論文94-31A-8-12

# 비터비 디코더의 성능 향상을 위한 역추적 알고리듬의 설계

# (Design of Traceback Algorithm for Performance Improvement in Viterbi Decoder)

黃 義 俊*, 李 種 和*, 林 信 一*, 黃 善 泳*

(Ei Jun Hwang, Jong Hwa Lee, Shin Il Lim and Sun Young Hwang)

要 約

본 논문은 Viterbi 알고리듬의 병렬 처리 하드웨어 구현을 위한 효율적인 역추적 방법을 제안한다. 역추적의 초기 상태를 임의적으로 선택하는 기존의 Viterbi 알고리듬에 비하여 제안된 알고리듬은 연속적인 역추적에 의하여 얻어진 경로를 분석하여 디코딩 출력을 결정한다. 제안된 알고리듬은 생존자가 하나 이상 존재하는 경우에도 효율적으로 에러를 정정할 수 있다. 제안된 알고리듬의 하드웨어의 구현을 위한 블럭도를 제시한다.

실험 결과를 통하여 잡음이 있는 채널에서 제안된 알고리듬은 기존의 알고리듬에 비하여 디코딩 길이에 따라 에러의 정정 효율이 최소 50% 이상 증가되었음을 확인하였다.

**Abstract**

This paper proposes an efficient traceback method for parallel hardware implementation of the Viterbi algorithm. Compared to the conventional Viterbi algorithm where initial state for traceback is selected arbitrarily, the proposed algorithm decides decoding output by analyzing the survivor paths of consecutive tracebacks. This makes Viterbi algorithm more efficient in error correction, even when more than one survivor path exists. The proposed traceback algorithm together with its hardware realization is presented in this paper.

Experimental results show that the proposed algorithm is efficient in error correction in noisy channels, compared to the existing algorithms.

*正會員, 西江大學校 電子工學科 CAD 및 Computer System 研究室
(CAD and Computer System Lab. Dept. of Elec. Eng., Sogang Univ.)

接受日字 : 1993年 7月22日

## Ⅰ. 서론

기존의 아날로그 통신 시스템은 위성 통신이나 이동 통신 등에서 채널 용량의 한계성과 정보의 신뢰성에서 문제를 드러내었으며 이에 디지탈 통신 시스템이 대두되었다. 디지탈 통신 시스템은 채널 용량 한

계의 극복과 정보의 신뢰성을 보장하기 위하여 각각 아날로그 신호를 이산화하는 소스 코딩과 채널상의 에러를 정정하기 위한 채널 코딩의 과정을 거쳐 데이타를 전송한다.[4] 소스 코딩의 방식으로 Huffman Coding, Run Length Coding, Quantize and Noiseless Coding 등의 잡음이 없는 소스 코딩(noisless source coding)과 DPCM (Differential Pulse Coded Modulation), Waveform Tree Codes 등의 예견 소스 코딩(predictive source coding)이 제안되었다.[1] [2] 채널상에서 발생하는 에러의 정정을 위한 채널 코딩은 크게 메모리를 필요로 하지 않는 블럭 코딩과 메모리를 필요로 하는 콘볼루션 코딩으로 구분된다.[1] [2] 블럭 코딩은 parity를 추가하는 방법으로 Linear Block Codes, Cyclic Codes, BCH Codes 등이 제안되었으며[1] [2], 콘볼루션 코딩은 입력 데이타를 메모리에 있는 기존의 데이타와 상관 관계를 갖게 인코딩하는 방법이다.[1] [2]

통신 시스템의 일차적인 목적인 정확한 정보의 전달을 위한 채널 코딩은 디지탈 통신 시스템에서 주요한 연구의 대상이며, 블럭 코딩에 비하여 에러 정정 효율이 우수한 콘볼루션 코딩이 주로 사용되고 있다. 콘볼루션 코딩은 디코딩시에 많은 양의 메모리를 필요로하며 디코딩 지연이 발생하기 때문에 실시간 처리가 가능한 디코딩 알고리듬이 필요하게 되었다. 이에 콘볼루션 코딩의 최대 유사 디코딩 알고리듬인 Viterbi 알고리듬이 제안되었고[5] [6] [7], 데이타의 실시간 처리를 위하여 일정량의 디코딩 길이(truncation length)를 갖는 병렬처리 하드웨어가 설계되어 사용되고있다.[8] [9] [10] [11] 디코딩 길이는 에러의 정정 효율과 하드웨어 구현 효율에 동시에 여향을 미치며[12] [13] [14], 최대 유사 디코딩 출력을 결정하기 위해 많은 양의 데이타를 사용하여야 하므로 디코딩 길이는 길어야하나 실시간 처리를 위한 하드웨어의 구현을 위하여 디코딩 길이는 제한된다.

디코딩 길이의 제한과 더불어 채널 에러의 정정 효율을 감소시키는 원인은 하드웨어 구현상 역추적의 초기상태를 임의로 선택하는 것이며, 생존자가 두개 이상 존재할 때 채널 에러가 발생할 수 있다. 따라서 하드웨어 구현를 고려한 최대 유사 디코딩 출력의 결정을 위한 알고리듬이 필요하다. 제안된 알고리듬은 제한된 디코딩 길이내에서 에러 정정 효율을 증대시키기 위하여 연속적으로 수행되는 역추적(trace-back)에 의해 도달되는 최종 상태를 비교하여 디코딩 출력을 결정하며, 이는 상대적으로 디코딩 길이를 늘이는 것과 동일한 효과를 얻을 수 있다.

본 논문에서는 Viterbi 알고리듬을 이용한 기존의 하드웨어의 구현 방법에 있어서의 문제점을 기술하고 역추적의 도달 상태를 비교하여 디코딩 출력을 결정하는 해결방안을 제시함으로서 효율적인 디지탈 통신 시스템을 가능하도록 한다. 제 2 장에서는 Viterbi 알고리듬과 기존의 하드웨어의 구현 방법의 문제점을 설명하며, 제 3 장에서는 해결방안을 제시한다. 제 4 장에서는 3 장에서 제안된 알고리듬의 하드웨어 구현 방안을 제시하고, 제 5 장과 제 6 장에서는 확률 계산과 실험 결과의 분석을 통한 해결방안의 효율을 검증한다. 마지막 7 장에서 결론과 추후과제를 제시한다.

## Ⅱ. Viterbi 알고리듬과 하드웨어 구현의 문제점

### 1. 콘볼루션 인코딩과 Viterbi 알고리듬

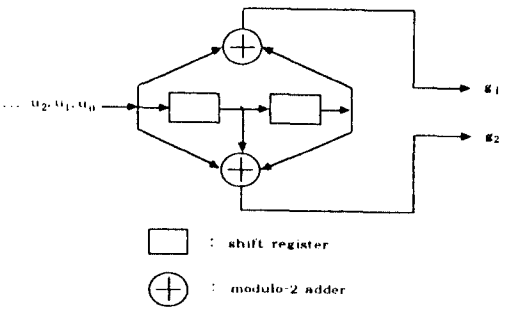



그림 1. R=1/2, K=3인 콘볼루션 인코더
Fig. 1. Convolution encoder with R=1/2, K=3.

콘볼루션 인코더는 그림 1과 같이 쉬프트 레지스터와 modulo-2 덧셈기에 의하여 구현되며, 인코더의 출력신호를 결정하는데 영향을 미치는 입력 수인 제한 길이(constraint length) K와 출력 비트 수에 대한 입력 비트 수의 비인 코드비 R에 의하여 구조가 결정된다.[1] [2] [3] [4]

콘볼루션 인코더는 각각의 덧셈기의 연결점을 생성 벡터 혹은 생성 다항식으로 표현하는 방법과 상태도 혹은 trellis도 등으로 나타내는 방법이 있다.[1] [2] 식 (1)과 식 (2)는 각각 그림 1의 콘볼루션 인코더의 생성 벡터와 생성 다항식을 나타낸다.

$$g_1 = 101_{(2)} \quad g_2 = 111_{(2)} \tag{1}$$

$$g_1 = 1+X^2 \quad g_2 = 1+X+X^2 \tag{2}$$

상태도를 이용한 콘볼루션 인코더의 기술은 콘볼루션 인코더의 레지스터 값을 하나의 상태로 하여 각 입력 값에 따른 상태의 변화를 나타낸 것이며, 상태

의 변화를 이산적인 시간축의 함수로 나타낸 것이 trellis도이다. 그림 2 (a)는 그림 1의 콘볼루션 인코더의 상태도를 나타낸 것으로, 각 에지위의 u/x는 u의 입력에 대한 x의 콘볼루션 출력을 나타낸다. 그림 2 (b)는 그림 1의 인코더의 trellis 도를 나타낸 것으로 각 분기(branch)위의 값들은 인코더의 콘볼루션 출력 $g_1g_2$를 의미한다.

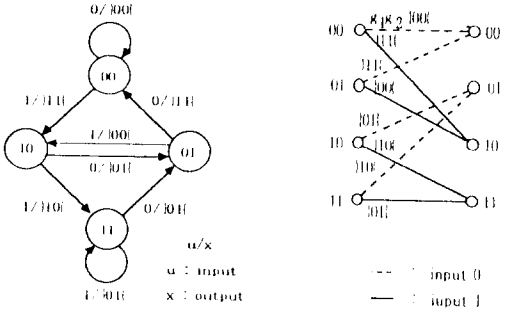



그림 2. 상태도와 trellis 도의 예
(a) 상태도 (b) trellis 도

Fig. 2. An example (a) State diagram (b) Trellis diagram.

최대 유사 디코딩(Maximum Likelihood Decoding)은 주어진 입력값에 대한 가장 유사한 형태를 찾아 디코딩 출력을 결정하는 방식으로, 최적의 디코딩 효율을 얻을 수 있음이 보고되었으며[5] [6] [7], Viterbi 디코딩 알고리듬은 콘볼루션 코딩의 최대 유사 디코딩 알고리듬으로 알려져있다. Viterbi 디코딩 알고리듬은 디코더의 입력과 인코더에 의하여 주어지는 trellis도의 기준값과의 Hamming distance인 branch metric를 누적한 path metric를 이용하며, 각 상태에서의 path metric은 들어오는 두개의 분기중에서 전 단계에서의 path metric과 branch metric를 합한것 중 작은 것을 선택하여 얻어진다. 선택된 분기를 생존자(survivor)라고하며 가장 작은 path metric를 갖는 상태에서 시작하여 생존자를 역추적함으로써 디코딩 출력을 결정한다.[5] [6]

그림 3은 Viterbi 알고리듬에 의하여 디코딩 출력을 결정하는 예를 보여주고 있다. 각 분기위의 숫자는 branch metric를 의미하며, 각 상태에 괄호안의 수는 path metric를 의미한다. 그림 3은 디코딩 시간의 흐름에 따른 변화를 나타내며, 입력 $g_1g_2$는 각각 {00}, {10}, {11}, {01}이다. 디코딩 출력은 그림 3 (d)에서 최소의 path metric를 갖는 상태 2에서 굵은 선의 생존자를 역추적함으로서 결정된다.

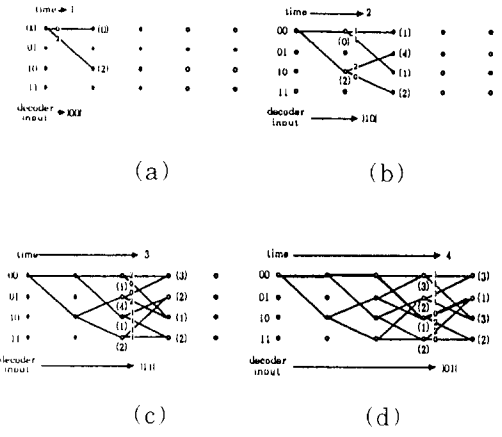



그림 3. Viterbi 알고리듬에서 생존자의 결정과 디코딩 출력의 결정

Fig. 3. Decision of survivor path and decoding output in Viterbi algorithm.

2. Viterbi 디코딩 알고리듬의 하드웨어 구현과 문제점

Viterbi 디코딩 알고리듬을 구현한 하드웨어는 그림 4에 보인바와 같이 크게 branch metric 생성, state metric update(SMU) 그리고 survivor update의 세 부분으로 구성된다.[9] [10]

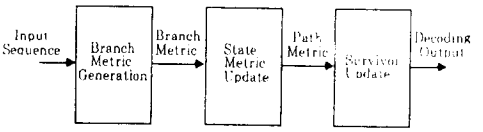



그림 4. Viterbi 디코딩 알고리듬의 하드웨어 블럭 구성도

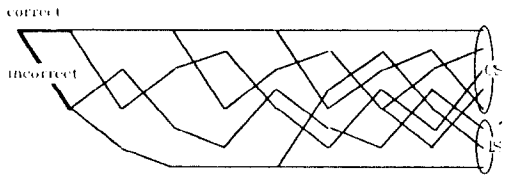
Fig. 4. Hardware block diagram for the Viterbi decoding algorithm.

Branch metric 생성은 인코더에 의한 기준값과 통신 채널을 통해 입력된 데이타간의 Hamming distance를 계산하여 branch metric를 생성하는 블럭이며, SMU은 앞단에서의 branch metric을 각각의 상태에 누적하여 path metric과 상태 변화 정보를 생성하는 블럭이다. 이 두개의 블럭은 콘볼루션 인코딩의 특징인 perfect shuffle exchange 구조를 이용함으로서 능률적으로 구현이 가능하다. Survivor update은 디코딩 성능을 좌우하는 주요 블럭으로써 일반적으로

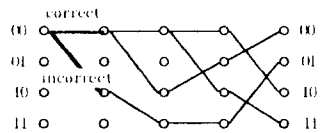
5K의 디코딩 길이의 생존자를 역추적하며[12] [13] [14] [15], Systolic array를 이용한 병렬처리방식을 이용함으로써 매 디코딩 사이클마다 디코딩 출력을 결정할 수 있는 하드웨어가 제안되었다.[9] Survivor update 블럭에서는 최대 유사 디코딩 출력을 결정하기 위하여 역추적의 초기 상태를 최소의 path metric를 갖는 상태로 결정하여야 한다. 그러나 정보의 정확도를 높이기 위하여 K를 7 이상으로 정하며, 이때 trellis 도에서 상태의 수는 64 개 이상이 된다. 따라서 한 디코딩 사이클 동안 최소의 path metric를 갖는 상태를 결정하기 어렵게되어 역추적를 실행하는 초기 상태를 임의적으로 선택하도록 하드웨어를 구현한다.

임의적으로 선택된 역추적의 초기 상태는 디코딩 길이가 충분히 크지 못하면 디코딩 출력을 결정하여야 하는 시점에서 두개 이상의 생존자가 존재하는 경우 디코딩 에러를 양산하게 하는 원인이 된다.

R=1/2인 콘볼루션 코딩의 경우 임의의 상태에서 시작하는 trellis 도는 올바른 분기와 그릇된 분기로 나누어지며, 올바른 분기가 특정의 디코딩 길이를 지난후에 도달되는 상태의 집합을 correct subset(CS)으로, 그릇된 분기가 차지하는 상태의 집합을 incorrect subset(IS)로 정의한다. 그림 5는 CS와 IS의 예이며, (b)는 디코딩 길이가 4일때의 CS와 IS을 보이고 있다.




(a)




(b)

그림 5. Correct subset과 Incorrect subset의 예 (a) 일반예 (b) 특정예

Fig. 5. An example diagram showing correct subset and incorrect subset (a) General example (b) Special example.

하나의 디코딩 출력을 결정하기 위하여 입력되는 데이타가 무한히 많은 경우(디코딩 길이가 무한히 큰 경우) IS는 공집합이되지만 디코딩 길이가 5K로 제한되는 시스템에서는 IS가 공집합이 되지않으며, 역추적의 초기 상태가 IS에서 선택되었을 경우 디코딩된 결과는 에러가된다. 또한 같은 디코딩 길이에 대해서 채널에서 많은 양의 에러가 일어나게 되면 IS의 원소의 갯수는 증가하게 되며, 채널에서의 에러가 50%일때 IS와 CS의 원소의 갯수는 같아지게 된다. 따라서 그림 5와 같이 생존자가 두개 이상 존재할 경우 이를 분석할 수 있어야하며, IS에서 역추적의 초기 상태가 선택되어 에러를 일으킬 경우 이것을 정정할 수 있도록 하여야한다.

## Ⅲ. 디코딩 출력의 결정을 위한 제안된 알고리듬

### 1. 디코딩 출력의 결정 방안과 용어 정의

일반적으로 50%보다 작은 에러율을 갖는 채널에서 IS의 크기는 CS의 크기에 비하여 상당히 작다. 이는 역추적를 수행하는 초기의 상태를 임의로 선택할 경우 연속적으로 IS에서 선택할 확률은 작음을 의미하며, 매 디코딩 사이클마다 이루어지는 역추적의 최종 상태를 분석함으로서 초기에 선택된 상태가 IS에서 선택되었을 경우도 이를 정정할 수 있다.

콘볼루션 인코딩에 있어서 R이 1/2보다 큰 경우는 punctured coding의 방법을 이용함으로서 R=1/2이 되도록 할 수 있으며[15], 이때 trellis 도의 모든 상태에서 발생하는 분기는 두개가 된다. 이 두개의 분기는 trellis 도의 상태사이에 분기의 전 상태의 LSB와 분기의 다음 상태의 MSB를 제외한 나머지 비트가 같은 관계를 유지하면서 이루어진다. 즉 000 상태에 연결된 분기는 100 상태와 000 상태만이 가능하다. 연속적으로 이루어지는 두번의 역추적의 최종 상태는 분기에 의하여 연결되어야하며, 특정의 분기에 의하여 연결되지 않았다면 두번의 역추적은 다른 경로를 탐색하였음을 나타낸다. 즉, i번째의 역추적에 의하여 도달된 최종 상태를 s라고 한다면 s에서 발생하는 두개의 분기는 특정의 상태 $s_1$, $s_2$가 되며, i+1번째의 역추적이 도달하는 최종 상태가 $s_1$ 혹은 $s_2$상태가 아니라면 i, i+1번째의 역추적은 다른 경로를 탐색하였음을 의미한다. 이는 생존자가 두개 이상 존재함을 의미하며, 역추적의 최종 도달 상태를 분석함으로서 생존자의 상황을 알 수 있고 에러의 정정 효율을 높일 수 있음을 나타낸다.

본 논문에서 사용되는 용어는 아래와 같이 정의된다.

$S_f^i$ : i 번째 역추적 과정에서 도달되는 최종 상태

$S^i_{f+1}$ : i 번째 역추적 과정에서 도달되는 최종 상태의 전 상태

TD : 역추적 경로의 검토를 위한 Transition Detector

TD1 - $S^i_f$ , $S^{i+1}_f$ 사이에 경로가 존재하는가를 조사한다.
$S^i_f$ 의 LSB와 $S^{i+1}_f$의 MSB를 제외한 나머지 비트를 비교함으로서 필요한 정보를 얻을 수 있다.

TD2 - $S^i_{f+1}$, $S^{i+1}_f$를 비교하여 i 번째 역추적에 의한 경로와 i+1 번째 역추적에 의한 경로가 동일한 가를 조사한다.

$O^i_f$, $O^i_{f+1}$ : TD1, TD2의 출력

$DO^i$ : i번째 역추적에 의한 디코딩 출력의 결정되어지는 스테이지

2. TD1,TD2의 결과를 이용한 분석

Transition Detector TD1과 TD2의 출력으로 얻어진 $(O^i_f, O^i_{f+1})$를 분석함으로서 표 1과 같은 의미를 얻을 수 있다. 표 1은 $(O^i_f, O^i_{f+1})$의 결과에 따른 존재 가능한 생존자의 상황을 분석한 것으로 (1,1)인 경우는 $DO^i$까지 생존자가 하나로 존재할 가능성이 높음을 의미하며, (1,0)인 경우는 $DO^i$에서 생존자가 두개로 갈라짐을 의미한다. (0,0)은 생존자가 $DO^i$보다 이전에 두개로 갈라졌음을 의미한다.

표 1. $(O^i_f, O^i_{f+1})$의 분석

Table 1. Analysis of the values of $(O^i_f, O^i_{f+1})$.

| $O^i_f$ | $O^i_{f+1}$ | 의 미 |
|---|---|---|
| 1 | 1 | 두번의 역추적이 같은 경로를 탐색하였음을 의미한다. 이는 확률적으로 생존자가 하나 존재함을 의미한다. |
| 1 | 0 | 두번의 역추적이 같은 경로를 탐색한것은 아니며, 생존자 두개 존재할 수 있다. 그러나 $DO^i$전단에서는 생존자가 하나있다는 것을 확률적으로 높여준다. |
| 0 | 1 | 절대로 일어날 수 없는 상황이다. |
| 0 | 0 | 두번의 역추적이 다른 경로를 탐색하였음을 의미하며, 생존자가 여러개 존재할 수 있다. |

그림 6은 예를 보여주고 있으며, $DO^i$는 두번째 스테이지에 해당한다. 그림 6 (a)는 생존자가 계속 하나로 존재하는 경우로 $(O^i_f, O^i_{f+1})$ = (1,1)일때의 상황이다. $O^i_f$ = 1은 $S^{i+1}_f$는 $S^i_f$에서의 두개의 분기중에서 어느 하나에 연결되었음을, $O^i_{f+1}$ = 1은 $S^{i+1}_f$ 와 $S^i_{f+1}$이 같음을 의미하므로 $S^i_f$에서의 두개의 분기중에서 i, i+1 번째의 역추적은 같은 분기를 탐색하였음을 의미한다. 그림 6 (b)는 하나로 존재하던 생존자가 $DO^i$에서 두개로 갈라지는 경우로 $(O^i_f, O^i_{f+1})$ = (1,0)일때의 상황이다. $O^i_{f+1}$ = 0이므로 $S^{i+1}_f$와 $S^i_{f+1}$은 다른 상태이지만 $O^i_f$ = 1이므로 $S^{i+1}_f$는 $S^i_f$에서의 두개의 분기중 다른 하나와 연결되었음을 의미한다. 그림 6 (c)는 $DO^i$이전 부터 생존자가 두개이상 존재하는 경우로 $(O^i_f, O^i_{f+1})$ = (0,0)일때의 상황이다. $O^i_f$=0이므로 $S^{i+1}_f$는 $S^i_f$에서의 두개의 분기중 어떤 분기와도 연결이 되지 않았음을 의미하며 이는 생존자가 $DO^i$이전 부터 두개이상 존재하였음을 의미한다.

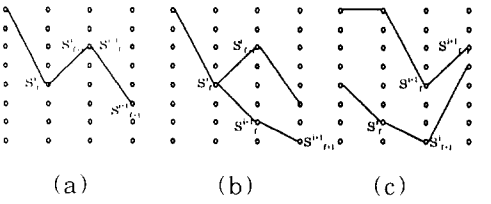



그림 6. Transition Detector 출력에 따른 생존자정보
(a) $(O^i_f, O^i_{f+1})$ = (1,1) (b) $(O^i_f, O^i_{f+1})$ = (1,0) (c) $(O^i_f, O^i_{f+1})$ = (0,0)

Fig 6. Survivor information obtained by transition detector outputs.
(a) $(O^i_f, O^i_{f+1})$ = (1,1) (b) $(O^i_f, O^i_{f+1})$ = (1,0) (c) $(O^i_f, O^i_{f+1})$ = (0,0).

3. 예를 통한 생존자의 상황 분석

TD의 결과중에서 $(O^i_f, O^i_{f+1})$, $(O^{i+1}_f, O^{i+1}_{f+1})$를 판단함으로서 생존자의 상황 분석을 살펴 보기로한다. 표 2는 가능한 모든 경우를 보여주며 각각에 대하여 가능한 예를 그림 7의 trellis 도로 표현하였다. 각 분기위의 첨자i는 i번째의 역추적이 그 분기를 경과함을 표시하며 실선은 확실한 경로이고 점선은 추정에 의한 경로이다. (이후 모든 그림에서 표시방법은 동일하다.)

표 2는 표 1에서 설명한 것을 두번 적용함으로서 의미를 분석할 수 있다. 예를 들어 $\{(O^i_f, O^i_{f+1}), (O^{i+1}_f, O^{i+1}_{f+1})\}$ = {(1,1), (1,1)}인 경우는 앞의 (1,1)에 의하여 생존자는 하나이며, 뒤의 (1,1)에 의해서도 생존자는 하나이다. 따라서 이를 연결하면 그림 7 (a)와 같이 된다. 또 $\{(O^i_f, O^i_{f+1}), (O^{i+1}_f, O^{i+1}_{f+1})\}$ = {(1,0), (1,0)}인 경우는 앞의 (1,0)에 의하면 생존자는 두개로 갈라지고 뒤의 (1,0)에 의해서 또다시 두개로 갈라지며 그림 7 (e)와 같이 된다.

또 $\{(O^i_f, O^i_{f+1}), (O^{i+1}_f, O^{i+1}_{f+1})\}$ = {(0,0), (0,0)}인 경우는 앞의 (0,0)에 의하면 생존자가 두개 존재하며, 뒤의 (0,0)에 의해서도 두개의 생존자가 존재함으로 이를 적절히 연결하면 그림 7 (i)와 같이 추정이

가능하다.

표 2. 생존자의 상황 분석
Table 2. Analysis of survivors.

| $(O^{i-1}_f, O^{i-1}_{f+1})$ | $(O^{i+1}_f, O^{i+1}_{f+1})$ | 의 미 |
|---|---|---|
| (1,1) | (1,1) | i-1, i, i+1의 세번의 역추적이 동일한 경로를 지나갔으며 생존자는 하나 존재할 가능성이 높다. (그림 7 (a)) |
| (1,1) | (1,0) | 하나로 존재해오던 생존자가 두개로 나뉘어진다. (그림 7 (b)) |
| (1,1) | (0,0) | 두개 이상의 생존자가 계속 존재해왔음을 나타낸다. (그림 7 (c)) |
| (1,0) | (1,1) | 생존자가 전단에서 두개로 갈라졌으며, 현재단에서는 최소한 두개의 생존자가 존재한다. (그림 7 (d)) |
| (1,0) | (1,0) | 생존자가 전단에서 두개로 갈라졌으며, 현재단에서는 최소한 세개의 생존자가 존재한다. (그림 7 (e)) |
| (1,0) | (0,0) | 생존자가 전단에서 두개로 갈라졌고 현재단에서는 최소한 두개의 생존자가 존재하며 추정에 의해 경로를 결정할 수 있다. (그림 7 (f)) |
| (0,0) | (1,1) | 전단에서 부터 완전히 다른 두개의 생존자가 존재하여왔다. 최소한 두개의 생존자가 존재한다. (그림 7 (g)) |
| (0,0) | (1,0) | 완전히 다른 두개의 생존자가 현재단에서 최소한 세개 이상으로 늘어난다. (그림 7 (h)) |
| (0,0) | (0,0) | 생존자가 두개 이상이 존재하지만 분석하기 어렵다. (그림 7 (i)) |

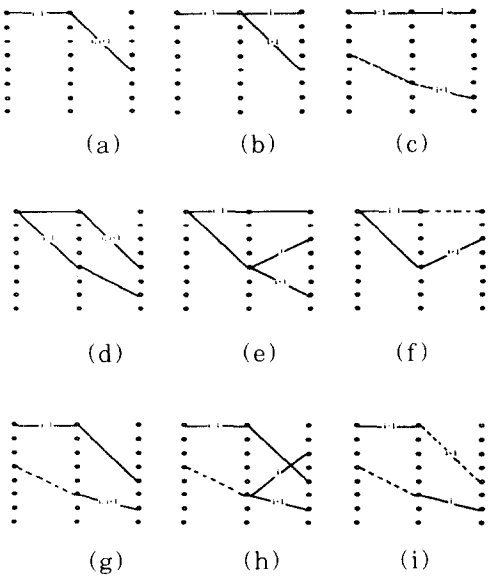



그림 7. 역추적 결과에 따른 생존자의 상황 예
Fig 7. Survivors according to traceback results.

4. TDs를 이용한 디코딩 출력의 결정

에러의 확률을 줄이면서 최적의 디코딩 출력을 얻기위해서는 TD 출력에 의한 충분한 양의 정보가 필요하며 TD 출력의 양의 결정은 생존자의 대략적인 상황 분석과 하드웨어 구현 효율을 고려하여 이루어져야 한다. 이절에서는 i번째 역추적에 의하여 디코딩 출력을 얻기위하여 i-1, i, i+1 그리고 i+2의 네번의 역추적에 의한 TD 출력을 이용하였다. 그림 8은 디코딩 출력을 결정하기위한 방법의 예이다.

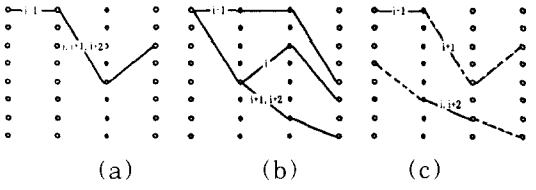



그림 8. $\{(O^{i-1}_f, O^{i-1}_{f+1}), (O^i_f, O^i_{f+1}), (O^{i+1}_f, O^{i+1}_{f+1})\}$를 이용한 분석
(a) {(1,1), (1,1), (1,1)} (b) {(1,0), (1,0), (1,0)} (c) {(0,0), (0,0), (0,0)}
Fig. 8. Analysis using $\{(O^{i-1}_f, O^{i-1}_{f+1}), (O^i_f, O^i_{f+1}), (O^{i+1}_f, O^{i+1}_{f+1})\}$.
(a) {(1,1), (1,1), (1,1)} (b) {(1,0), (1,0), (1,0)} (c) {(0,0), (0,0), (0,0)}.

표 3. 디코딩 출력의 결정
Table 3. Decision of decoding output.

| $\{(O^{i-1}_f, O^{i-1}_{f+1})$ $(O^i_f, O^i_{f+1})$ $(O^{i+1}_f, O^{i+1}_{f+1})\}$ | 디코딩 출력의 결정 |
|---|---|
| {(1,1), (1,1), (X,X)} | i번째 역추적 |
| {(1,1), (1,0), (1,X)} | i+1번째 역추적 |
| {(1,1), (1,0), (0,0)} | i번째 역추적 |
| {(1,1), (0,0), (1,X)} | i+1번째 역추적 |
| {(1,1), (0,0), (0,0)} | i번째 역추적 |
| {(1,0), (1,1), (X,X)} | i번째 역추적 |
| {(1,0), (1,0), (1,X)} | i+1번째 역추적 |
| {(1,0), (1,0), (0,0)} | i번째 역추적 |
| {(1,0), (0,0), (1,X)} | i+1번째 역추적 |
| {(1,0), (0,0), (0,0)} | i번째 역추적 |
| {(0,0), (1,1), (X,X)} | i번째 역추적 |
| {(0,0), (1,0), (1,X)} | i+1번째 역추적 |
| {(0,0), (1,0), (0,0)} | i번째 역추적 |
| {(0,0), (0,0), (1,X)} | i+1번째 역추적 |
| {(0,0), (0,0), (0,0)} | i번째 역추적 |

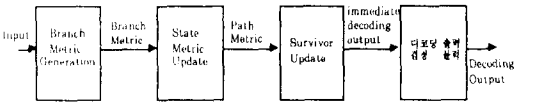
그림 8 (a)는 $\{(O^{i-1}_f, O^{i-1}_{f+1}), (O^i_f, O^i_{f+1}), (O^{i+1}_f, O^{i+1}_{f+1})\}$ = {(1,1), (1,1), (1,1)}인 경우를 보여주며 이상의 정보로서는 생존자가 하나 존재할 가능성이 상당히 높다. 따라서 디코딩 출력은 i번째 역추적에 의하여 결정되면 타당하다. 그림 8 (b)는 $\{(O^{i-1}_f, O^{i-1}_{f+1}), (O^i_f, O^i_{f+1}), (O^{i+1}_f, O^{i+1}_{f+1})\}$ = {(1,0), (1,0), (1,0)}인 경우로 생존자는 여러개 존재하는데 i+1번째 역추적에 의한 경로가 가장 많이 탐색되므로 i+1번째의 역추적에

의하여 디코딩 출력이 결정되어야 한다. 그림 8(c)는 $\{(O^{i-1}_f, O^{i-1}_{f+1}), (O^i_f, O^i_{f+1}), (O^{i+1}_f, O^{i+1}_{f+1})\} = \{(0,0), (0,0), (0,0)\}$인 경우로 생존자의 상황을 분석하기가 가장 어려운 경우이다. 따라서 이런 경우는 디코딩 출력을 결정하기는 어렵지만 그림과 같은 상황이 기능성이 높으므로 디코딩 출력은 i번째 역추적에 의하여 결정되어진다.

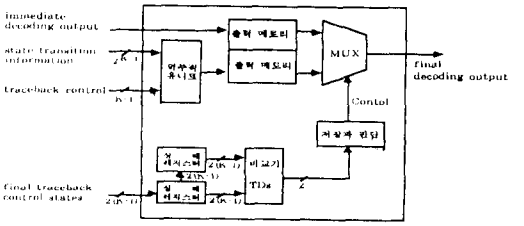
이와 같이 $\{(O^{i-1}_f, O^{i-1}_{f+1}), (O^i_f, O^i_{f+1}), (O^{i+1}_f, O^{i+1}_{f+1})\}$의 결과를 표 1에서 설명된 것을 기초로하여 적절히 연결함으로서 디코딩 출력을 확률이 높은 것으로 결정할 수 있다. 표 3에 모든 경우의 디코딩 출력의 대략적인 결정 방법을 보여준다.

## Ⅳ. 하드웨어 구현

위에서 기술한 알고리듬을 적용하기 위한 하드웨어의 구성은 그림 9 (a)와 같다. 이는 Ⅱ의 2. 절에서 기술한 하드웨의 구성에 디코딩 출력 결정 블럭이 첨가된 것이며 첨가된 블럭의 내부 구성도를 그림 9 (b)에 제시하였다. 그림 9 (b)에서 출력 메모리는 디코딩 결과를 저장하는 쉬프트 레지스터로 상단 쉬프트 레지스터는 Survivor Update 블럭에서의 디코딩 결과를 그대로 저장하며 하단은 한번의 역추적 유니트를 더 통과한 후의 디코딩 결과를 저장한다. 이와 같은 두개의 디코딩 결과는 MUX에 의하여 선택되어 최종의 디코딩 결과를 출력한다. 상태 레지스터의 상단은 $S^i_f, S^i_{f+1}$를 하단은 $S^{i+1}_f, S^{i+1}_{f+1}$를 각각 저장하여 비교기의 입력으로 사용한다. 비교기의 출력은 적당량이 저장되어 Ⅲ의 4. 절의 디코딩 출력의 결정에서 보여준 방법에 의하여 디코딩 출력의 결정을 위한 선택신호를 MUX의 제어신호로서 만들어 낸다. 비교기는 TD1, TD2로서 구성되며 따라서 비교기의 출력은 한번에 두 비트가 된다. 출력 메모리는 쉬프트 레지스터의 형태로 구현되며 한 비트씩 쉬프트시킨다. Ⅲ의 4. 절에서 제안된 것과 같이 네번의 역추적를 이용하여 디코딩 출력을 결정하는 경우는 3 비트의 출력 메모리를 요구한다. 출력 메모리는 두개가 필요하므로 결국 6 비트이 필요하다. MUX는 2-to-1의 형태로 구현되며 MUX의 입력은 출력 메모리에서 받아들이므로 1 비트가 된다. 상태 레지스터는 역추적에 의하여 도달되는 $S^i_f, S^i_{f+1}$를 저장하는 유니트이기 때문에 2 * (K-1)비트가 요구되며, 4 * (K-1) 비트 레지스터가 된다. 비교기는 TD1과 TD2의 두개의 비교기에 의하여 구성되며 TD1은 상태 레지스터 상단의 $S^i_f$와 하단의 $S^{i+1}_f$를 입력으로 받아들이며, TD2는 상태 레지스터 상단의 $S^i_{f+1}$와 하단의 $S^{i+1}_f$를 입력으로 받아들인다. 그리고 하단의 상태 레지스터은 상단의 상태 레지스터로 전체가 쉬프트되어 다음 디코딩 사이클에서 이용된다. 비교기에 의하여 생성된 결과는 저장된후 4번의 디코딩 사이클에서 얻은 결과를 Ⅲ의 4. 절에서 기술한 디코딩 출력의 결정 방법에 의하여 선택신호를 생성한다.




(a)




(b)

그림 9. 제안된 알고리듬의 하드웨어 구현
(a) 제안된 하드웨어 블럭도
(b) 디코딩 결정 블럭의 내부 블럭도

Fig. 9. Hardware realization of the proposed algorithm.
(a) Hardware block diagram
(b) Internal blocks of decoding output decision unit.

## Ⅴ. 확률의 계산에 의한 검증

참고문헌 [16] 에서 제시한 바와 같이 Output-symmetric 채널에서의 에러의 확률을 계산할때 인코더는 0만을 전달한다고 가정한다. 만약 역추적를 시작하는 초기 상태를 누적된 path metric의 값이 가장 작은 상태로 결정한다면 에러의 확률은 식 (3)과 같다.

$$P_b^{(ff)} \le \sum_{d=d_{free}}^{\infty} \left[ b(d)/k + 1/(2^k - 1) \sum_{s=1}^{2^{m-1}} [2^k - 1 * a_s(d,T) - 1/2 * a_s(d,T+1)] \right] * P_d \quad (3)$$

여기서 $P_d$는 전달된 경로로 부터 거리가 d인 임의의 경로를 디코더가 선택할 확률을 의미하고 b(d)는

0 상태로부터 다시 0 상태로 가는 경로중 중간에 0 상태를 거치지 않았을 때 weight d인 trellis 경로를 만드는 정보 신호의 weight를 모두 합한 것이다. 또 $a_s(d,T)$는 0 상태를 거치지 않고 0 상태로부터 s 상태로 가는 경로로, 길이 T이고 weight d인 trellis 경로들의 수를 의미한다. k는 입력 비트를, $2^m$은 상태수를 각각 의미하며, T는 디코딩 길이(truncation length)를 의미한다.[12] 역추적을 시작하는 초기 상태를 임의로 선택하는 경우는 식 (3)에서의 에러 확률과 같은 에러 확률을 얻기 위해서는 디코딩 길이가 두배가 필요하게된다. 따라서 역추적을 시작하는 초기 상태를 임의로 선택하고 디코딩 길이가 T인 경우의 에러의 확률은 $P_b^{(T/2)}$와 같다.

제 3 장에서 설명한 디코딩 방법을 에러의 확률을 통하여 계산하기 위하여 디코딩 길이를 T라고 가정한다. 또 T는 큰 값을 가지므로 $P_b^{(T/2-1)} = P_b^{(T/2)} = P_b^{(T/2+1)} = P_b^{(T/2+2)} = P_b$라고 가정한다. 디코딩 에러가 발생하는 경우는 4번의 역추적 중에서 4번이 모두 에러이거나 3번이 에러가 일어날 때이다. 또 2번이 에러가 일어날때는 디코딩 에러가 발생할 확률이 1/2이 된다. 따라서 디코딩 에러의 확률은 식 (4)와 같이 표시된다.

$$P_{b,new} = (P_b)^4 +_4 C_3 (P_b)^3 (1-P_b) + 1/2 *_4 C_2 (P_b)^2 = 3(P_b)^2 - 2(P_b)^3 \quad (4)$$

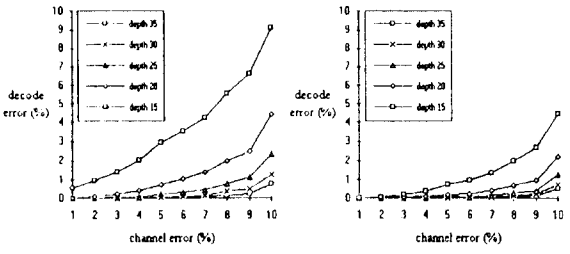
제안된 디코딩 방식의 효율성을 보이기 위하여 Pb와 Pb,new의 차이를 구하면 식 (5)와 같다.

$$P_b - P_{b,new} = 2(P_b)^3 - 3(P_b)^2 + P_b \quad (5)$$

식 (5)에서 $P_b$ 1/2 인 경우는 양수가 되므로 $P_{b,new}$ $P_b$가 된다. (단 등호는 $P_b$ = 1/2일때 성립한다. ) 대부분의 경우는 $P_b$ 1/2 이므로 개선된 디코딩 방법은 에러의 확률을 줄이는데 효과적이며, $P_b$가 작아지면 질수록 $P_{b,new}$사이의 차이가 커짐을 알 수 있다. 이는 채널에서의 에러가 작아 $P_b$가 상당히 작아질때 $P_{b,new}$는 상대적으로 훨씬 적어짐을 의미하며, 결국 제안된 디코딩 방법은 역추적을 위한 초기 상태가 IS에서 연속적으로 선택되지 않는다면 디코딩 에러을 찾아 정정하기 쉬어진다는 것을 의미한다.

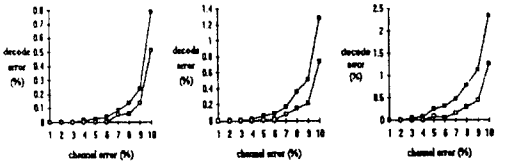
## Ⅵ. 실험 결과 및 분석

잡음이 있는 채널에서 소스 데이타와 디코딩 출력 데이타간의 오류 데이타의 퍼센티지를 나타내는 디코딩에러에 대한 실험 결과를 제시한다. 그림 10은 채널에서 발생하는 채널 에러에 대한 디코딩 에러를 제시하고 있다.

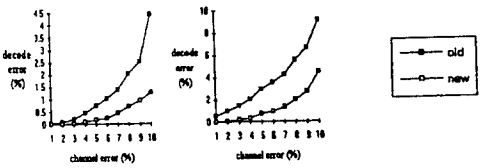



(a) (b)

그림 10. 채널에러 vs 디코딩에러
(a) 기존의 알고리듬 (b) 제시된 알고리듬
Fig. 10. Channel error vs decoding error.
(a) by existing algorithm
(b) by the proposed algorithm.



(a) (b) (c)




(d) (e)

그림 11. Depth별 기존 알고리듬과 제안된 알고리듬의 디코딩 에러 비교 (a) depth = 35 (b) depth = 30 (c) depth = 25 (d) depth = 20 (e) depth = 15
Fig 11. Comparison of decoding errors for a given depth. (a) depth = 35 (b) depth = 30 (c) depth = 25 (d) depth = 20 (e) depth = 15.

그림 10 (a)는 기존의 알고리듬만을 적용한 경우에, 그림 10 (b)는 제안된 알고리듬을 첨가하였을 경우의 디코딩 에러를 나타낸다. 채널 에러가 증가되면 디코

딩 에러도 따라서 증가되는 것을 보여주고 있으며, 제안된 알고리듬의 디코딩 에러는 기존의 알고리듬에 비해 크게 개선됨을 보인다. 예를 들어 디코딩 길이(depth)가 25이고 채널에러가 5%일때 디코딩 에러는 기존의 알고리듬이 0.242%인 반면 개안된 것은 0.088%로 개선된다.

그림 11은 디코딩 길이별로 기존의 알고리듬과 제안된 알고리듬에 의한 디코딩 에러를 비교한 것으로 전반적으로 에러율이 크게 개선됨을 보인다.

그림 11 (a)는 디코딩 길이가 35일때의 디코딩 에러로 채널에서의 에러가 5%일때 기존의 알고리듬은 0.022%이고 제안된 알고리듬은 0.001%의 에러율을 보인다.

그림 12는 제안된 알고리듬의 효율을 나타내는 것으로 기존의 알고리듬에 의하여 발생한 에러 중에서 제안된 알고리듬에 의하여 정정된 에러의 비, 즉 성능 향상을 채널에러에 대하여 표시하였다. 채널에서의 에러가 3%이고 디코딩 길이가 25일때 기존의 알고리듬은 0.046%의 디코딩 에러를 발생하고 제안된 알고리듬은 0.004%의 디코딩 에러를 발생한다. 이는 1,000,000개의 소스 데이타 중에서 기존의 알고리듬은 460개의 에러를 발생하지만 제안된 알고리듬은 40개의 에러를 발생한다는 것이다. 따라서 제안된 알고리듬은 기존의 알고리듬에 의하여 정정되지 못한 460개의 에러중에서 420개는 더 정정하였고 40개를 정정하지 못하였음을 의미한다.

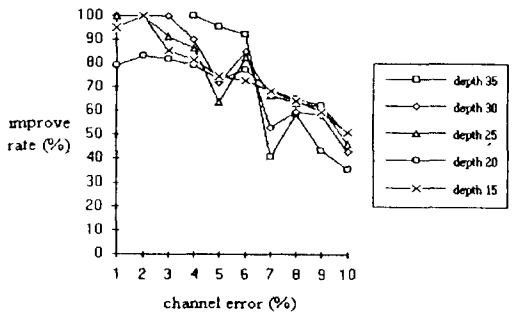



그림 12. 채널에러에 따른 제안된 알고리듬의 성능 향상

Fig. 12. Perfomance improvement of the proposed algorithm vs channel error.

그림 12에서 향상도는 채널에서의 에러가 증가함에 따라서 감소함을 알 수 있는데 이는 제 V 장의 확률 계산에서 예견한 바로, 채널에서의 에러가 증가하게되면 채널에러가 낮을때보다 IS가 커지게되어 역추적의 초기 상태가 연속적으로 선택될 수 있는 확률이 커지기 때문이다. 이런 향상도는 채널에러가 10%이하일때 보통 50%이상을 보였으며, 채널에러가 50%이상 일어날때 0%가 될것으로 예견된다. 그림 12에서 디코딩 길이가 35인 경우 채널에러가 4%일때부터 표시되었으며, 이는 채널에러가 4%보다 작을때는 기존의 것에서 에러가 발생하지 않았기 때문에 향상도를 표기할 수 없었다.

## Ⅶ. 결론 및 추후 과제

에러 정정을 위한 Viterbi 알고리듬은 실시간 처리가 가능하도록 하드웨어를 구현하는 과정에서 역추적의 초기 상태를 하나의 디코딩 사이클내에 path metric이 가장 작은 상태로 선택하기 어려우므로 임의적으로 선택한다. 이 경우 디코딩 길이가 무한히 크다면 생존자는 하나만 남게되어 에러를 정정할 수 있으나 디코딩 길이를 5K로 사용하는 대부분의 경우는 생존자가 두개 이상이 될 수 있다. 따라서 두개 이상 존재하는 생존자에 대한 상황을 분석하여 디코딩 출력을 결정함으로서 효율적인 디지탈 통신시스템을 구현할 수 있다. 본 논문에서는 생존자 상황의 판단을 위하여 디코딩 출력을 결정하는데 있어서 연속적으로 실행되는 역추적된 최종 상태를 비교하여 판단하는 방법을 제안하였다. 제안된 알고리듬은 기존의 알고리듬에 비하여 에러의 정정 효율을 개선하였으며, 이는 같은 디코딩 에러를 얻기 위하여 요구되는 디코딩 길이가 기존의 알고리듬에 비하여 줄어든다는 것을 의미한다. 디코딩 길이를 줄임으로써 코딩율 R이 1/2보다 큰경우의 하드웨어 구현에 사용되는 punctured coding의 구현시 요구되는 큰 디코딩 길이에 의한 문제점을 효과적으로 극복할 수 있으며, 디코딩시 디코딩 초기 딜레이를 줄일 수 있다.

통신 시스템에서 요구하는 실시간 처리를 위하여 디코딩 사이클은 더욱 줄어들고 있으며, 이에 제안된 알고리듬의 효율적인 하드웨어 구현은 중요한 과제로 남는다.

## 參 考 文 獻

[1] J. Anderson, S. Mohan, *Source and Chanel Coding : An Algorithmic Approach*, Kluwer Academic Pub.: Boston, MA, 1991.

[2] S. Lin, *An Introduction to Error-*

*Correction Codes*, Prentice-Hall, Englewood Cliffs, NJ, 1970.

[3] S. Lin, D. Costello, *Error Control Coding : Fundamentals and Application*, Prentice Hall, Englewood Cliffs, NJ, 1983.

[4] G. Clark, J. Cain, *Error-Correction Coding for Digital Communication*, Plenum Press : New-York, 1981.

[5] J. Omura, "On the Viterbi Decoding Algorithm," *IEEE Trans. Inform. Theory*, vol. IT-15, No.1, Jan. 1969, pp.177-179.

[6] G. Forney, "Convolution Codes : Maximum Likelihood Decoding," *Inform. Theory*, vol. 25, July 1974, pp.222-266.

[7] A. Viterbi, "Convolution Codes and their Performance in Communication Systems," *IEEE Trans. Comm.*, vol. 19, No. 10, Oct. 1971, pp.751-772.

[8] T. Truong, M. Shih, I. Reed, E.H. Satorius, "A VLSI Design for a Trace-Back Viterbi Decoder," *IEEE Trans. Comm.*, vol. 40, No.3, March 1992, pp.616-624.

[9] C. Chang, K. Yao, "Systolic Arrary Processing of the Viterbi Algorithm," *IEEE Trans. Inform. Theory*, vol. IT-35, No. 1, Jan. 1989, pp.76-86.

[10] P. Gulak, T. Kailath, "Locally Connected VLSI Architectures for the Viterbi Algorithm," *IEEE J. Sel. Areas Comm.*, vol. 6, No. 3, April 1988, pp. 527-537.

[11] G. Fettweis, H. Meyr, "High-Speed Parallel Viterbi Decoding: Algorithm and VLSI Architecture," *IEEE Comm.*, vol. No. 5, May 1991, pp.46-55.

[12] I. Onyszchuk, "Truncation Length for Viterbi Decoding," *IEEE Trans. Comm.*, vol. 39, No.7, July 1991, pp. 1023-1026.

[13] F. Hemmati, D. Costello, Jr., "Truncation Error Probability in Viterbi Decoding," *IEEE Trans. Comm.*, vol. COM-25, No.5, May 1977, pp.530-532.

[14] R. McEliece, I. Onyszchuk, "Truncation Effects in Viterbi Decoding," in Proc. MILCOM'89, vol. 2, Oct. 1989, pp.29.3.1-29.3.5.

[15] J. Cain, G. Clark, J. Geist, "Punctured Convolutional Codes of Rate (n-1)/n and Simplified Maximum Likelihood Decoding," *IEEE Trans. Inform. Theory*, vol. IT-25, No. 1, Jan. 1979, pp.97-100.

[16] A. Viterbi and J. Omura, *Principles of Digital Communications and Coding*, McGraw-Hill: New York, 1979.

## 著 者 紹 介

**黃 義 俊(正會員)**

1969年 12月 21日生. 1993年 2月 서강대학교 전자공학과 졸업. 1993年 3月 ~ 현재 서강대학교 전자공학과 대학원 석사과정 재학중. 주관심 분야는 VLSI 설계, Computer Architecture 및 System Design, CAD 등임.

•

**林 信 一(正會員)**

1957年 4月 2日生. 1980年 2月 서강대학교 전자공학과 졸업. 1983年 2月 서강대학교 전자공학과 공학 석사 취득. 1990年 8月 ~ 현재 서강대학교 전자공학과 대학원 박사과정 재학중. 1980年 2月 ~ 1981年 2月 (주)한국소프트웨어 근무. 1982年 3月 ~ 1992年 1月 한국전자통신연구소 선임연구원. 1992年 1月 ~ 현재 전자부품종합기술연구소 주문형반도체설계센터 선임연구원. 주관심 분야는 VLSI 설계, Mixed Mode 회로 설계(ADC/DAC등), Testable Design 등임.

**李 種 和(正會員)**

1967年 11月生. 1991年 2月 서강대학교 전자공학과 졸업. 1993年 2月 서강대학교 전자공학과 공학 석사 취득. 1993年 3月 ~ 현재 서강대학교 전자공학과 대학원 박사과정 재학중. 주관심 분야는 VLSI 설계, Computer Architecture 및 System Design, CAD 등임.

•

**黃 善 泳(正會員)**

1954年 5月 20日生. 1976年 2月 서울대학교 공대 전자공학과 졸업. 1978年 2月 한국과학원 전기 및 전자공학과 공학 석사 학위 취득. 1986年 10月 미국 Stanford대학 전기전자공학과 공학 박사 학위 취득. 1976年 3月 ~ 1981年 7月 삼성반도체(주) 연구원. 1986年 7月 ~ 1986年 1月 Stanford대학 CIS연구소 연구원. 1986年 12月 ~ 1988年 2月 Fairchild Semiconductor PARC 기술 자문. 1989年 3月 ~ 현재 서강대학교 전자공학과 교수. 주관심 분야는 CAD, Computer Architecture 및 System Design, VLSI 설계 등임.