Turbo decoder의 설계

박 성 진, 송 인 채
숭실대학교 전자공학과
전화 : (02) 816-6073 / 팩스 : (02) 821-7653

Design of a Turbo Decoder

Sung-Jin Park, Inchae Song
Dept. of Electronics Engineering, Soongsil University
E-mail : skynet@hanul.ssu.ac.kr

Abstract

In this paper, we designed a turbo decoder using VHDL. To maximize effective free distance of the turbo code, we implemented pseudo random interleaver. A MAP(Maximum a posteriori) decoder is used as a primimary decoder. We avoided multiplication by using lookup tables(ROM). We expect that this small-sized turbo decoder is suitable for mobile communication. We simulated turbo decoder with Altera MAX+PLUS II.

I. 서론

Turbo code는 1993년에 Claude Berrou와 Alain Glavieux에 의해 논문이 발표된 이래 많은 사람들이 그 우수한 특성을 연구하기 시작했다[1]. 일반적인 concatenation block code는 직렬형태로 두 개의 inner code와 outer code를 갖지만 turbo code는 병렬형태의 구조를 갖는다[1]. 그리고 기존의 어떠한 convolutional code보다 뛰어난 성능을 발휘한다. 두 개의 encoder 사이에는 interleaver가 들어가는데 encoder의 출력단 사이의 분산도를 높여서 신호의 연속적인 오류를 피할 수 있게 한다. 기본 decoder는 MAP (Maximum a posteriori) decoder를 쓰며 interleaver와 deinterleaver가 사용된다. MAP decoder의 장점은 여러번 decoding 과정을 거칠수 있고, 반복횟수에 따라 좀 더 낮은 BER을 갖는다[2].

본 논문에서는 VHDL을 사용하여 복잡도와 크기를 축소하여 이동통신에 적합한 turbo decoder를 설계하였다[3]. Turbo code는 decoding하는데 많은 연산과정을 거치게 된다. 반복적이고 제한된 범위내의 연산은 미리 계산된 값을 메모리에서 참조하는 방식으로 연산과정을 줄이고 복잡도를 낮출 수 있도록 설계하였다.

II. Turbo code

Turbo code는 병렬형태의 concatenation code를 사용한다. 정보신호가 입력되면 정보신호, encoding된 부가정보, interleaver를 통해서 재배열된 신호를 encoding 한 후 출력된 부가신호, 이렇게 3개의 정보가 출력된다. 그림 1은 turbo encoder를 보여주고 있다. Interleaver는 입력된 정보를 다시 재배치 하는 것으로 임의의 한 신호와, 그 신호와 인접한 신호들 사이의 분산도를 높이는 역할을 한다.

Turbo decoder는직렬로 연결된 두 개의 MAP decoder와 그 사이에서 정보의 순서를 재배열 해주는 interleaver, deinterleaver로 구성된다. 그림 2는 Turbo decoder의 전체 구성도를 보여주고 있다. 입력단은

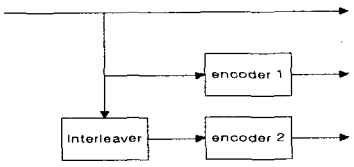

그림 1. Coderate R = 1/3인
Turbo encoder

메모리 버퍼에 연결되어 미리 수신된 블록단위의 신호를 입력받는다. 첫 번째 MAP decoder는 x_k와 y_{1k}의 입력을 받아서 decoding이 이루어진다. 첫 번째 MAP decoder를 거친 신호는 interleaver를 거쳐서 두 번째 MAP decoder로 y_{2k}와 함께 입력된다. Turbo encoder에서 interleaver를 거친 후 출력되는 신호가 y_{2k}이므로 y_{1k}

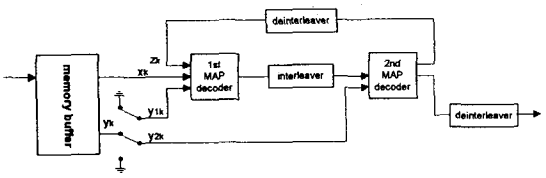

그림 2. Turbo decoder의 전체 block diagram

와 다른 순서로 재배열이 되어있는 상태이다. 첫 번째 MAP decoder에서 출력된 신호를 interleaver를 통해서 재배열하면 y_{2k}와 같은 순서로 배열되므로, 두 번째 MAP decoder에서 제대로 된 decoding 과정을 거칠 수 있게 된다. 출력된 신호는 다시 deinterleaver를 거쳐서 첫 번째 MAP decoder로 들어가게 되고, 필요에 따라 원하는 만큼 반복복호 과정을 거치게 된다[4].

Ⅲ. MAP decoder

SISO (Soft Input Soft Output) decoder중 대표적인 구현방법에는 MAP(Maximum a posteriori) 방식과 SOVA (Soft Output Viterbi Algorithm) 방식이 있다. 이 두 방식 모두 encoder의 상태변화를 확률적으로 추정한다. Viterbi decoder는 일반적으로 forward state에 따른 확률을 근거로 traceback과정을 거친 후 출력 값을 결정하지만, MAP decoder는 forward state와 backward state에 따른 확률을 모두 계산하고 그 두 값으로 출력을 결정하는 방식이다. 이것은 Viterbi decoder에 비해서 좀더 큰 폭으로 변화하는 출력 값을 hard decision에 반영시킬 수 있다. 또한 반복적인 decoding 과정으로 BER를 높이는 방식에서, 다른 알고리즘에 비해 매우 우수한 성능을 발휘한다.

3-1. Encoder

본 논문에서는 generator polynomial G = 13/15, constraint length v = 4인 RSC(Recursive Systematic convolutional encoder)를 사용하였다. 그림 3에서 입력이 들어가면 정보신호는 x, 오류정정을 위해 사용되는 부가정보는 y를 통해서 출력되는 것을 알 수 있다. Encoder의 초기 상태값은 모두 0으로 설정된다. MAP decoder에서는 이것과 함께 전체 블록의 encoding이 끝난 후 encoder의 상태를 모두 0으로 만들어 주기 위해서 tail bit라는 신호를 추가한다. 이것은 MAP decoder단에서 backward state 값의 계산을 수월하게 만들기 위해서이다.

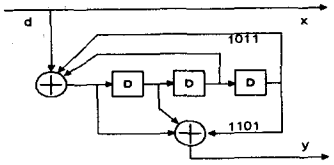

그림 3. G=13/15, v= 4 인
turbo encoder

3-2. MAP algorithm

MAP decoder의 입력에는 BPSK로 변조된 신호와 그 신호에 평균값이 0인 Gaussian random noise가 첨가된다. encoder 입력값이 d_k, 오류정정을 위해 첨가된 신호가p_k 이고 noise가 n_k, m_k라고 하면 decoder에서 수신된 신호를 (1)식과 같이 표현할 수 있다.

$$a_k = (2d_k - 1) + n_k, \quad b_k = (2p_k - 1) + m_k \qquad (1)$$

Decoding은 log likelihood ratio(LLR) Λ_k를 계산하고 그 값을 hard decision함으로써 이루어진다. Log likelihood ratio Λ_k 은 N번째 까지 수신된 신호가 $R_{1,N}$ 일 때 (2)식과 같다.

$$\Lambda_k(d_k) = \log\left(\frac{\Pr(d_k=1 | R_{1,N})}{\Pr(d_k=0 | R_{1,N})}\right) \qquad (2)$$

Hard decision은 (3)식과 같이 이루어진다.

$$\begin{aligned} &\text{if } \Lambda_k(d_k) \geq 0 \quad \rightarrow \quad d_k = 1, \\ &\text{if } \Lambda_k(d_k) < 0 \quad \rightarrow \quad d_k = 0, \end{aligned} \qquad (3)$$

Encoder의 state를 S_k, 시간을 k, encoding 하려고 하는 정보 신호를 i로 표현하면 (4)식과 같이 표현할 수 있다.

$$\Pr(d_k=i \mid R_{1,N}) = \sum_{m=0}^{2^{\nu}-1} \Pr(d_k=i,\ S_k=m \mid R_{1,N}) \quad (4)$$

$\lambda_{k,i}(m) = \Pr(d_k=i,\ S_k=m \mid R_{1,N})$ 로 치환하면 Λ_k를 (5)식과 같이 표현할 수 있다.

$$\Lambda_k = \log\left(\frac{\sum_{m=0}^{2^{\nu}-1} \lambda_{k,1}(m)}{\sum_{m=0}^{2^{\nu}-1} \lambda_{k,0}(m)}\right) \quad (5)$$

여기서 새로운 파라미터 $\alpha_{k,i}(m)$, $\beta_{k,i}(m)$를 (6),(7)식과 같이 정의할 수 있다.

$$\alpha_{k,i}(m) = \Pr(d_k=i,\ S_k=m,\ R_{1,k}) \quad (6)$$

$$\beta_{k,i}(m) = \Pr(R_{k+1,N} \mid d_k=i,\ S_k=m) \quad (7)$$

이 두 개의 파라미터로 Λ_k를 (8)식과 같이 표현할 수 있다.

$$\Lambda_k = \left(\frac{\sum_m \alpha_{k,1}(m)\ \beta_{k,1}(m)}{\sum_m \alpha_{k,0}(m)\ \beta_{k,0}(m)}\right) \quad (8)$$

Branch metric $\delta_i(R_k,m)$는 encoder로부터 수신된 신호와 trellis diagram의 각각의 state transition을 비교한 각 단계별 유사도(likelihoood) 이다. 수신받은 데이터의 soft값의 범위를 0에서 1까지라고 가정할 때 각 state에서의 유사도는 (9)식과 같다. 여기서 i와 $y_{k,i}(m)$은 encoder의 상태변화에 따른 encoder의 출력값을, σ^2는 수신된 신호의 분산을 의미한다.

$$\delta_i(R_k,m) = \exp\left(\frac{2}{\sigma^2}(|a_k-i|+|b_k-y_{k,i}(m)|)\right) \quad (9)$$

$\alpha_{k,i}(m)$와 $\beta_{k,i}(m)$는 각각 forward state metric, backward state metric을 나타내며 (10)식과 (11)식으로 표현할 수 있다.

$$\alpha_{k,i}(m) = \delta_i(R_k,m) \sum_{j=0}^{1} \alpha_{k-1,j}(S_j(m)) \quad (10)$$

$$\beta_{k,i}(m) = \sum_{j=0}^{1} \beta_{k+1,j}(S_j(m))\ \delta_i(R_{k+1},S_i(m)) \quad (11)$$

수신된 데이터 블록의 길이가 N이고, constraint length가 v일 때 다음과 같은 6단계의 과정을 거친다.

1단계 : 수신된 데이터 블록 전체에 걸쳐서 branch metric $\delta_i(R_k,m)$을 구하고, 메모리에 저장.
2단계 : t=0일 때 forward state metric 값을 초기화.
3단계 : t=1에서 t=N+v-1까지의 forward state metric $\alpha_{k,i}(m)$을 구하고 메모리에 저장.
4단계 : t=N+v-1일 때의 backward state metric 값을 초기화.
5단계 : t=N+v-2에서 t=0까지의 backward state metric $\beta_{k,i}(m)$를 구하고 메모리에 저장.
6단계 : t=0에서 t=N+v-1까지의 Λ_k를 계산하고 출력.

3-3. α, β, Λ 수식의 변형

MAP decoding 과정은 많은 곱셈연산이 들어간다. 이러한 곱셈연산으로 인해 복잡도 또한 크게 증가한다. 따라서 곱셈연산을 하지않고 단순히 덧셈, 뺄셈연산만으로 구현하기 위해서 수식을 변형시켜야 한다.

우선 (12)식과 같이 E 연산자를 정의한다. 여기서 $L_c = (2/\sigma^2)$이다.

$$E_{j=0}^{k} = -\frac{1}{L_c}\ln\left(\sum_{j=0}^{k} e^{-L_c j}\right) \quad (12)$$

그리고 각각의 파라미터에 log값을 취하면 다음의 (13),(14),(15),(16)식과 같은 새로운 파라미터를 정의할 수 있게 된다.

$$D_i(R_k,m) = |a_k-i|+|b_k-y_{k,i}(m)| \quad (13)$$

$$A_{k,i}(m) = D_i(R_k,m) + E_{j=0}^{1} A_{k-1,j}(S_j(m)) \quad (14)$$

$$B_{k,i}(m) = E_{j=0}^{1}[B_{k+1,j}(S_i(m)) + D_j(R_{k+1},S_i(m))] \quad (15)$$

$$\Lambda_k = E_{m=0}^{2^{\nu}-1}[A_{k,1}(m)+B_{k,1}(m)] - E_{m=0}^{2^{\nu}-1}[A_{k,0}(m)+B_{k,0}(m)] \quad (16)$$

새로 유도된 수식들은 위에서 정의된 E 연산자를 제외하고는 덧셈과 뺄셈으로 모든 파라미터의 연산이 가능하다. 그리고 E 연산자는 x와 y에 들어오는 값의 범위가 제한되어 있기 때문에 미리 계산값을 lookup table 에 넣어놓고 참조하게 되면 exponential 연산을

피할 수가 있다.

Ⅳ. Turbo decoder의 설계

그림 2에서 보여지는 바와 같이 turbo decoder에는 interleaver가 필요하다. Random interleaver는 다른 interleaver에 비해 분산도가 높아서 연속적인 오류에 대해 우수한 성능을 나타낸다. Interleaver는 그림 4와 같이 메모리와 pseudo random number 정보를 가지고 있는 ROM으로 구성된다.

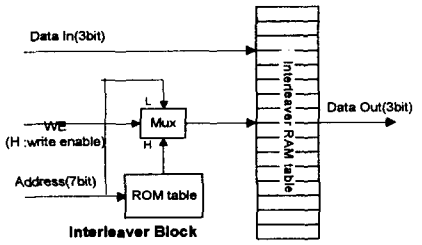

그림 4. Interleaver의 구조

그림 5는 미리 계산되어진 E 값을 참조하여 forward state와 backward state의 값을 계산하는 회로이다. Forward state metric $A_{k,i}(m)$을 구하기 위해서 $A_{k-1,i}(m)$ 이 입력되는 것을 볼수 있다. 반면 backward state metric $B_{k,i}(m)$을 구하기 위해서 $B_{k+1,i}(m)$이 입력되는 것을 알 수 있다.

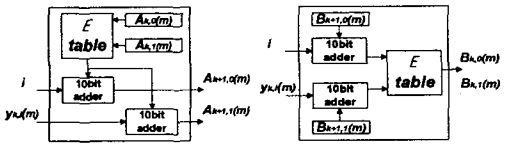

그림 5. forward state, backward state의 E function block

그림 6은 Turbo decoder의 최종단 Λ_k 연산을 위한 회로이다.

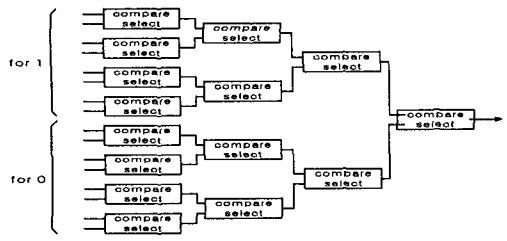

그림 6. Likelihood ratio Λ_k block diagram

설계된 turbo decoder는 반도체설계 교육센터(IDEC)에서 지원해 준 Altera MAX+Plus II를 사용하여 simulation하였다. 그림7은 Turbo decoder의 최종 출력 파형을 보여주고 있다. 파형은 두 번째 MAP decoder로부터 나온 값으로 soft output을 hard decision 과정을 통해서 0 또는 7로 바뀌어 나오는 것을 알 수 있다.

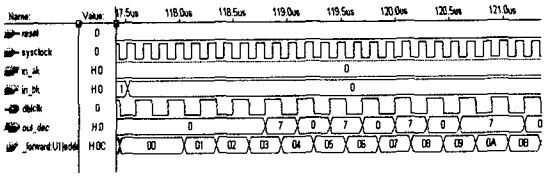

그림 7. Turbo decoder 출력파형

Ⅴ. 결론

본 논문에서는 한 블록의 데이터 크기가 192bit, constraint length = 4 인 Turbo decoder를 설계하였다. 시뮬레이션을 통하여 에러정정 기능을 수행하는 것을 확인하였다. 본 turbo decoder는 처리되는 블록의 크기를 축소하여 설계하였으므로 이동통신에 분야에 응용될 수 있을 것으로 기대된다.

참 고 문 헌

[1] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes." Proc. *ICC'93*, Geneve, Switzerland, pp. 1064-1070, May 1993.

[2] S.Benedetto, D.Divsalar, G.Montorsi,F.Pollara, "A Soft-Input Soft-Output Maximum A Posteriori (MAP) Module to Decode Parallel and Serial Concatenated Codes", JPL TDA Progress Report 42-127, Nov 15, 1996

[3] D.Divsalar and F.Pollara, "Turbo Codes for PCS Applications," *Proc. of IEEE ICC'95*, Seattle, Washington, pp.54-59, June 1995.

[4] J. Hageauer, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. Inform. Theory*, vol. 42, pp.429-445, Mar. 1996.