A High Speed Bit-level Viterbi Decoder

김민우, 조준동

성균관대학교 정보통신대학 전자전기공학과
수원시 장안구 천천동 440-746
Tel: +82-31-290-7200, Fax: +82-31-290-7683, E-mail:nakmw@vada1.skku.ac.kr

Abstract

Viterbi decoder 는 크게 BM(Branch metric), ACS(Add-Compare-Select), SM(Survivor Memory) block 으로 구성되어 있다. 이중 ACSU 부분은 고속 데이터 처리를 위한 bottleneck 이 되어 왔으며, 이의 해결을 위한 많은 연구가 활발히 진행되어 왔다. look ahead technique 은 ACSU 를 M-step 으로 처리하고, CS(Carry save) number 를 사용한 새로운 비교 알고리즘을 제안하여 high throughput 을 추구했으며, minimized method 는 block processing 방식으로 forward, backward 방향으로 decoding 을 수행하여 ACSU 부분의 feedback 을 완전히 제거하여 exteremely high throughput 을 추구하고 있다. 이에 대해 look ahead technique 의 기본 PE(Processing Element) 를 바탕으로 minimized method 알고리즘의 core block 을 bit-level 로 구현하였으며, code converter 를 이용하여 CS number 가운데 redundat number(1) 를 제거하여 비교기를 더 간단히 하였다. SYNOPSYS 의 Designcompiler 와 TSMC 0.18 um library 를 이용하여 합성하였다.

Keywords:

viterbi decoder, look ahead, minimized method

Introduction

Viterbi decoder 는 디지털 통신에서 채널 decoding 을 해주는 역할로 널리 사용되고 있다. 그 구조는 일반적으로 BM(Branch Metric)부, ACS(Add Compare Select)부,SM(Survivorpath Memory)부로 나뉜다.(그림1) BM 부는 수신된 code word 에 대해 hamming distance (또는 Euclidean distance) 를 이용하여 branch metric 을 계산하며, ACS 부는 BM 부로부터 입력받은 branch metric 과 이전의 state metric 을 더하여 path metric 을 구한다. 그리고 이를 비교하여 state metric 을 업 데이트하고, decision vector 를 결정한다.[1] SM 부는 ACS 부로터 입력받은 decision vector 를 바탕으로 원래의 정보를 찾아내는 역할을 한다. 이 중, ACS 부는 많은 반복적인 연산과 state metric 의 업데이트로 인한 feedback 때문에 속도 향상의 주요한 걸림돌이 되고 있다. 그래서 이의 해결을 위한 많은 연구가 진행되어 왔다. 그 중의 대표적인 고속 알고리즘으로 M-step look ahead technique 과[5] minimized method[4] 알고리즘을 들 수 있으며 이를 바탕으로 핵심 코어 블록을 설계하였다.

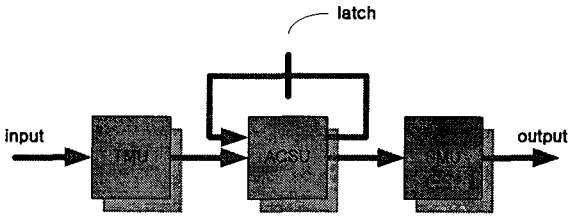


그림 1 Viterbi decoder 의 구조

본 논문의 구성은 다음과 같다. look ahead technique 과 minimized method 알고리즘을 살펴본 후, 이를 바탕으로 우리가 새롭게 개선한 점을 살펴본다. 그리고 마지막으로 개선된 점을 바탕으로 설계한 결과를 제시한다.

Algorithm Review

M-step Look Ahead

이 알고리즘은 ACS 부에서의 iteration operation 을 matrix 구조로 표현 하여 기존의 feedback operation 을 현저히 줄일 수 있음을 보여준다. 아래의 N=4 states 에 대한 trellis diagram 을 통 하여 그내용을 살펴본다.

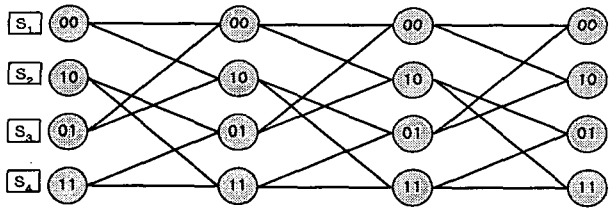


그림 2 *Trellis diagram, N=4 states*

$$S_{1,k+1} = \max(B_{11,k} + S_{1,k} ; B_{13,k} + S_{3,k})$$

$$S_{2,k+1} = \max(B_{21,k} + S_{1,k} ; B_{23,k} + S_{3,k})$$

$$S_{3,k+1} = \max(B_{32,k} + S_{2,k} ; B_{34,k} + S_{4,k})$$

$$S_{4,k+1} = \max(B_{42,k} + S_{2,k} ; B_{44,k} + S_{4,k}) \quad (1)$$

위의 trellis diagram 은 식 (1) 과 같이 표현될 수 있으며, 이는 다시 semiring algebra notation 을 이용하여 다음과 같은 linear form으로 표현할 수 있다. (S_k 는 state metirc 을 나타내며, B_k 는 branch metric 을 나타낸다.)

$$S_{1,k+1} = B_{11,k} \otimes S_{1,k} \oplus B_{13,k} \otimes S_{3,k} \quad (2)$$

$$S_{2,k+1} = B_{21,k} \otimes S_{1,k} \oplus B_{23,k} \otimes S_{3,k}$$

$$S_{3,k+1} = B_{32,k} \otimes S_{2,k} \oplus B_{34,k} \otimes S_{4,k}$$

$$S_{4,k+1} = B_{42,k} \otimes S_{2,k} \oplus B_{44,k} \otimes S_{4,k}$$

그리고 이는 다음과 같이 matrix-vector 의 product 형태로 나타낼 수 있다. 이 때, 주의할 것은 matrix-vector operation 이 일반적인 곱과 합의 연산이 아니라 합과 비교의 연산이라는 점이다.

$$S_{k+1} = A_k \otimes S_k \quad (3)$$

$$A_k = \begin{bmatrix} B_{11,k} & Q & B_{13,k} & Q \\ B_{21,k} & Q & B_{23,k} & Q \\ Q & B_{32,k} & Q & B_{34,k} \\ Q & B_{42,k} & Q & B_{44,k} \end{bmatrix}$$

Q 는 상태천이가 없음을 나타내며, column 은 k 시점의 각 상태에서 나가는 branch metric , row 는 k+1 시점의 각 상태로 들어오는 branch metric 을 의미한다. 그리고 우리는 식 (3) 을 이용하여 다음을 유도할 수 있다.

$$S_{k+2} = A_{k+1} \otimes S_{k+1} \quad (4)$$

$$= A_{k+1} \otimes (A_k \otimes S_k)$$

이를 더 일반적으로 나타내면 다음과 같다.

$$S_{k+M} = {}_MA_k \otimes S_k \quad (5)$$

$$({}_MA_k = A_{k+M-1} \otimes \cdots\cdots A_{k+1} \otimes A_k)$$

즉, k 부터 k+M 까지의 branch metric 들간의 합을 구한 후 k 시점의 state metirc 과 더하면 k+M 시점 의 state metirc 을 구할 수 있다. 그러므로, 매 iteration 마다 state metric 을 update 해야하는 번거로움이 없어지는 것이다. 이는 직관적으로 봐도 속도 향상에 큰 도움이 됨을 알 수 있다. 그러나 그림3 에서 볼 수 있듯 이 M-step ACSU 블록에서는 여전히 M 단계마다 state metric 의 update 를 위해 feedback operation이 존재한다. 이는 minimized method 알고리즘을 통하여 해결될 수 있다.

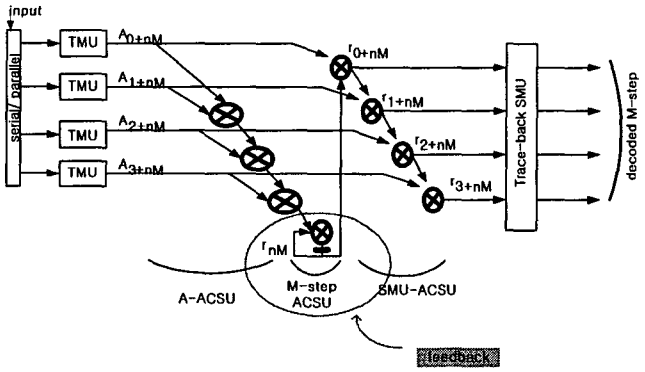


그림 3 *Structure of the M-step look ahead technique*

Minimized Method

Survivor depth(decoding depth) 는 보통 K(constraint length) 의 5 배를 취한다. 그리고 5k 이상의 depth 를 decoding 했을 때 유일한 경로를 얻을 수 있다. 그러므로 M 을 5k 이상으로 취해서 M-step look ahead technique을 적용하면 하나의 state 에서 나가는 branch metric 들이 다른 모든 state 와 연결된다. (그림4) 이 때, k 시점의 decoded state 의 metric 값이 'a' 라고 했을 때, k+M 시점의 state metrics 는 모두 공통적으로 'a' 라는 값을 포함하고

있다. 그러므로 k 시점의 state metric 은 normalization 관점에서 봤을 때 고려되지 않아도 된다. 이는 k+M 시점의 모든 st ate metric에서 k 시점의 state metric 을 뺀다하더라도 어떤 에러도 발생하지 않기 때문이다. 그러므로, k+M 시점의 path metric 값은 k부터 M까지의 branch metric 들의 합이 되는 것이다. 이러한 branch metric 값들은 M-step look ahead technique 에서 살펴봤듯이, feedforward 이므로 pipeline 방식으로 구현될 수 있다. 즉, M-step 동안의 branch metric 값들을 더하는 것이 M-step ACSU 블록에서의 state metric 을 update 하는 것과 동일한 것이 되므로 M-step ACS 부분에서의 feedback 까지도 완전히 사라지게 되는 것이다. 또 한, k+M 시점에서의 decision 값들은 모두 k시점의 임의의 한 state 를 가리키는 동일한 결과를 갖는다. 그러므로, k+M 시점에서의 임의의 한 state 에서만 decision 값을 얻어내면 k 시점의 decoding 시작 state 를 알 수 있다. 이는 k 시점의 임의의 한 column 으로부터 M-step operation 을 수행한 후, maximum state metric 값을 갖는 state 를 찾아내는 것이고, 다음 식으로 나타낼 수 있다.

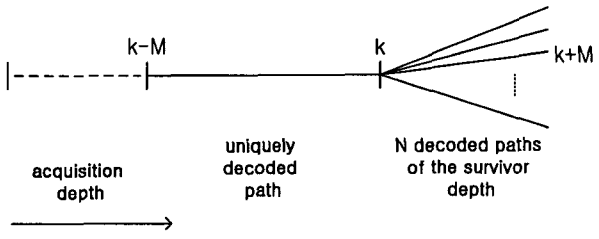


그림 4 scheme of decoded path

$$S_{k+M} = \text{column}_J({}_M A_k) \quad (6)$$

$$= A_{k+M-1} \otimes \cdots\cdots \otimes A_{k+1} \otimes \text{column}_J(A_k)$$

$$= {}_M A_k \otimes S_k = {}_M A_k \otimes \text{column}_J({}_M A_{k-M})$$

위에서 언급했듯이 k+M 시점의 decision 값은 하나의 state 에서만 구하면 되고, 이는 다음과 같이 간략화된 연산을 통해 구할 수 있다.

$$\text{row}_K({}_M A_k) \otimes \text{column}_J({}_M A_{k-M}) \quad (7)$$

$$\text{row}_K({}_M A_k) = \text{column}_K({}_M A_k{}^T)$$

$$= A_k{}^T \otimes \cdots \otimes A_{k+M-2}{}^T \otimes \text{column}_K(A_{k+M-1}{}^T)$$

${}_M A_k$ 의 row 는 matrix 의 transpose 를 통해 column 으로 전환할 수 있고, 식 (7) 의 연산과정은 서로 방향만 틀릴 뿐 동일하다. Row operation 은 오른쪽에서 왼쪽으로 decoding 을 하는 것이고, column operation 은 왼쪽에서 오른쪽으로 decoding 을 행하는 것이다. 그래서 이들이 만나는 지점에서 각각의 값을 더한 후, 최대값을 갖는 것이 구하고자 하는 decoding 시작 s tate 가 되는 것이다. (그림 5)

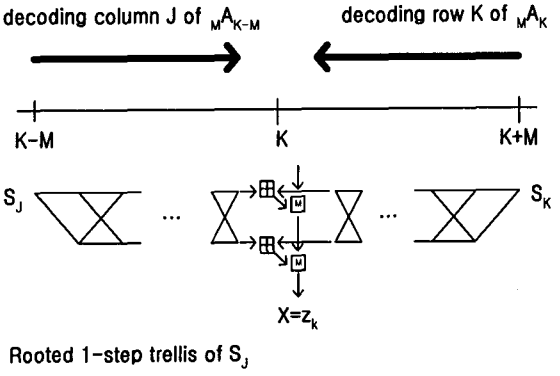


그림 5 forward and backward decoding scheme

그리고 이 렇게 얻어 진 state 로부터 다시 decoding 을 시작하면서 decision vector 를 구하고, trace-back 을 이용하여 de coded bit 를 얻을 수 있다. (그림 6)

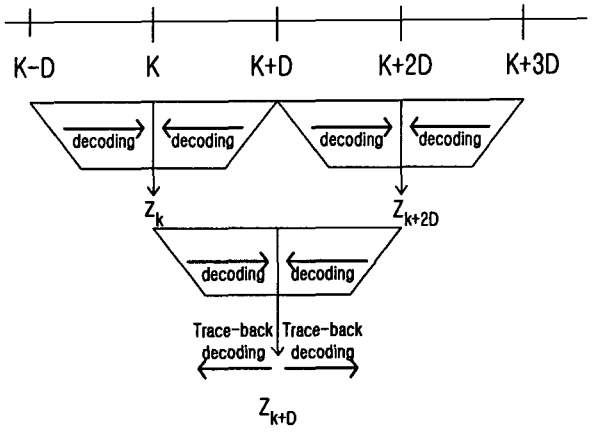


그림 6 scheme of minimized method

Improvement of minimized method

우리는 M-step look ahead technique 에서 제시한 기본 PE(Processing Element) 와 code optimized array를 이용하여 minimized method 의 core block 을 비트단위로 설계하 였다.

Modified Processing Element

Carry save number (0,1,2) 를 사용함으로써 발생하는 redundant bit(cs number 1 을 표현할 때, 01,10 두 가지 경우가 존재한다.) 를 [6]에서 제시한 code converter 를 이용하여 제거하였다. 그리고 CSM 블럭간의 state 를 나타내는 정보 비트를 3bits 로 설정함으로써 최 하위 CSM 에서 나가는 state 정보의 msb 만으로 크기를 비교할 수 있는 decision block 을 추가하였다.(그림 7,8,9)

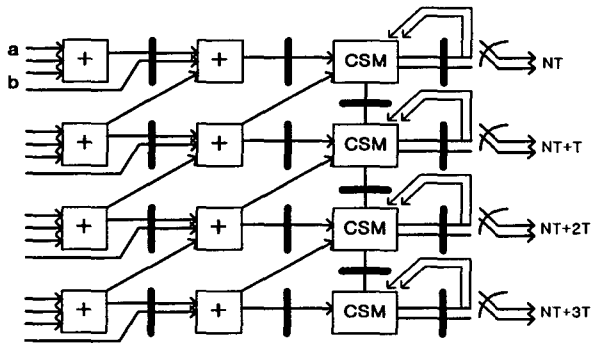


그림 7 proposed processing Element

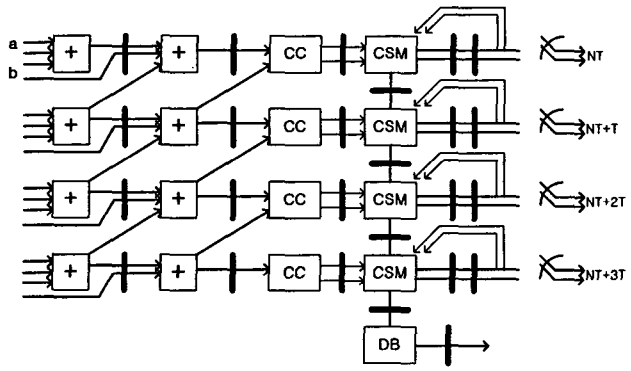
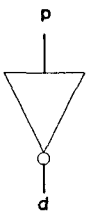


그림 8 Modified Processing Element

(CC: code converter, 4bit level)

state	Presentation bit (p)	Decision value (d)
A = B	000	1
Predecision A	101	0
A > B	110	0
Predecision B	001	1
A < B	011	1

(A)



(B)

그림 9 (A) Truth table (B) logic

(p: msb of presentation bit, d: decision value)

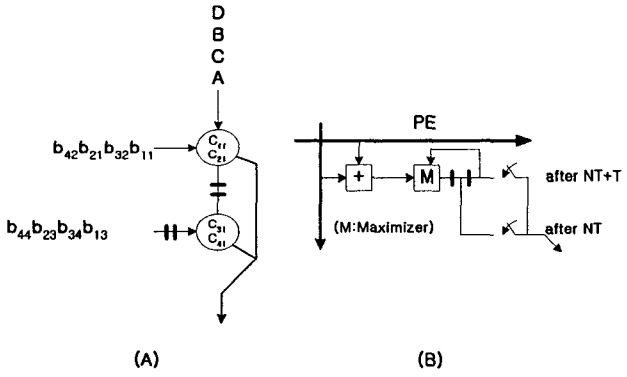
Retimed Code Optimized Array

Minimized method 알고리즘에서 forward 방향으로 decoding 을 진행할 경우 [5]에서 제시한 code optimized array 를 이용할 수 있다. 그러나 backward 방향으로 진행할 경우 이를 그대로 적용할 수 없다. 그래서 우리는 transposed matrix 의 column 의 위치를 바꾸고, vector 부분의 row 위치를 바꿈으로써 backward 방향에도 code optimized array 를 적용할 수 있도록 하였다.(그림10,11)

$$\begin{bmatrix} B_{11,k} & B_{21,k} & Q & Q \\ Q & Q & B_{32,k} & B_{42,k} \\ B_{13,k} & B_{23,k} & Q & Q \\ Q & Q & B_{34,k} & B_{44,k} \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix}$$

$$\begin{bmatrix} B_{11,k} & Q & B_{21,k} & Q \\ Q & B_{32,k} & Q & B_{42,k} \\ B_{13,k} & Q & B_{23,k} & Q \\ Q & B_{34,k} & Q & B_{44,k} \end{bmatrix} \begin{bmatrix} A \\ C \\ B \\ D \end{bmatrix}$$

그림 10 Switched matrix-vector



(A) (B)

그림 11 (A) Modified code optimized array (B) its PE realization

Normalization

Carry save number 를 사용할 때, state metric 의 msb 가 '2' 이면, 다른 모든 state metric 에서 동시에 '1' 을 빼줌으로써 normalization 을 매우 간단히 해결할 수 있는 것은 매우 훌륭한 기법이다.[2] 그러나 [7]에서 제시한 방법에 의해 우리는 normalization 블록을 생략했다. 이는 당연히 hardware complexity 를 줄여준다.

$$S_{bits} = [\log_2(2\Delta_{max})] + 1bit \quad (8)$$

Δ_{max} 는 state metric 의 dynamic range 이며 다음과 같이 주어진 다.

$$\Delta_{max} \le \lambda_{max} \log_2(N) \quad (9)$$

λ_{max} 는 maximum branch metric 이며, N 은 state number 이다. 우리의 경우, N=4 states, 4-bit soft decision 이므로 요구되는 state metric 의 비트수는 8 bits 였다.

Conclusions

우리는 [4], [5] 에서 제시된 방법을 기초로 modified minimized method 의 core block 을 bit-level 로 설계하였다. (K=3, R=1/2, 4-bits soft decision) SYNOPSYS 의 Designcompiler 와 TSMC 0.18um library 를 이용하여 합성을 하였으며, CADENCE 의 Verilog-XL 을 이용하여 timing simulation 을 검증하였다. 합성 주파수는 모두 200 Mhz 로 설정하였다. 표1 에 합성결과후 gate counts 를 제시하였다. 그림 6 의 구조 에서 임의의 state 로부터 maximum state (Z_k , Z_{k+2D}) 를 찾아가는 과정을 Level 1, Level 1 에서 찾은 state 를 바탕으로 Z_{k+D} 를 찾아가면서 decision vector 를 얻어내는 과정을 Level 2 로 하기로 한다. Code optimized array 를 이용했을 때 새로운 하나의 vector 를 출력하기 위해 필요한 PE 의 갯수는 2개이며, 그렇지 않을 경우 4개의 PE 가 필요하다. 그리고 NT 라는 시간마다 데이터를 출력하기 위해서 K=3 일 때, survivor depth 가 10 이므로 10개의 code word 를 계산하기 위한 9개의 stage 가 필요하다. 그러므로 Backward 방향의 PE 에 code optimized array 를 적용 함으로써 9 stage 동안 약 46,000 gates(9*2*2602) 의 area 를 줄일 수 있음을 짐작할 수 있다. 실제로 우리는 이를 검증하기 위해 Level 1 의 backward 방향에 대하여 code optimized array 를 사용하지 않고, 합성을 해보았다. 그 결과 표 2 에서 보는 바와 같이 약 35,000 gates 의 차이가 났다. 예상보다 적은 수의 gate 차이가 났는데 이는 modified code optimized array 부분에서 timing 을 맞추기 위해 B 의 값을 한 클럭 더 유지시켜주기 위해 사용된 레지스터 때문일 것이다. 그러나 분명 많은 수의 gate 가 줄어들었 음을 확인할 수 있다.

표 1 area after code optimized array

	Forward	Backward
Level 1	87,217	86,460
Level 2	89,437	88,566
PE(Level 1)	2,891	2,602

표 2 area before code optimized array

	Backward
Level 1	121,640

References

[1] A. J. Viterbi, "Error bounds for convolutional coding and an asymptotically optimum decoding algorithm," *IEEE Trans. Inform. Theory*, vol.IT-13, pp. 260–269, Apr. 1967.

[2] G. Fettweis and H. Meyr, "A 100 Mbit/s Viterbi decoder chip: Novel architecture and its realization," in *Proc. IEEE Int. Conf. Commun.*

[3] G.Fettweis and H.Meyr, "Feedforward architecture for parallel Viterbi decoding"*J.VLSI Signal Processing*,vol.3,pp.105-119,1991.

[4] G.Fettweis, H.Dawid, and H.Meyr,"Minimized method Viterbi decoding: 600 Mb/s per chip," in *proc. GLOBECOM 90*, vol.3,Dec. 1990,pp.1712-1716

[5] G.Fettweis, H.Meyr, " High rate Viterbi processor : A Systolic array solution," IEEE J.SAC, Oct. 1990.

[6] V. S. Gierenz, O.Weiss, T. G. Noll, I. Carew, J. Ashley, and R. Karabed,"A 550 Mb/s radix-4 bit-level pipelined 16-state 0.25-_m CMOS Viterbi decoder," in *Proc. IEEE Int. Conf. Application-Specific Systems, Architectures, and Processors*, 2000, pp. 195–201.

[7] P. J. Black and T. H.-Y. Meng, "A 1-Gb/s, four-state,sliding block Viterbi decoder," *IEEE Journal of Solid-State Circuits*, vol. 32, no. 6, pp. 797–805, June 1997.

[8] Je-Hyuk Ryu and Jun-Dong Cho, "Low Power Systolic Array Viterbi Decoder Implementation with a Clock-gating Method", Vol. 12-A No. 1, *Korea Information Processing Society*, pp. 1-6, feb 2005