일 반 투 고

MAP(Maximum A Posteriori) 복호 알고리즘을 이용한 MAP Decoder의 설계

김 지 호, 정 득 수, 송 오 영

중앙대학교 전자전기공학부

요 약

본 논문은 MAP(Maximum A Posteriori) 복호 알고리즘을 이용한 MAP Decoder의 설계에 관해 다룬다. 채널코딩기법은 채널을 통해서 디지털 정보를 전송할 때 신뢰성을 제공하기 위해서 사용되어진다. 즉 수신 단에서 수신된 정보의 오류를 검사하고 수정하기 위한 목적으로 송신 단에서는 디지털 정보에 부가 정보를 첨가해서 전송하게 된다. 그래서 무선 이동 통신에서 성능이 우수한 채널코딩기법은 우수한 통신 품질을 위해서는 필수적이라고 할 수 있다. 최근에 Shannon의 한계에 매우 근접한 성능으로 많이 알려진 오류정정부호로 터보코드가 발표되었고 많은 연구가 진행되고 있다. 터보코드의 부호기로는 RSC(Recursive Systematic Convolutional) 코드가 사용되며 복호 알고리즘으로는 주로 MAP 복호 알고리즘을 사용한다. 본 논문에서 제안된 MAP 복호기는 하드웨어로 구현하기 위해서 변형된 LOG-MAP 복호 알고리즘을 이용하였고 터보디코더의 반복 복호에 이용할 수 있다.

I. 서 론

현대 디지털 통신 시스템에서 없어서는 안 되는 중요한 요소 중의 하나가 오류정정 부호화 기법이다. 오류정정 부호화 기법은 채널 상에서 발생하는 오류를 효율적으로 교정함으로써 같은 전송 에너지를 사용하고서도 부호화 방식을 사용하지 않은 시스템에 비하여 더 우수한 성능을 얻을 수 있다. 바꾸어 말하면 같은 성능을 얻기 위하여 더 적은 에너지를 사용해도 되는 것이다. 단, 이때 어느 정도의 대역폭증가는 감수해야 한다. 이러한 부호화 방식이 주는 이득 때문에 여러 가지 효율적인 부호화 방식에 관한 연구가 광범위하게 수행되어 왔다. 오류정정 부호화기법(error correcting coding technique)[1]이란 일반적인 디지털 통신 시스템에서 채널 상에서 발생하는 오류를 정정하기 위하여 사용되는 방식으로써, 보통 소스 부호화(source coding) 기법과 구별해서 채널 부호화(channel coding) 기법이라고도 불린다. 여기서 소스 부호화기법은 가능한 부가 비트를 제거하여 송신되는 정보 비트를 최소화하는 기법을 말한다. 반면에 오류정정 부호화 기법을 사용하는 통신 시스템에서는 소스 부호기에서 나오는 정보 비트에 임의의 부가 비트(redundancy)를 덧붙여 오류가 발생하는 채널로 전송하기 위한 부호어를 만들고, 수신단에서는 수신된 부호어로부터 송신단에서 덧붙인 부가의 비트를 이용하여 오류가 없는 원래의 정보 비트를 복원하게 된다.

본 논문에서 다루는 터보코드는 1993년에 스위스 제네바에서 개최된 International Conference on Communications에서 프랑스의 Berrou가 "Near Shannon Limit Error Correcting Coding and Decoding : Turbo Codes"라는 제목으로 논문을 발표하면서 소개되기 시작했다[2,3]. 터보부호는 연판정 입/출력(soft-in/soft-out)이 가능하고, 정보신호에 대해서 서로

다른 인터리버에 의해 분리된 2개 이상의 구성코드(component code)들이 병렬연접(parallel concatenation)된 구성을 하고 있다[4,5,6]. 터보 부호의 기본 개념은 선행하는 구성코드의 복호기 연판정 출력(soft decision output)을 다시 나머지 부호기에 입력하고 이러한 과정을 반복함으로써 향상된 판단(decision)을 가능하게 하는 것이다. 터보 부호의 복호기로는 SOVA(Soft Output Viterbi Algorithm), MAP, Sub-MAP 복호기 등이 있는데 채널의 잡음분산평가가 필요하다는 단점이 있지만 일반적으로 성능이 우수한 MAP을 사용한다. 이러한 MAP기반의 터보부호는 MAP복호기의 복잡성과 많은 연산량, 아주 큰 인터리버 블록크기로 인해 하드웨어로의 구현에는 많은 어려움이 있다. 지금까지 터보코드에 대한 많은 복호 알고리즘과 복호 방법에 대해서는 발표가 되었지만 실제 구현에 있어서는 아직 미흡한 실정이다. 그러므로 본 논문에서는 터보코드에 대한 복호 알고리즘의 분석과 시뮬레이션을 통한 알고리즘의 성능을 확인하고 알고리즘에 대한 하드웨어 구조를 제안하는데 그 의의를 지닌다. 그리고 여러 복호기 중에서 성능이 우수한 MAP 복호기의 구조를 제안하며 제안된 구조에 따라 Verilog HDL을 이용하여 설계를 수행하였으며, 그 구조 검증을 위해서 Cadence사의 Verilog-XL™ 시뮬레이터에서 시뮬레이션을 수행하였다.

본 논문은 MAP(Maximum A Posteriori) 복호 알고리즘을 이용한 MAP Decoder의 설계에 관해 다룬다. 본문의 구성은 본론 1절에서는 터보코드가 회귀되는 정보를 이용한 반복 복호의 이득을 얻기 위해서 Bahl 알고리즘[7]을 변형한 MAP(Maximum A Posteriori) 복호 알고리즘[8]에 대하여 설명한다. 본론 2절에서는 본론 1절에서 설명한 MAP 복호 알고리즘을 하드웨어로 구현하기 위한 LOG-MAP 복호 알고리즘을 이용하여 MAP 복호기의 구조를 제시하고 동작원리에 관해서 설명한다. 본론 3절에서는 설계한 MAP 복호기의 시뮬레이션 결과와 복잡도를 분석하고 본론 4절에서는 MAP 복호기를 이용한 터보 코드 복호기의 구성과 성능에 대해 설명한다. 마지막으로 결론에서는 결과를 고찰하고 결론을 맺는다.

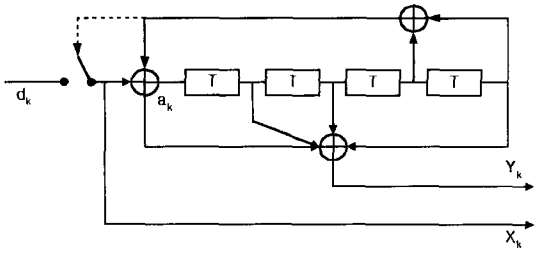
II. 본 론

1. MAP 알고리즘의 기본 이론

일반적으로 길쌈 부호(convolutional code)에 대한 Maximum likelihood 복호 방식은 비터비 복호 방식이라고 알려져 있으며, 길쌈 부호에 대해서는 비터비 알고리즘[9]과 Bahl 알고리즘의 성능은 같으면서 복호기의 복잡도는 Bahl 알고리즘이 4배 정도로 훨씬 크다. 그런데 터보코드에서 Bahl 알고리즘을 사용하는 이유는 터보코드가 회귀되는 정보를 이용한 반복 복호의 이득을 얻기 위해서는 반드시 Bahl 알고리즘을 변형하여 사용해야 한다는 것이다. 이산적인 무기억 가우스 채널을 통해서 수신된 시퀀스에 대해서 복호된 데이터의 APP(A Posteriori Probabilities)를 추정하는 알고리즘은 Bahl et. al.에 의해서 처음으로 제시되었다. 간단히 말해서 Bahl 알고리즘은 전체 수신된 신호 블록을 관찰한 후 임의의 시점에서의 정보 비트가 0일 확률과 1일 확률을 비교하여 확률이 더 큰 값을 복호 값으로 선택하는 것이다. 이것은 전체 어느 임의의 블록에 대한 관찰 없이 현재 시점에서 최적의 경로를 선택하는 비터비 알고리즘과는 그 개념에서 차이가 있다[10]. 본 장의 1절에서는 터보코드에서 매 복호시마다 각 복호된 값에 대한 신뢰성 정보를 다시 회귀시켜 반복적으로 복호를 수행할 때마다 복호의 성능을 향상시킬 수 있는 기법을 채택하기 위해서 수정된 Bahl 알고리즘에 대해서 설명하고 2절에서는 복호 알고리즘에 대하여 설명한다.

1) 수정된 Bahl 알고리즘(Modified Bahl Algorithm)

먼저 rate가 1/2인 systematic feedback



〈그림 1〉 R=1/2인 (23, 35) RSC 부호기

encoder에서 시간이 k일 때 〈그림 1〉과 같은 RSC(Recursive Systematic Code) 부호기에 의해 생성된 X_k와 coded data bit을 Y_k라고 했을 때 출력 부호어 시퀀스를 BPSK나 혹은 QPSK로 변조해서 AWGN(Additive White Gaussian Noise) channel을 통과한 후 수신된 시퀀스를 R_1^N라고 정의하면 식 (1)과 같이 나타낼 수 있다.

$$R_1^N=(R_1, \cdots, R_k, \cdots, R_N) \tag{1}$$

식 (1)에서 $R_k=(x_k, y_k)$는 시간이 k일 때 수신된 심볼이며 x_k, y_k는 식 (2), (3)과 같이 정의된다.

$$x_k=(2d_k-1)+p_k \tag{2}$$

$$y_k=(2Y_k-1)+q_k \tag{3}$$

식 (2), (3)에서 p_k, q_k는 분산이 σ^2이고 서로 독립된 랜덤 변수이다. 이 때 각 복호 비트 d_k에 대한 Log Likelihood Ratio $L(d_k)$을 식 (4)와 같이 정의할 수 있다.

$$\begin{aligned} L(d_k)&=\log\frac{P_r(d_k=1|R_1^N)}{P_r(d_k=0|R_1^N)}\\ &=\log\frac{P_r(d_k=1,\ S_k=m|R_1^N)}{P_r(d_k=0,\ S_k=m|R_1^N)}\\ &=\log\frac{\sum_m \lambda_k^1(m)}{\sum_m \lambda_k^0(m)} \end{aligned} \tag{4}$$

식 (4)에서 m은 부호기의 상태 값을 나타내고 $\sum_m$은 모든 상태 수를 나타낸다. 복호기에서는 LLR 함수 $L(d_k)$의 부호를 판별하여 전송되어진 정보비트값을 식 (5)처럼 $L(d_k)$을 임계값 0과 비교함으로써 복호기는 hard-decision 할 수 있다.

$$\hat{d}_k=\begin{cases}1: L(d_k)\geq 0\\ 0: L(d_k)<0\end{cases} \tag{5}$$

식 (5)에서 정의된 결합 확률을 계산하기 위해 다음과 같이 확률함수를 정의한다.

$$\begin{aligned} \alpha_k^i(m)&=\Pr(d_k=i,\ S_k=m,\ R_1^k)\\ &=\delta_i(R_k,\ m)\sum_{j=0}^{1}\alpha_{k-1}^j(S_b^j(m)) \end{aligned} \tag{6}$$

$$\begin{aligned} \beta_k^i(m)&=\Pr(R_{k+1}^N|\,d_k=i,\ S_k=m)\\ &=\sum_{j=0}^{1}\beta_{k+1}^j(S_f^i(m))\\ &\quad\times\delta_i(R_{k+1},\ S_f^i(m)) \end{aligned} \tag{7}$$

그리고 평균이 0이고 분산이 σ^2인 AWGN 채널에서 $\delta_i(R_k, m)$은 다음과 같이 표현된다.

$$\delta_i(R_k,\ m)=\exp\left(\frac{2}{\sigma^2}(x_k i+y_k Y_k^i(m)\right) \tag{8}$$

식 (8)에서 입력비트 i와 부호기 상태가 m일 때 부호기의 출력 값은 $Y_k^i(m)$으로 표현된다. 이 확률함수식을 이용하면 식 (9)와 같은 LLR 함수를 구할 수 있다.

$$L(d_k)=\log\frac{\sum_m \alpha_k^1(m)\,\beta_k^1(m)}{\sum_m \alpha_k^0(m)\,\beta_k^0(m)} \tag{9}$$

2) MAP 복호 알고리즘

정의된 순방향 메트릭 α, 역방향 메트릭 β, 가지 메트릭 σ로 복호알고리즘을 설명하면 아래와 같은 순서로 이루어진다.

(1) 시간 k=0에서 시작한다. 모든 수신 심볼들에 대해서 $\delta_i(R_k, m)$을 계산하고 (2^n개의 모든 가능한 부호 심볼에 대해, 이 경우 n=2) 크기 2^nN인 배열에 저장한다.

(2) i=0, 1에 대해 $\beta_{N-1}^i(S_f^i(0))$=1이고, 그 외

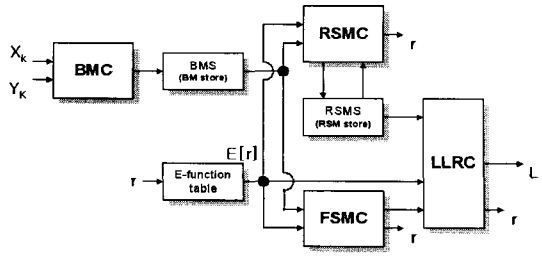
다른 모든 m과 i에 대해 $\beta_{N-1}^{i}(m)=0$으로 초기화 시킨다. 시간 $k=N-2$에서 시작하여, 확률함수식을 사용하여 $\beta_k^i(m)$을 반복적으로 계산해내고 크기 $2^{\nu}N$인 배열에 저장한다.

($S_f^1(m)=S_f^0(m)$일 때 $\beta_k^1(m)=\beta_k^0(m)$이면 배열의 크기를 절반으로 줄일 수 있다.)

(3) $i=0$, 1에 대해 $\alpha_0^i=\delta_i(R_0, 0)$이고 $m\neq 0$이고 $i=0$, 1일 때 $\alpha_0^i(m)=0$으로 초기화 시킨다. 시간 $k=1$에서 시작하며, 확률함수식을 이용하여 $\alpha_k^i(m)$을 반복적으로 계산해낸다. $k=0$을 포함하여 각 k에 대해 $\hat{d}_k$을 이용해서 $L(d_k)$을 계산한다.

2. MAP 복호기 설계

앞에서 유도한 식을 하드웨어로 구현하기에는 너무 복잡하다. 그래서 MAP 복호 알고리즘을 하드웨어로 구현할 수 있도록 단순화시키기 위해서 log을 사용하는 E 함수를 도입한 LOG-MAP 알고리즘을 사용하면 모든 곱셈의 계산이 덧셈으로 계산되어지며 logarithm과 exponential의 계산은 E 함수를 이용하여 계산하여 복잡도를 줄일 수 있다. MAP 복호기는 크게 가지 메트릭을 계산하는 BMC(Branch Metric Calculator) 블록, 역방향 메트릭을 계산하는 RSMC(Reverse State Metric Calculator)블록, 순방향 메트릭을 계산하는 FSMC(Forward State Metric Clculator) 블록 그리고 LLR값을 계산하는 블록인 LLRC(Log Likelihood Ratio Calculator)로 구성된다. 여기에 E 함수 테이블 블록이 추가되며 RSMC가 역방향으로 계산되기에 BMC의 결과를 저장하는 공간인 BMS(Branch Metric Store) 블록과 RSMC의 결과를 저장하는 RSMS(Reverse State Metric Store) 블록이 필요하다. E 함수 테이블 블록은 가지 메트릭을 계산하는 과정을 제외하고는 모든 계산 블록에서 참조가 되기 때문에 적절한 제어신호가 필요하게 된다. MAP 복호기에서는 메모리의 양을 줄이기 위해서 역방향 메트릭값만 저장하게 되고 순방향 메트릭은 따로 저장할 필요 없이 역방향 메트릭



〈그림 2〉 MAP 복호기 구조

값을 참조해서 LLR을 계산하도록 구현되었다. 또한 MAP 복호기의 구현을 간단하게 하기 위해서 LOG-MAP 알고리즘에 사용되는 E 함수를 새롭게 정의하였으며 이에 따른 메트릭값을 새롭게 유도하였다. 〈그림 2〉는 전체 MAP 복호기의 블록을 표시한 그림이다.

1) Log-MAP 알고리즘을 위한 E 함수의 정의 및 Lookup Table

본 논문에서는 MAP 복호 알고리즘을 하드웨어로 구현할 수 있도록 단순화시키기 위해서 E 함수를 식 (10)과 같이 정의하였다.

$$x\,E\,y=-\frac{1}{L_c}\ln(e^{-L_cx}+e^{-L_cy}) \tag{10}$$

식 (10)에서 L_c는 식 (11)과 같이 정의된다.

$$L_c\doteq\frac{4E_c}{N_o} \tag{11}$$

BPSK 변조에서 잡음의 편차는 식 (12)와 같이 정의되며

$$\sigma^2=\frac{N_o}{2E_c},\quad \frac{1}{\sigma^2}=\frac{E_c}{N_o/2} \tag{12}$$

식 (12)에서 $E_c=rE_b$로 표현되고 채널 당 에너지를 나타낸다. E 함수는 하드웨어로 구현하기 쉽게 식 (13)과 같이 전개할 수 있다.

$$\begin{aligned} x\,E\,y&=-\frac{1}{L_c}\ln(e^{-L_cx}+e^{-L_cy})\\ &=-\frac{1}{L_c}\times\ln(e^{-L_cx}(1+e^{-L_cy+L_cx}))\\ &=x-\frac{1}{L_c}\ln(1+e^{L_c(x-y)}) \end{aligned}$$

$$=y-\frac{1}{L_c}\ln(1+e^{L_c(y-x)})$$

$$=\min(x,y)-\frac{1}{L_c}\ln(1+e^{-L_c|y-x|}) \quad (13)$$

식 (13)에서 볼 수 있듯이 $x\,E\,y$와 $y\,E\,x$은 같으며 x와 y의 차이는 매우 작으며 E 함수의 값은 빠르게 0으로 수렴하게 된다. lookup table에는 식 (13)의 로그부분의 값을 미리 계산하여 ROM에 저장하여 필요할 때마다 참조하게 된다. MAP 복호기에서 E 함수의 사용은 순방향 메트릭, 역방향 메트릭, LLR 계산시 모두 사용된다.

2) BMC(Branch Metric Calculator) 블록

앞에서 정의된 E 함수를 이용하기 위해서 가지 메트릭 $\delta_i(R_k, m)$에 $-\frac{1}{L_c}$을 곱하고 로그를 취하면 식 (14)와 같이 표현된다.

$$D_i(R_k, m)=-\frac{1}{L_c}\ln\delta_i(R_k, m)$$
$$=-(x_k i+y_k Y_k^i(m)) \quad (14)$$

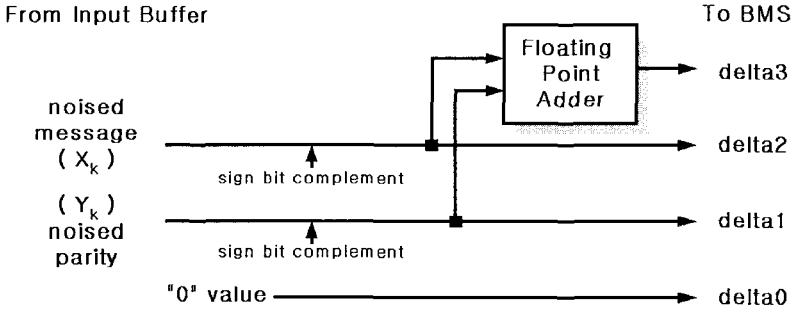
여기서 i는 입력 비트를 나타내고 $Y_k^i(m)$는 입력 비트에 대한 패러티 비트를 나타낸다. 입력 i가 0과 1의 값을 가지고 $Y_k^i(m)$도 0과 1의 값이므로 $D_i(R_k, m)$의 값은 각 x_k, y_k에 대해서 4가지 경우의 값을 가지게 된다. BMC 블록은 〈그림 3〉과 같이 x_k, y_k을 입력으로 받아서 delta0, delta1, delta2, delta3의 4가지 $D_i(R_k, m)$ 값을 생성해서 BMS 블록에 값을 보내고 메모리에 저장되게 된다. 그러므로 $4N$(N은 메시지 개수) 개의 메모리가 필요하게 된다. 메모리에 저장된 값은 입력 비트와 패러티 비트를 이용해서 참조할 수 있게된다. 식 (14)를 이용하여 BMC 블록을 구성하면 〈그림 3〉과 같다.

3) RSMC(Reverse State Metric Calculator) 블록

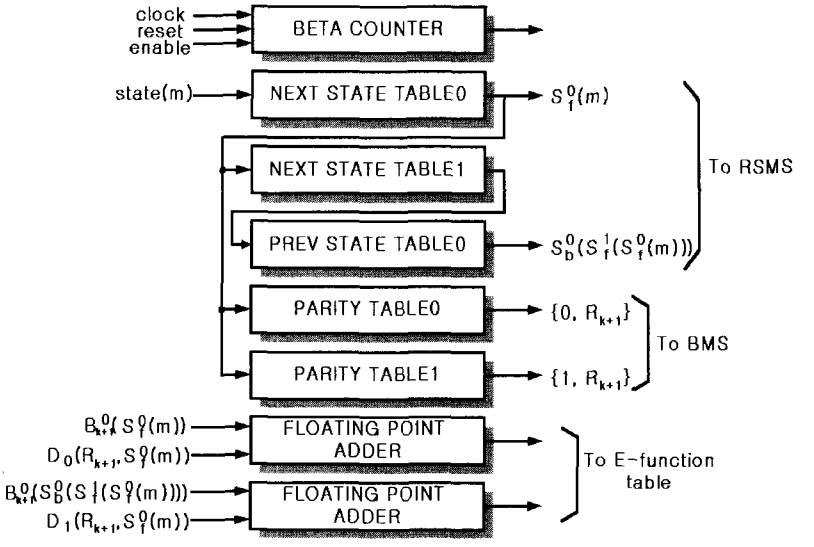
가지 메트릭과 같은 식으로 역방향 메트릭 $\beta_k^i(m)$에 $-\frac{1}{L_c}$을 곱하고 로그를 취하면 식 (15)와 같이 표현된다.

$$B_k^i(m)=-\frac{1}{L_c}\ln\beta_k^i(m)$$
$$=E_{j=0}^{1}\,B_{k+1}^i(S_f^i(m))$$
$$+D_j(R_{k+1},\,(S_f^i(m)) \quad (15)$$

우선 i=0, 1에 대해 $\beta_{N-1}^i(S_f^i(0))=1$이고, 그 외 다른 모든 m과 i에 대해 $\beta_{N-1}^i(m)=0$으로 초기화를 시키므로 로그를 취하면 $B_{N-1}^i(S_f^i(0))$는 0이 되고 그 외 다른 모든 m과 i에 대해서는 무한대 값으로 초기화가 된다. 역방향 메트릭의 경우에는 $S_f^1(m)=S_f^0(m)$일 때 $\beta_k^1(S_f^1(m))=\beta_k^0(S_f^0(m))$이므로 i가 0인 경우만 계산을 하고 $S_f^1(m)$는 $S_b^0(S_f^1(S_f^0(m)))$인 관계를 이용하여 구할 수 있다. 이렇게 하면 배열의 크기를 절반으로 줄일 수 있다. 식 (15)에서 i는 0일 때만 계산하는 식으로 나타내면 아래와 같이 식 (16)으로 전개할 수 있다.



〈그림 3〉 BMC 블록



〈그림 4〉 RSMC 블록

$$B_k^0(m)=B_{k+1}^0(S_f^0(m))+D_0(R_{k+1},\ S_f^0(m))\,E\cdot B_{k+1}^0(S_b^0(S_f^1(S_f^0(m))))+D_1(R_{k+1},\ S_f^0(m)) \quad (16)$$

식 (16)을 이용하여 역방향 메트릭을 구하기 위한 RSMC 블록을 나타내면 〈그림 4〉와 같이 구성할 수 있다. 베타 카운터를 이용하여 k개의 역방향 메트릭을 계산하게 된다. 수식에 사용되는 상태 값과 패러티값은 테이블을 이용하여 계산하게 되며 메모리의 어드레스로 사용되며 메모리에서 읽은 값을 서로 더해서 E 함수를 참조하게 된다.

4) FSMC(Forward State Metric Calculator) 블록

역방향 메트릭에서와 같은 방식으로 순방향 메트릭 $\alpha_k^i(m)$에 $-\frac{1}{L_c}$을 곱하고 로그를 취하면 식 (17)과 같이 표현된다

$$A_k^i(m)=-\frac{1}{L_c}\ln\alpha_k^i(m)=D_i(R_k,\ m)+E_{j=0}^1 A_{k-1}^i(S_b^i(m)) \quad (17)$$

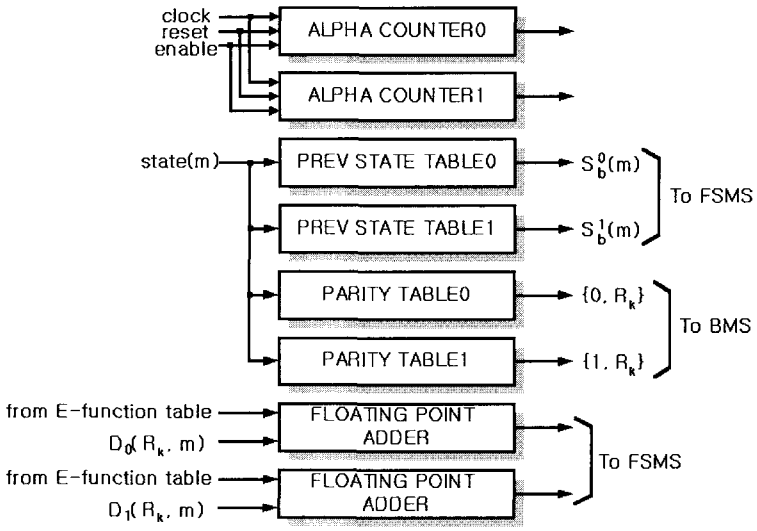
우선 $i=0$, 1에 대해 $\alpha_0^i=\delta_i(R_0,\ 0)$이고, $m\neq 0$이고 $i=0$, 1일 때 $\alpha_0^i(m)=0$으로 초기화되므로 로그를 취하면 $A_0^i=D_i(R_0,\ 0)$이 되고 그 외 다른 모든 m과 i에 대해서는 무한대 값으로 초기화가 된다. 식 (17)에서 i가 0일 때와 1일 때 아래와 같이 식 (18)로 전개할 수 있다.

$$A_k^0(m)=D_0(R_k,\ m)+A_{k-1}^0(S_b^0(m))\,E\,A_{k-1}^1(S_b^1(m))$$
$$A_k^1(m)=D_1(R_k,\ m)+A_{k-1}^0(S_b^0(m))\,E\,A_{k-1}^1(S_b^1(m)) \quad (18)$$

순방향 메트릭을 구하기 위한 FSMC 블록을 나타내면 〈그림 5〉와 같이 구성할 수 있다. 알파 카운터를 이용하여 k개의 순방향 메트릭을 계산하게 된다. 수식에 사용되는 상태 값과 패러티값은 테이블을 이용하여 계산하게 되며 메모리의 어드레스로 사용된다. RSMC 블록과 다른 점은 수식에서 볼 수 있듯이 E 함수 참조 값을 입력받아 역방향 메트릭을 계산하게 된다.

5) LLRC(Log Likelihood Ratio Metric Calculator) 블록

〈그림 1〉에서 정의한 E 함수를 연속적으로 계산하게 되면 식 (19)와 같이 표현할 수 있다.



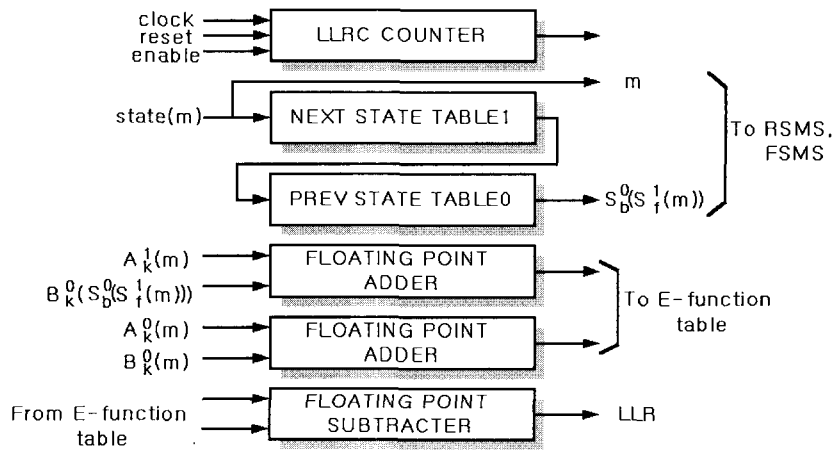
〈그림 5〉 FSMC 블록

$$E_{m=0}^{2^v-1} f(m) = f(0)\, E f(1)\, E \ldots\ldots E f(2^v-1) = -\frac{1}{L_c} \ln \left(\sum_{m=0}^{2^v-1} \exp(-L_c f(m)) \right) \quad (19)$$

식 (19)를 이용하면 $L(d_k)$을 식 (20)과 같이 전개할 수 있다.

$$\Lambda_k = E_{m=0}^{2^v-1} A_k^0(m) + B_k^0(m) - E_{m=0}^{2^v-1} A_k^1(m) + B_k^1(m) \quad (20)$$

LLR(Logarithm of Likelihood Ratio)를 구하기 위한 LLRC 블록을 나타내면 〈그림 6〉과 같이 구성할 수 있다. LLRC 카운터를 이용하여 k개의 LLR을 계산하게 된다. 수식에 사용되는 상태 값은 테이블을 이용하여 계산하게 되며 메모리의 어드레스로 사용된다. 메모리에서 역방향 메트릭값을 참조하게 되며 순방향 메트릭과 계산한 결과 값을 E 함수 테이블에 보내고 참조된 값을 입력으로 받아 들여 LLR을 계산하게 된다. 여기서 LLRC 블록이 RSMC나 FSMC



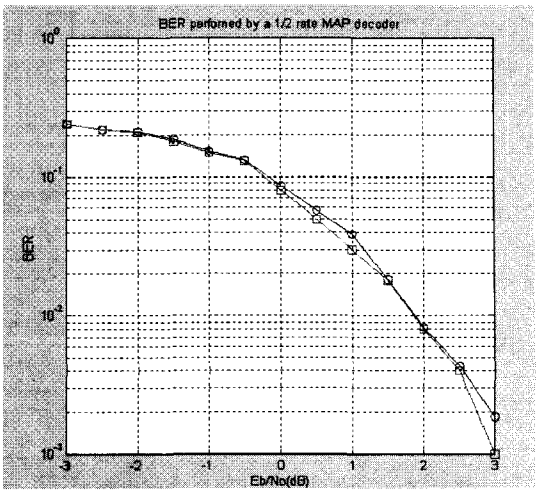
〈그림 6〉 LLRC 블록

블록과 다른 점은 수식에서 볼 수 있듯이 E 함수 테이블을 상태 수만큼 참조해서 그 때의 LLR을 계산하게 된다는 점이다.

3. 시뮬레이션 결과 및 복잡도 분석

1) Simulation 결과[1)]

〈그림 7〉은 부호율이 1/2이고 (23, 35) RSC 부호기를 사용했을 때 AWGN 채널상에서의 C 시뮬레이션에서의 BER과 설계한 MAP 복호기에서의 BER 성능을 비교한 그래프이다. 시뮬레이션 결과는 MAP 복호기 하나일 때의 성능이며 터보 디코더에 적용하여 여러 번의 반복을 수행하면 더 좋은 성능을 얻을 수 있을 것이다. 본 논문에서는 MAP 복호기의 구현을 부동소수점연산을 이용하였고 C++에서의 시뮬레이션결과와 거의 유사한 결과를 얻을 수 있었다. 시뮬레이션 결과에서 동그라미표가 되어 있는 선이 설계된 MAP 복호기의 BER 그래프를 나타내고 있으며 네모 표가 되어 있는 선이 C++ 시뮬레이션에서의 BER 그래프를 나타낸 것이다. 〈그림 8〉은 Synopsys에서 전체 MAP 복호기의 합성결과를 보여준다. 그림에서는 앞에서 설계한 각각의 블록들을 알아보기 쉽게 표시하였다. 각 블록들은 각각의 메트릭을 계산하는 블록인 BMC, RSMC, FSMC 블록들과 각각의 메트릭을 외부메모리에 저장하는 신호들을 콘트롤하는 블록인 BM_STORE, RSM_STORE, FSM_STORE 블록들과 함수 테이블을 콘트롤하는 E_FUNCTION_TABLE 블록과 입력을 받아들이는 INPUT_BUFFER



〈그림 7〉 설계된 MAP 복호기와의 BER성능 비교

블록과 전체 복호기를 콘트롤하는 DEC_CON 블록으로 구성되어 있다.

2) 복잡도 분석

〈표 1〉에서 보는 바와 같이 Log-MAP 알고리즘의 사용으로 모든 곱셈의 계산이 덧셈으로 계산되어지며 logarithm과 exponential의 계산은 E 함수를 이용하여 계산하여 복잡도를 줄일 수 있다. 또한 MAP 복호기에서는 많은 메모리의 사용이 필요한데 본 논문에서 사용한 복호 알고리즘은 δ을 저장하기 위한 2^nN(여기서 n=2이고 블록의 크기가 N인 경우)크기의 메모리와 β을 저장하기 위한 $2^{\nu-1}N$(ν는 부호기의 레지스터 크기)크기의 메모리가 필요하다.

〈표 1〉 Pure-MAP 알고리즘과 Log-MAP 알고리즘의 복잡도 분석

	Pure-MAP	Log-MAP
δ	N×(2×2$^{\nu}$)×(exponential and multiplications)	N×(2×2$^{\nu}$)×(additions)
α	N×2$^{\nu}$×(1 addition and 2 multiplications)	N×2$^{\nu}$×(1 E function call and 2 additions)
β	N×2$^{\nu}$(1 addition and 2 multiplications)	N×2$^{\nu}$×(1 E function call and 2 additions)
LLR	N×(1 logarithm and divisions and 2×(2$^{\nu}$−1) additions and 2×2$^{\nu}$ multiplications)	N×(1 subtraction and 2×(2$^{\nu}$−1) E function calls and 2×2$^{\nu}$ additions)

4. MAP 복호기를 사용한 터보 코드 복호기의 구성과 성능

MAP 복호기는 복호되는 값 자체를 연판정으로 내보내고 이 값을 다시 사용하여 복호 성능을 향상시킬 수 있다. 즉 한번 복호를 수행한 후의 출력을 다시 복호기의 입력 단으로 회귀시킨 후 이 회귀된 정보를 이용하여 이전보다는 좀 더 우수한 성능을 낼 수 있도록 하는 것이다. 〈그림 9〉와 같이 Turbo 복호기는 인터리버 1개, 디인터리버 2개, 직렬 연접된 두 개의 구성 복호기(DEC1과 DEC2)와 지연 부분들로 구성되어진다.

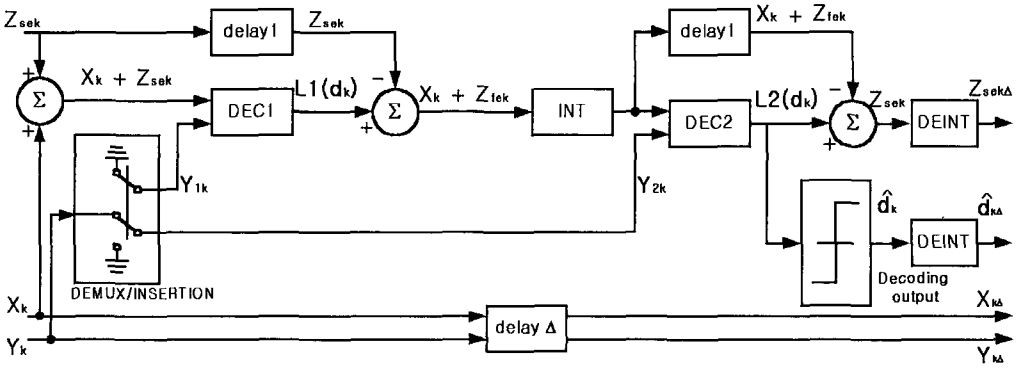
구성 복호기의 방식에는 SOVA(Soft-Output Viterbi Algorithm), MAP(Maximum A Posteriori), Sub_MAP 복호기 등이 있다. SOVA는 Viterbi 복호기를 softout으로 확장한 것으로 구현은 간단하나 MAP 복호기에 비해서 BER 성능이 떨어지는 문제점이 있다. 이러한 이유로 하드웨어 구현은 복잡하지만 BER 성능이 좋은 MAP 알고리즘이 사용되고 있는 추세이다. 본 논문에서는 앞에서 살펴본 MAP 알고리즘의 하드웨어 구현을 위해서 곱셈을 특정한 함수로 바꾸는 LOG-MAP 알고리즘을 사용한다. 이러한 MAP 기반의 터보부호는 MAP복호기의 복잡성과 많은 연산량, 아주 큰 인터리버 블록크기로 인해 음성, 테이터, 동영상을 포함한 무선 멀티미디어 서비스를 요구하는 고속 무선 통신시스템의 채널부호로는 문제점을 가지고 있어 여기에 대한 연구가 계속 되고 있는 추세[10]이다. 터보 복호기의 원리를 살펴보면 우선 MAP 복호기의 출력 $L(d_k)$는 RSC 부호가 systematic 부호이므로 정보 부호와 향상된 정보(Extrinsic information)의 합으로 표시된다. 즉 $L(d_k)=x_k+z_k$이며 z_k는 향상된 정보를 나타낸다. 두 번째 복호기인 DEC2에서 향상된 정보인 z_{sek}만 계산해서 이 정보를 첫 번째 복호기인 DEC1의 입력으로 사용된다. 그러므로 첫 번째 복호기인 DEC1에서의 는 식 (21)과 같이 표현 할 수 있다.

$$L1(d_k)=x_k+z_{sek}+z_{fek} \tag{21}$$

식 (21)에서 z_{fek}는 첫 번째 DEC1의 향상된 정보를 나타내며 z_{sek}는 두 번째 DEC2의 향상된 정보를 나타낸다. 그러면 첫 번째 MAP 복호기에서 사용되는 가지 메트릭 $\sigma_i(R_k,\ m)$는 식 (22)와 같이 정의된다.

$$\sigma_i(R_k,\ m)=\exp((x_k+z_{sek})\,i+y_kY_k^i(m)) \tag{22}$$

식 (22)를 식 (8)과 비교해 보면 입력부분에 DEC2에 의한 외부정보(extrinsic information)가 더해지게 된다. 그러면 DEC1의 출력 값인 $L1(d_k)$에서 DEC1에 의한 향상된 정보만을 DEC2에 사용하기 위해서 z_{sek}는 제거한 후에 인터리버를 통과시켜 DEC2의 입력으로 사용된다. DEC2에서는 부호기에서 이미 인터리버를 통과하고 전달되어진 패러티 정보를 입력으로 받아들여 $L2(d_k)$을 계산하게 된다. 그래서 $L2(d_k)$에서 x_k+z_{fek}을 제거하고 DEC2에 의한 향상된



〈그림 9〉 터보코드의 복호기 구조

정보인 z_{sek}만을 DEC1에 전달하게 되며 어느 정도의 반복을 한 후에는 $L2(d_k)$값을 이용하여 복호 비트를 결정하게 된다. 이런 방식으로 터보 코드의 복호기에서는 반복적으로 복호를 수행함으로써 복호 후에 발생한 오류를 점점 개선하는 방식을 취하고 있다. 터보 코드에서 복호기의 입력단으로 다시 회귀시켜주는 값은 각 시점에서 복호된 값이 어느 정도의 신뢰성을 갖는가를 나타내주는 값이어서 다음 복호 때 이러한 정도를 반영하게 되는 것이다. 이러한 반복 횟수는 증가할수록 성능은 증가하지만 성능이 증가하는 폭은 점점 감소하여 어느 횟수 이상이 되면 포화 상태에 이르게 됨을 시뮬레이션을 통해서 알 수 있다.

이제부터 터보코드의 성능을 좌우하는 요소에는 어떠한 것들이 있는지, 그래서 성능을 향상시키기 위한 방안은 무엇인지에 대해 기술하기로 한다. 우선 터보코드의 성능을 좌우하는 요소를 부호기의 측면에서 바라보면, 첫 번째가 RSC부호의 생성 함수(generator)와 RSC 부호기 내의 메모리 개수이다. 이것은 터보 코드를 구성하고 있는 것이 길쌈 부호의 일종인 RSC 부호이므로 길쌈 부호에서와 마찬가지로 최적의 성능을 낼 수 있는 생성기를 구성해야 하며, RSC 부호기의 메모리 개수가 증가할수록 성능은 향상된다. 그러나 부호기의 메모리 개수가 증가한다는 것은 해당 복호기의 복잡성이 증가한다는 것이므로 성능과 구현의 복잡성을 trade-off하여 결정하여야 할 것이다. 부호기 측면에서 터보 코드의 성능을 좌우하는 또 하나의 요소는 두 RSC 부호를 연결하는 인터리버이다. 인터리버의 사이즈가 클수록 성능은 좋아지며, 인터리버의 종류도 일반적인 블록 인터리버(균일 인터리버)보다는 비균일 인터리버가 성능을 더욱 향상시킬 수 있다고 알려져 있다. 이것은 터보코드의 거리 특성과 깊이 연관되어 있다. 이미 언급한 바대로 인터리버의 사이즈가 클수록 성능은 향상되지만 그만큼 지연이 생긴다는 의미이므로 이 또한 trade-off 분석이 요구된다고 할 것이다. 복호기의 측면에서 터보코드의 성능을 좌우하는 요소는 반복적으로 수행하는 복호 횟수라고 할 수 있을 것이다. 그러나 이 또한 반복적으로 수행하는 복호 횟수가 증가할수록 성능은 향상되지만 그만큼 지연이 생기게 된다. 복호기 측면에서 성능을 좌우할 수 있는 또 다른 요소 중의 하나는 회귀되는 정보의 양자화(quantization) 방식과 그 정도이다. 이 양자화 방식이 최적이 되기 위해서는 채널 잡음의 확률 밀도 함수에 따라 적절히 선택해 주어야 하며 복호기의 성능은 상세하게 양자화 될수록, 즉 무한대로 양자 화되어 실수 값 자체가 회귀될 때 가장 우수하게 된다. 그러나 디지털 시스템에서는 불가능한 일이며, 성능과 구현의 복잡도를 신중히 고려하여 선택해 주어야 할 것이다.

Ⅲ. 결 론

본 논문에서는 MAP(Maximum A Posteriori) 복호 알고리즘을 이용한 MAP Decoder를 설계하였다. MAP 복호 알고리즘을 하드웨어로 구현하기 위하여 LOG-MAP 복호 알고리즘을 이용하여 MAP 복호기의 구조를 제시하고 동작원리에 관해서 설명하였다. MAP Decoder는 Turbo Decoder의 반복 복호에 적용할 수 있으며 연판정 입/출력(soft-in/soft-out)이 가능하다. Turbo Decoder에 사용되는 MAP 복호기는 Verilog HDL을 이용하였으며 설계한 후 Cadence Verilog-XL™ 시뮬레이터로 동작을 검증하였다. 설계된 MAP 복호기는 Turbo Decoder의 반복 복호에 응용이 가능하다.

※ 본 논문은 2001학년도 중앙대학교 학술연구비 지원에 의한 것이며, 본 연구의 실험에서 사용한 S/W 및 H/W는 부분적으로 IDEC의 지원에 의한 것임.

참 고 문 헌

〔1〕 B. Sklar, "Digital Communications-

Fundamentals and Applications", *Prentice Hall*, Chapter 5, 1988

〔2〕 C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding Turbo codes," *Proc. Int. Conf. Comm.*, pp. 1064-1070, 1993

〔3〕 C. Berrou and A. Glavieux, "Hear optimum error correcting coding and decoding : turbo-codes," *IEEE Trans. Comm.*, pp.1261-1271, Oct. 1996

〔4〕 김수영, 이수인, "터보 코드 개발동향", 주간 기술동향 통권 888호, 한국전자통신연구원, 1999. 3. 13

〔5〕 Hagenauer, P. Roberston and L. Papke, "Iterative(turbo) decoding of systematic convolutional codes with the MAP and SOVA algorithms," *ITG Conf., Frankfurt, Germany*, Oct. 1994

〔6〕 S. S. Pietrobon, "Implementation and performance of a serial MAP decoder for use in an iterative turbo decoder," *IEEE Int. Symp. Inform. Theory & its Applic., Sydney, Australia*, pp.1073-1077, Nov. 1994

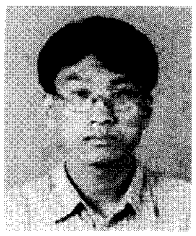
〔7〕 Bahl. L. R., Cocke, J., Jeinek, F. and Raviv, J., "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate", *IEEE Transactions on Information Theory, Vol. IT-20*, pp.248-287, 1974

〔8〕 S. S. Pietrobon and A. S. Barbulescu, "A simplification of the modified Bahl decoding algorithm for systematic convolutional codes," *Int. Symp. Inform. Theory & its Applic., Sydney, Australia*, pp.1073-1077, Nov. 1994

〔9〕 Forney, G. D., "The Viterbi Algorithm", *Proceedings of IEEE, Vol. 61.*, pp. 268-278, 1973

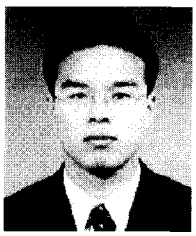
〔10〕 김수영, 홍성원, "터보 코드 연구동향", 주간 기술동향 통권 843호, 한국전자통신연구원, 1998. 4. 23.

저 자 소 개



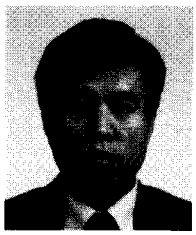
金 志 浩

2000년 2월 중앙대학교 전자전기공학부 학사, 2002년 8월 중앙대학교 전자전기공학부 석사, 2002년 9월~현재 : 중앙대학교 전자전기공학부 박사 과정, 〈주관심 분야 : VLSI시스템 설계 및 테스트, Channel Coding, 정보 보안〉



鄭 得 秀

1999년 2월 중앙대학교 제어계측공학과 학사, 2001년 2월 중앙대학교 제어계측공학과 석사, 2001년 2월~현재 : 삼성 전자, 〈주관심 분야 : VLSI시스템 설계 및 테스트, Channel Coding〉



宋 五 永

1980년 2월 서울대학교 전기공학과 학사, 1982년 2월 과학기술원 전기 및 전자공학과 석사술 1992년 2월 University of Massachusetts at Amherst, 전기 및 컴퓨터공학과 박사, 1982년 3월~1985년 5월 : 국방부 기술연구 사무관, 1991년 9월~1992년 10월 : Intergraph Corp. Electronics 수석연구원, 1992년 1월~1993년 11월 : IBM Microelectronics 수석연구원, 1994년 1월~1994년 8월 : 삼성전자 LSI사업부 수석연구원, 1994년 9월~현재 : 중앙대학교 전자전기공학부 교수, 〈주관심 분야 : 정보보안, 컴퓨터네트워크, VLSI시스템 설계 및 테스트〉