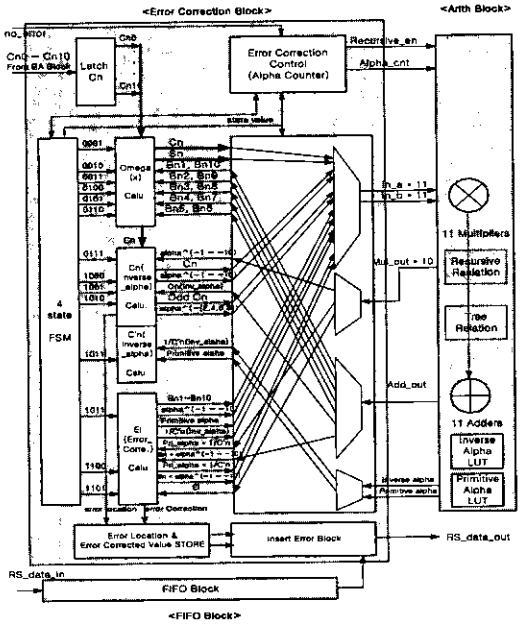

그림 7. Error Correction의 블럭 다이어그램

step2(0010) : Bn_coef9, Bn_coef2를 계산한다.
step3(0011) : Bn_coef8, Bn_coef3을 계산한다.
step4(0100) : Bn_coef7, Bn_coef4를 계산한다.
step5(0101) : Bn_coef6, Bn_coef5를 계산한다. 여기까지 완료되면 Ω(x)의 모든 값이 다 계산된다.
step6(0110) : α^{-i}을 Recursive하게 구하기 위해 recursive_en Flag를 Active 시킨다. 그러면 Arith Block에서 $\alpha^{-1}, \ldots, \alpha^{-10}$까지가 구해진다.
Alpha_cnt(0~N-1)값이 Arith Block으로 recursive_en값과 함께 입력되면 Inverse LUT Table을 통과하여 α^{-1}을 구하면 이값이 직렬로 연결된 10개의 Multiplier들을 거치게 된다. 그러면 각 각의 Mutiplier들의 출력들은 $\alpha^{-1}, \ldots, \alpha^{-10}$이 될것이다.
step7(0111) : 안전하게 alpha값을 연산하도록 한 Clk의 마진을 두었다. 이는 10개의 Multiplier가 직렬연결되었으므로 계산 Delay를 고려한것이다.

step8(1000) : $C(\alpha^{-i})$ 를 구하기 위해 Arith에서 구해진 Alpha값과 Berlekamp의 결과인 Cn값과의 Multiply 연산과 Addition연산을 실행하도록 각각 값을 제공한다.

step9(1001) : 충분한 연산 시간을 위해 한 Clk 지연을 한다.

step10(1010) : 이 스텝에서 no_error (즉, Syndrom 결과의 모든 Sn값이 Zero인 경우)의 Flag가 탐지되면 에러가 없는 Segment이므로 Errorcor Block의 알고리즘은 여기에서 끝내고(go to Step0) 다음 Segment값을 기다리게 된다. 그리고 Cn값에 Alpha 값을 대입하여 나온 결과값이 0이면 Error 위치(alpha_cnt)가 출력되고 Error Count(error_cnt)가 계수된다. 이때 alpha_cnt는 Alpha값이 α^0 부터 α^{N-1} 까지 이므로 206까지 수행 했으면 Step0로 가서 Evaluation을 끝내게 된다. 그리고 마지막에 Error 갯수와 Berlekamp로 부터의 L값과 같지 않으면 허용에러범위를 넘은 경우가 되므로 over_error신호를 출력하게 된다.

step11(1011) : 이 스텝부터는 결과값이 Zero가 되어 Error임을 발견한 상황에서 교정할 에러값을 찾기위한 단계 이다. Cn에 대한 미분값을 구하기위해 미리 계산해 놓은 미분 계수값과 해당 Alpha값을 Multiplier로 보내어(step10에서 이미 수행) Addition을 수행하고 나온 결과 값 Cn'을 Inverse 시켜 나온 값과 α^i값과 곱한값을 제공하고 이전에 계산해 놓은 Bn, 즉 $\Omega(x)$에 α^{-i} 값을 대입하여 $\Omega(\alpha^{-i})$를 구한다.

step12 : 그래서 $E_i = \alpha^i * \Omega(\alpha^{-i})/C'(\alpha^{-i})$이 계산이 된다.

step13 : 이렇게 계산된 Error Correction값은 error_end라는 플레그와함께 error_correction의 변수에다 옮겨져 출력하게되고 alpha_cnt를 증가시켜 Step7로 보낸다. 여기에서 alpha_cnt가 206이면 eval_end 플레그를 주고 이 기능블락의 종료를 알리며 Step0로 가서 다음 Segment를 기다린다.

이 Block의 계산시간은 최대 에러 10개가 발생할 때는 가정하면 $\Omega(x)$의 계수값을 계산하기 위해 7 Clock(Step0 ~ Step6), Error Evaluation을 위해 206 x 4(Step7 ~ Step10) = 824 Clocks이다. 마지막으로 10개의 에러정정을 위한 Step들을 계산하면 10 x 3(Step11 ~ Step13) = 30 Clocks이며 이들을 모두 합하면 861 Clocks이 되며 이를 Symbol Clock으로 환산하면 108 T가 된다.

2.6 에러위치탐지와 에러 Evaluation 구현에 따르는 계산 시간과 FIFO의 감소.

RS Bit Clock과 Tree구조의 공유된 Arithmetic Block을 사용하므로써, BA Block과 Error Correction Block의 계산시간이 각각 17 T (BA : 17 Symbol Clk)와 108 T (Error Evaluation과 Correction)가 필요하다. 여기에서 Error Evaluation을 구하는 재래적인 방법은 RS Symbol Clock을 이용하였을때 그것의 Evaluation Time은 N(207) T가 필요하다. 제안한 구조의 전체 Latency Time은 Syndrom계산 207T + BA 계산 17T + Error Correction 계산 108T를 합한 332 T가 되며 재래적인 방법은 Syndrome 207T + 약간의 BA계산 시간 + Error Evaluation 207T가 소요되므로 약 430T 정도가 소요된다. 결과적으로 제안한 구조는 약 100T정도의 시간감소를 이루었으며 FIFO 개수또한 100개 정도를 줄이는 결과를 이루었다. 그림8에 타이밍 다이어그램을 보여준다.

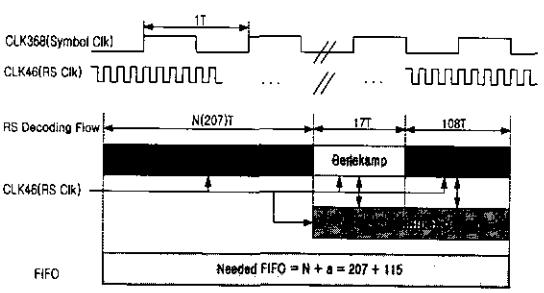

그림 8. 제안한 RS Decoder의 타이밍과 Decoding Flow 다이어그램

2.7 공유 RS Arithmetic Block의 구조

이 Arith Block은 다양한 기능을 갖는다. 즉 갈로아스필드(GF)의 범용 Multiply와 Addition기능을 가지며 또한 $\alpha^0, \ldots, \alpha^{10}$를 계산하기 위한 Recursive Computation 및 Tree 구조를 사용하여 BA와 Error Evaluation에 대한 특수한 계산을 수행하도록 설계되어진다. 이 Arith Block은 총 11 Multiplier와 21 Adder들 그리고 이 자원들을 각 기능에 맞도록 구성을 하도록 하는 제어 Flag들로 구성된다. 이 Flags은 승산기들과 가산기들을 기능에 맞도록 적절하게 조합되도록 한다. 이들 기능들에 대한 논리적인 구조와 이 구조들의 H/W 구현 구조의 Block Diagram이 그림9, 10, 11, 12에 보여준다.

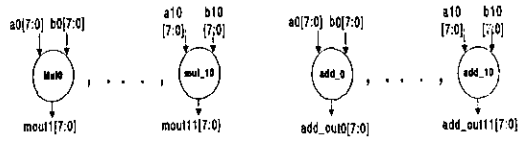

그림 9. Multiplier And Adder로서의 기본 기능

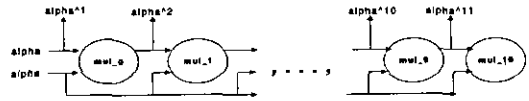

그림 10. $\alpha^0,\ldots,\alpha^{10}$의 계산을 위한 Recursive 기능

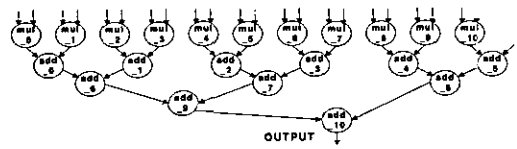

그림 11. BA와 Error Evaluation을 위한 Tree 구조

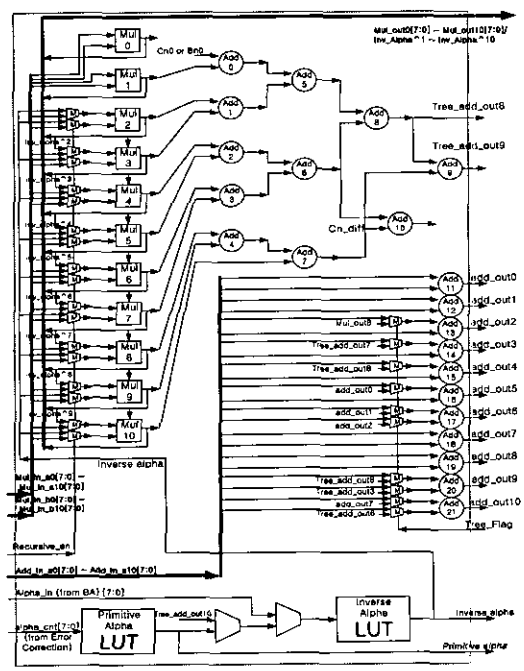

그림 12. 공유된 Arith의 Block Diagram

2.8 Insert Error Block

그림 4에서 보여주듯이, 이 블럭은 최종 RS Decoding 데이타를 출력하는 기능을 하는 블럭으로서 두개의 모듈로 서로 스위칭해가면서, 처리해야할 데이타 세그먼트에 대해 에러값과 위치값을 받아 저장하고 있다가 해당되는 데이타 세그먼트가 출력될때 FIFO에서 출력되는 데이타와 교정할 데이타의 EXOR을 실행하여 교정된 데이터를 출력한다. insert_err Block의 주요기능이 각 해당 FIFO에 대해 각각 Pointer를 두어 에러값을 교대로 저장하는 것이다. 이 모듈은 2개로 구성되어 있고 각각의 동작원리는 같으므로 하나의 예를 들겠다. 이의 Step은 다음과 같다.

step1 : Berlekamp기능이 끝나면 Step_end라는 신호가 들어오는데 이는 Errorcor Block에서 에러 Evaluation기능이 시작되었음을 알리는 것과 같다. 그러므로 상대 모듈이 현재 사용하지 않음을 확인한후 상대 모듈에게 사용함을 알리는 플레그를 띄우고 Evaluation이 끝났음을 알리는 eval_end라는 신호가 입력될때까지 최대 10개까지error_correction 값과 error_location값을 저장해 놓는다.

step2 : Evaluation이 끝났으면(eval_end가 입력) 상대 모듈이 사용될수 있도록 사용하라는 플레그를 주며 Pointer가 가르키는 해당 FIFO(pointer)가 출력이 됨을 기다린다. 그리고 출력이 완료되면 저장해 놨던 에러값과 위치값을 새로운 데이터 세그먼트에 할당하기위해 Step1으로 간다.

이러한 기능을 두 모듈이 서로 충돌나지 않도록 실행을 하게된다. 이들의 발생된 값을 이용하여 최종 RSD값을 출력하는데 해당 FIFO가 출력될 때 해당 모듈의 에러값과 위치값을 가지고 FIFO Block에서 제공하는 fifo_out_cnt값(이값이 바로 위치값에 대응된다)을 비교하여 같으때, 즉 에러 위치의 값과 같을때 FIFO Block의 출력 데이터 값과 에러교정 값과 EXOR을 하여 출력하면 복구된 데이타 값이 출력되게 된다. 여기에서 fifo_out_cnt값이란 208 byte의 한 segment가 처음 입력하여 332만큼의 FIFO(shifting)를 통과한후에 출력될 때 해당 세그먼트의 순차적 순서, 즉 위치를 나타내는 pointer의 값을 나타낸다.

물론 두 모듈에서 각각 10개씩의 error_location 값과 error_correction 값들이 존재하므로 해당 FIFO에 맞는 위치 값과 에러교정 값을 Muxing 시켜주어야 함은 당연할 것이다.

Ⅲ. 결 론

본 논문은 ATV에서 요구하는 Spec.을 완전히 만족하는 RS Decoder의 개선된 구현구조를 보여준다. FEC Decoder를 구축하는 가장 Critical Problem이 Area와 Latency Time이다. 특히 RSD가 이 Area와

Latency Time에 주요 원인이 된다. 이러한 문제들을 해결하기 위하여 RS Decoder는 Tree구조를 사용하는 공유된 Arith Block 구조를 채택하였고 복조 Part로부터 발생된 Clock을 RS Bit Clock으로 재사용하는 구조를 사용하여 RS Decoder의 Latency Time과 FIFO의 수를 그림 10에서와 같이 N + A (207 + 115)까지 감소하였다. 마지막으로 RS Decoder의 가장 상위 모듈의 합성결과를 그림 15에 보여준다. 이하 Berlekamp Block이나 Error Correction Block등 하부 계층의 합성결과는 합성그림의 복잡성으로 생략하였다.

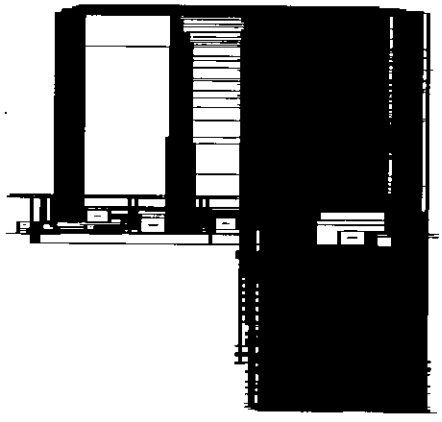

그림 13. RS Decorder Top module의 합성결과

참 고 문 헌

[1] Yonghee Im and Ohsang Kwon, Daewoo Electronics Co. Ltd., "An advanced architecture of RS Eecoders for advanced TV," in ICC '97, Montreal, Canada, Jun. 1997

[2] Shu Lin, Daniel J.Costello, Jr., Error Control Coding: Fundamentals and Applications, Prentic Hall, 1983.

[3] Richard E.Blahut, Theory and Practice of Error Control Codes, Addison-Wesley Publishing Co, May 1984.

[4] Howard M.shao, Irving S.Reed, On the VLSI Design of a Pipeline Reed-Solomon on Decoder Using Systolic Arrays, IEEE Trans. on computers, Vol.37, No. 10, Oct. 1988.

[5] Sterling R. Whitaker, John A. Canaris, Kelly B. Cameron, Reed Solomon Codec For Advenced Television, IEEE Trans. Circuits Systems, Vol1, No.2, jun. 1991.

[6] George C.Clark, Jr. and J.Bibb Cain, "Error-Crrection Coding for Digital Communications", Plenum Press. New York and London.

[7] Advanced Television System Committee , ATSC Digital TV Standard, Transmission charcteristics for terrestrial broadcast.

최 진 호(Jin-Ho Choi) 정회원

1992년 2월 : 숭실대학교 전자계산학과 졸업(학사)
1994년 2월 : 숭실대학교 컴퓨터학과 졸업 (공학석사)
1994년~1999년 : 대우전자 반도체연구소
1999년~현재 : LG전자 디지털미디어연구소 선임연구원
<주관심 분야> 정보통신, 디지털 미디어 ASIC Design

전 문 석(Moon-Seog Jun) 정회원

1980년 : 숭실대학교 전자계산학과 졸업(학사)
1996년 : University of Maryland 전산과 졸업(석사)
1989년 : University of Maryland 전산과 졸업(박사)
1989년 : Morgan State University 전산수학과 조교수
1989년~1991년 : New Mexico State University 부설 Physical Science Lab. 책임연구원
1991년~현재 : 숭실대학교 정보과학대학 부교수
<주관심 분야> 컴퓨터 알고리즘, 병렬처리, VLSI 설계, 암호학