고속 비터비 복호기를 위한 새로운 생존경로 메모리 관리 방법

정회원 김 진 율*, 김 범 진**

A New Survivor Path Memory Management Method for High-speed Viterbi Decoders

Jin-Yul Kim*, Beomjin Kim** *Regular Members*

요 약

본 논문에서는 고속의 전송속도를 요구하는 근래의 디지털 통신시스템에서 그 필요성이 크게 증가 하고 있는 고속 비터비 복호기의 설계를 위한 새로운 생존경로 메모리 관리 방법과 하드웨어 구조를 제안한다. 제안된 방법은 k-개의 시작노드번호 결정회로를 독창적 방법으로 제어함으로써 복호를 시작할 수 있는 합병된 생존경로를 즉시 역추적할 수 있으며 기존의 생존경로 관리 방법들에 비하여 더 작은 크기의 생존경로 메모리와 더 짧은 처리지연시간을 갖는다. 또, 제안된 방법에서는 동작 속도가 똑같은 1개의 읽기 포인터와 1개의 쓰기 포인터 만이 필요하므로 기존의 방법들에서 요구되었던 복잡한 k-포트 메모리나 k-배 빠른 읽기 능력을 갖는 메모리를 사용할 필요가 없이 표준적인 이중포트 메모리 구조를 사용하여 생존경로 메모리를 용이하게 구현할 수 있다. 제안된 방법은 즉시 역추적을 위한 추가의 하드웨어를 요구하지만 고속의 처리속도가 필요한 비터비 복호기 구현에 기존 방법들보다 더 우수하다.

ABSTRACT

In this paper, we present a new survivor path memory management method and a dedicated hardware architecture for the design of high-speed Viterbi decoders in modern digital communication systems. In the proposed method, a novel use of k-starting node number deciding circuits enables to acheive the immediate traceback of the merged survivor path from which we can decode output bits, and results in smaller survivor path memory size and processing delay time than the previously known methods. Also, in the proposed method, the survivor path memory can be constructed with ease using a simple standard dual-ported memory since one read-pointer and one write-pointer, that are updated at the same rate, are required for managing the survivor path: the previously known algorithms require either complex k-ported memory structure or k-times faster read capability than write. With a moderate hardware cost for immediate traceback capability the proposed method is superior to the previously known methods for high-speed Viterbi decoding.

Ⅰ. 서 론

비터비 복호기[1][2]는 자기(magnetic) 저장 장치, 음성 및 문자 인식 등 여러 응용분야에서 광범위하게 사용되고 있으며 디지털 통신시스템에서는 통신 채널에서 발생하는 데이터 오류를 정정하여 신뢰성 있는 통신을 하기 위하여 널리 사용되고 있는 길쌈부호(convolutional code)[1]의 복호를 위한 수단으로 채택되고 있다.

길쌈부호화기의 부호 경로는 상태도를 시간축상으로 전개한 트렐리스(trellis) 도를 사용하여 효율적

* 수원대학교 전자공학과 (jykim@mail.suwon.ac.kr), ** (주)MMC 테크놀로지 선임연구원 (kbj@mmctech.com)
논문번호 : 010357-1127, 접수일자 : 2001년 11월 27일

으로 기술될 수 있으며, 비터비 복호기는 트렐리스도에서 가능한 송신 부호의 모든 경로들 중에서 수신된 신호 부호의 경로와 가장 잘 일치하는 경로를 찾아 준다. 디지털 통신시스템에서 채택되고 있는 길쌈부호들은 일반적으로 구속장(constraint length)이 크고 따라서 부호 경로를 나타내기 위한 트렐리스도의 상태의 수가 많다. 반면, 자기 저장 장치나 음성 문자 인식 등 비터비 복호기를 사용하는 타 응용 분야에서는 일반적으로 트렐리스도의 상태의 수가 길쌈부호의 복호를 위해 사용하는 트렐리스도에서 보다 훨씬 적다. 이것은 길쌈부호의 복호를 위해 비터비 복호기를 구현할 때 고속의 처리속도를 얻기가 더 어렵다는 것을 의미한다.

음성신호 전송을 근간으로 하는 저속의 전송속도를 목표로 하는 통신시스템의 경우에는 필요 하드웨어의 양을 감소시키거나 소비 전력을 낮게하는 것이 비터비 복호기 설계의 주 관심 대상이다[3]. 그러나, 근래에 고속의 전송속도를 요구하는 통신시스템의 필요성이 크게 증가하고 있으며 이러한 통신시스템에서는 시스템에서 요구하는 전송속도를 만족할 수 있는 고속의 비터비 복호기를 사용하여야 한다. 예를 들어, 최근 사용이 확산되고 있는 IEEE 802.11a 무선 LAN 모뎀[9]의 경우 54Mbps까지의 전송속도를 처리 할 수 있는 고속 비터비 복호기가 필요하다.

먼저, 디지털 통신 시스템에서 사용하는 길쌈 부호를 복호하기 위한 비터비 복호기의 일반적인 구성도(그림 1)를 이용하여 비터비 복호 과정을 간략하게 설명하고 비터비 복호기의 고속화를 위하여 복호과정의 어떤 단계가 가장 결정적인 역할을 하는지 설명한다. 그림 1에서 보인 바와 같이 비터비 복호기는 크게 분기메트릭 연산 (BMC: branch metric calculation)부, 덧셈 비교 선택 (ACS: add-compare-select)부, 상태메트릭 메모리(SMM: state metric memory), 생존경로 메모리(SPM: survivor path memory), 그리고 역추적(TB: traceback) 제어부로 이루어져 있으며 입력된 수신 데이터에 대하여 다음 각 단계로 설명된 동작을 차례로 수행한다.

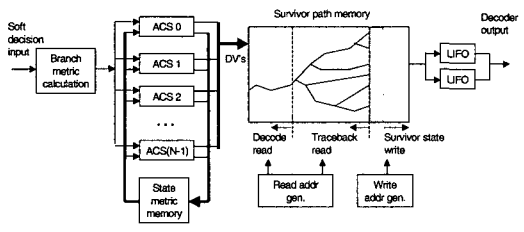


그림 1. 비터비 복호기의 블록 구성도

단계 1) 분기메트릭의 계산: 수신 데이타가 BMC에 인가되면 BMC는 부호기(encoder)에서 발생될 수 있는 각 부호 심볼과 수신데이터의 근사도를 계산하여 분기메트릭을 출력한다.

단계 2) 상태메트릭의 갱신 및 결정벡터 저장: ACS는 상태메트릭 메모리에 저장되어 있는 어떤 이전 상태에 대한 상태메트릭과 그 상태에서 현재 상태로 입력되는 각 분기(branch)에 대한 분기메트릭을 더하여 새로운 상태메트릭을 계산하며 모든 분기에 대해 계산된 상태메트릭들의 크기를 비교하여 상태메트릭의 크기가 가장 작은 분기를 선택하고 선택된 분기에 해당되는 상태메트릭을 현재 상태의 상태메트릭으로 갱신 저장한 후, 각 ACS에서 어느 분기가 선택되었는지를 나타내는 정보인 결정비트(decision bit) D_n을 출력한다. 각 ACS에서 출력된 N개의 결정비트들을 모아 결정벡터(DV: decision vector)라 부르며 결정벡터는 생존경로 메모리에 한번에 컬럼(column) 단위로 저장된다. 여기서 N은 상태의 개수를 나타낸다.

단계 3) 시작상태번호의 선택: 다음에 역추적을 위한 시작상태번호를 선택한다. 각 상태에 대하여 저장되어 있는 상태메트릭 값을 비교하여 최소의 상태메트릭을 갖는 상태번호를 선택하고 이 상태번호를 생존경로 역추적을 위한 시작상태번호로 선택한다. 이 단계에서는, 아래에 설명한 바와 같이 생존경로가 일정 시간 단계 이후에 하나의 생존경로로 합병(merge)되는 성질이 있으므로 최소의 상태메트릭을 갖는 상태번호를 찾는 대신 임의의 상태번호를 선택하여 시작상태번호로 두어도 된다.

단계 4) 생존경로 역추적: 선택된 역추적 시작상태번호로부터 생존경로의 역추적을 시작한다. 바로 이전의 상태번호는 현재 상태번호와 결정비트의 정보로부터 역추적할 수 있다. 예를 들어, $r=1/m$의 전송률을 갖는 길쌈코드의 경우 $S_{n-1}=(S_n \ll 1)|D_n^{S_n}$ 의 관계식으로 이전 상태번호를 역추적 할 수 있다. 즉, 현재 상태번호 값 S_n의 값을 왼쪽으로 1비트 만큼 시프트(shift)한 후 생존경로 메모리에 저장되어 있는 상태 S_n에 대한 결정비트 $D_n^{S_n}$를 읽어 맨 오른쪽에 덧붙히면 역추적하는 한 단계 전의 상태번호 S_{n-1}을 얻을 수 있다. 생존경로들은 구속장의 길이가 K인 길쌈 부호의 경우 일

정 시간 단계 $L \geq 4K \sim 5K$ 가 지나면 하나의 생존경로로 합병되는 성질이 있으므로[1] 위의 역추적 과정을 생존경로가 합병될 때 까지 L 시간 단계 동안 반복한다.

단계 5) 데이터 복호: L 단계 반복되는 생존경로 역추적 과정의 결과로 하나로 합병된 생존경로가 발견되면 이 경로에 속한 상태에 해당되는 결정 비트를 읽어 내어 수신된 데이터 비트를 복호할 수 있다. 복호된 비트들은 LIFO(last-in first-out)로 출력된다. 생존경로 메모리에서 복호되어 출력되는 비트의 순서는 역추적동작에 의하여 시간축에 대하여 역순으로 출력되므로 이중 버퍼된 LIFO를 사용하여 시간에 대하여 올바른 순서가 되도록 다시 역순으로 배열시킨다.

그런데, 위에서 설명한 비터비 복호 과정에 있어 단계 4)의 생존경로 역추적 과정에서는 하나의 합병된 생존경로를 찾기 위하여 L 단계의 역추적 반복 과정이 필요하므로 다른 단계의 동작에 비하여 가장 많은 시간이 소모된다. 따라서, 위에서 설명한 순차적 역추적 방법으로는 고속의 처리속도를 얻기가 불가능하다. 따라서, 복호처리의 고속화를 위하여 효율적인 생존경로 메모리의 구조와 역추적 제어 방법은 매우 중요하며 같은 처리속도를 얻는 경우에도 생존경로 메모리에서 필요한 메모리의 크기와 처리지연시간(processing latency)은 역추적 및 생존경로 메모리 관리 방법에 따라 크게 달라지게 된다. 이에 따라, 고속의 비터비 복호기를 구현하기 위하여 많은 생존경로 관리 방법들이 제안되어 왔다[4]-[8].

본 논문에서 기존에 제안되었던 생존경로 메모리의 구조와 역추적 방법들을 살펴보고 이들이 갖는 문제점을 필요한 메모리의 크기 및 구조와 관리 방법, 처리지연시간 등의 측면에서 먼저 살펴 본 후 고속 비터비 복호기의 설계에 우수한 성능을 보이는 개선된 새로운 생존경로 메모리 역추적 방법 및 그 장치를 제안한다. 또, 제안된 방법과 기존의 방법들을 비교하여 서로의 장단점을 비교 분석한다.

Ⅱ. 기존의 생존경로 관리 및 결정 방법

2.1 기존의 생존경로 결정 방법

기존에 제안된 생존경로 관리 및 복호를 위한 최적 경로 결정 방법으로 레지스터교환(RE: register exchange) 방법, 역추적(TB: traceback) 방법, 추적삭제(TD: trace-delete) 방법들이 있다. RE 방법[4]의 경우 개념적으로 간단하지만 구속장이 큰 경우 레지스터의 내용을 교환하기위한 MUX의 수가 많아지며, 전력소모가 크게 증가하여 VLSI 구현에는 적합하지 않다.

TB 방법[5]-[7]은 전통적으로 비터비 복호기의 설계에 가장 널리 채택되어 온 방법으로 다양한 생존경로 메모리 관리 방법이 제안되어 왔다. TB 방법에서는 생존경로 메모리를 하나의 블록으로 구성하여 순차적으로 하나로 합병된 생존경로를 찾아내는 순차적 TB 방법과, 생존경로 메모리를 복수의 가상적 메모리 뱅크로 나누고 각 메모리 뱅크에서 내용을 쓰거나 읽는 동작을 동시에 수행하여 고속의 처리속도를 얻는 방법이 제안되었다. 순차적 방법은 구현이 간편하지만 처리속도가 느려 복호속도가 느린 응용분야에서만 적용가능한 반면, 복수의 메모리 뱅크를 채용하는 방법은 고속의 처리가 가능하여 최근까지 활발한 관심과 연구가 진행되었다.

최근에 제안된 TD 방법[8]에서는 생존경로 메모리를 일반적인 메모리가 아닌 레지스터들로 구성하고 이 레지스터들의 각 단(stage)사이에 부호화기의 트렐리스도의 형태에 따라 배선 형태가 정해지는 조합논리 회로를 사용하여 각 단을 연결한다. 생존경로 메모리의 각 단에서는 앞단에서 입력되는 생존경로 존재 유무 정보와 그 단의 레지스터에 저장된 중간단계의 정보, 그리고 조합논리회로로 표현된 트렐리스도의 형태를 정보로 사용하여 다음 단에서 생존경로가 존재하게 될지 아닌지 여부를 결정하며 생존경로가 존재하는 경우 이 정보를 다음 단으로 전달하고 그렇지 않은 경우 생존경로를 제거(delete)하도록 한다. ACS에서 출력되는 결정벡터를 생존경로 메모리의 첫 단에 입력하고 위의 과정을 충분히 긴 단계 이상 연속적으로 반복하면 생존하지 못하는 경로는 제거되어 최종적으로 하나의 생존경로만이 남게된다.

TD 방법은 결정벡터 값을 쓰고나서 조합논리회로의 전달 지연시간 후에 바로 복호된 데이터 비트를 결정할 수 있고 또 출력단의 LIFO 버퍼가 필요없다는 장점을 가지고 있다. 그러나, TD는 기성의 표준적인 메모리 구조가 아닌 레지스터와 조합논리회로를 이용한 특별한 구조를 사용하므로 구속장이 큰 경우 설계가 복잡해지고 모든 단을 거치는데 필요한 조합논리회로에서의 전달지연이 매우 길어지는 문제가 있다.

본 논문에서는 간편한 표준적인 메모리 구조를 사용하여 고속의 비터비 복호기를 설계할 수 있도록 TB에 기초하는 새로운 생존경로 메모리 관리

방법을 제안한다. 제안된 방법은 구속장이 큰 고속의 비터비 복호기의 설계에 적합하며 사용된 메모리 뱅크의 크기와 처리지연시간 면에서 기존의 TB 방법에 비하여 우수하다.

2.2 기존의 TB 방법

비터비 복호를 위한 트렐리스도에서 모든 경로는 시간의 경과에 따라 하나의 경로로 합병되므로 아래의 설명에서 모든 경로는 L 시간 단계를 거친 후 하나의 경로로 수렴한다고 가정하기로 한다.

1) Stanford Telecom의 생존경로 메모리 관리법

기존에 제안된 TB 방법으로 먼저 Stanford Telecom사의 STEL-2020 비터비 복호기 칩 설계[5]에 사용된 방법을 들 수 있다 (그림 2). 이 방법에서는 생존경로 메모리를 컬럼의 수가 각각 L 개 (생존경로 메모리의 크기는 별도의 표시가 없는 한 모두 컬럼의 개수를 나타내며 한 컬럼의 길이는 N이다)인 4개의 메모리 뱅크로 구성하였다. 시각 $t=0$부터 시각 $t=(L-1)$까지의 구간에서는 결정벡터가 Bank0에 저장(W: write)되고 있는 동안, 과거에 저장된 결정벡터를 이용하여 Bank1에서는 출력 비트를 복호(D: decode)하고, Bank3에서는 역추적(T: traceback)을 수행한다. 이때 Bank2는 유휴(I: idle) 상태에 있게 되는데, 이것은 W, D, T의 세가지 동작이 서로 시간 동기가 맞아 올바르게 수행되기 위하여 필요한 지연시간을 얻기 위한 것이다.

어떤 메모리 뱅크에 기록된 결정벡터로부터 출력 신호를 복호하려면 생존경로가 하나로 합병된 이후에야 가능하다. 예를 들어, Bank0에 기록된 결정벡터로부터 올바로 복호하기 위해서는 Bank1에 기록된 결정 벡터를 이용하여 먼저 역추적하여 Bank0를 복호할 시작상태번호를 결정하여야 한다. L 시간 단계를 거친 후 모든 경로가 하나의 경로로 합병된다는 가정에 의해 Bank1의 마지막 임의의 상태번호로부터 역추적을 시작하여도 Bank1 내에서 L 단계를 거치므로 Bank1의 시작위치에서는 하나의 경로로 수렴된다. 이렇게 얻어진 상태번호로부터 출발하여 Bank0의 모든 출력 비트는 정확하게 복호되어질 수 있다. 그림 2에 Bank1의 역추적(T)후 Bank0의 복호(D)에 이르는 상호 관계를 점선으로 나타내어 보였다.

이 방법에서 필요한 총 생존경로 메모리의 크기는 $4L$ 컬럼(column) 개이고 TB에 소요되는 처리시간 지연은 $4L$ 단위 시간이다. 이것은 메모리 뱅크 Bank0에 기록된 데이터가 복호 되기 위해서는 Bank0에 저장(W) → Bank1에 저장(W) → Bank1에서 역추적(T)으로 Bank0의 복호 시작상태번호 결정 →Bank0의 데이터를 복호(D)하는데 전부 $4L$의 처리시간지연이 생기기 때문이다. 따라서, Bank0에는 이 처리지연시간 후에서야 비로소 다른 새로운 결정벡터가 기록될 수 있다. 한편, 이 방법에서는 생존경로 저장 메모리가 2개의 읽기 포인터(read pointer)와 하나의 쓰기(write pointer) 포인터를 가지고 있어야 한다.

2) k-포인터 알고리즘

위에서 설명한 문헌[5]의 알고리즘을 일반화시킨 방법으로 k-포인터 짝수 알고리즘(k-pointer even algorithm)[6]이 있다. 문헌[5]의 알고리즘은 k-포인터 짝수 알고리즘에서 k=2인 경우에 해당하는 특별한 경우이다. 이 방법에서는 k 개의 읽기 포인터와 1개의 쓰기 포인터를 사용한다. 그림 3은 k=3인 경우의 예이다. 이 방법에서는 메모리를 $L/(k-1)$의 크기를 갖는 $2k$ 개의 뱅크로 나누고 $(k-1)$개의 읽기 포인터가 역추적(T)을 위하여 사용되는 동안 1개의 읽기 포인터가 복호(D)를 위해 사용되며, 1개의 쓰기 포인터가 저장(W)을 위해 사용된다. 복호

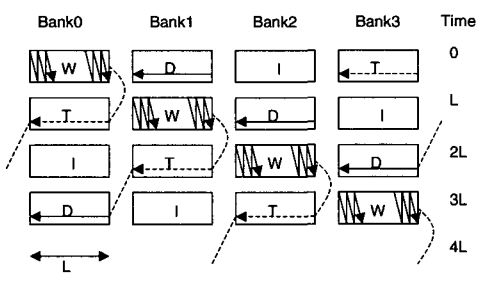


그림 2. Standford Telecom[5]의 생존경로 메모리 관리 방법

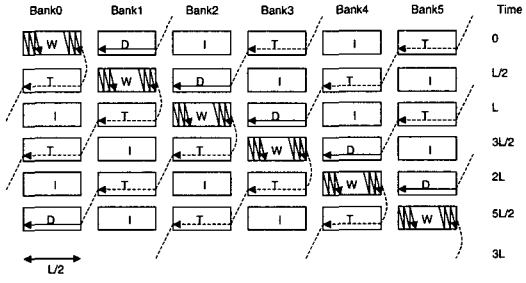


그림 3. 3-포인터 짝수 방법[6]을 사용하는 생존경로 메모리 관리 방법

를 수행하기 전에 적어도 L길이의 역추적(T)을 수행하여야 하므로 이 방법에서는 $(k-1)$개의 메모리 뱅크의 컬럼의 개수가 L보다 커야 한다. 이 방법에서 필요한 총 생존경로 메모리의 크기는 $2kL/(k-1)$이고 처리시간 지연은 $2kL/(k-1)$이다.

한편, 복호(D)와 저장(W)을 하는 포인터가 같은 메모리 뱅크를 가리키도록 k-포인터 짝수 알고리즘을 약간 변형하여 메모리 뱅크의 수를 한 개 줄인 k-포인터 홀수 알고리즘(k-pointer odd algorithm)[6]이 있으나 기본적으로 k-포인터 짝수 알고리즘과 비슷한 성능을 보이는 반면 더 복잡한 제어 구조를 가져 본 논문에서는 별도로 언급하지 않는다. k-포인터 홀수 알고리즘에서 필요한 총 생존경로 메모리의 크기는 $(2k-1)L/(k-1)$ 이고 처리시간 지연은 $2kL/(k-1)$이다.

3) 1-포인터 알고리즘

k-포인터 알고리즘에서 결정벡터를 생존경로 메모리에 한번 저장(W)하는 동안 k 개의 읽기 포인터를 사용하여 역추적(T)과 복호(D) 동작을 수행하여야 한다. 1-포인터 알고리즘[6]은 k-포인터 알고리즘에서 k 개의 읽기 포인터를 사용하는 대신 k배 더 빠른 1개의 읽기 포인터만을 사용하도록 변형한 방법이다. 이 구조에서 연속적인 동작을 보장하기 위하여 하나의 결정벡터를 저장(W)하는 동안 역추적(T) 혹은 복호(D)를 위한 k 번의 읽기 동작 주기를 끝낼 수 있어야 한다.

이 방법에서는 $(k+1)$개의 메모리 뱅크가 필요하며 각 메모리 뱅크의 크기는 $L/(k-1)$이다. 이때 필요한 총 생존경로 메모리의 크기는 $(k+1)L/(k-1)$이고 처리시간 지연은 $(k+1)L/(k-1)$이 된다. k가 2 이상이 되면 1-포인터 알고리즘은 k-포인터 알고리즘에 비하여 적어도 생존경로 메모리의 크기와 처리시간 지연에 있어 훨씬 더 우수해 보인다.

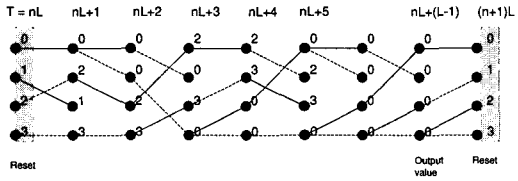
그러나, 생존경로 메모리에 결정벡터를 저장(W)하는 클럭 보다 역추적(T) 및 복호(D)를 위해 k배 더 빠른 읽기 클럭을 사용해야 하는 문제점이 있다. 이동통신용 단말기와 같이 비터비 복호기의 데이터 처리속도가 수 Mb/s 로 낮은 경우[3]에는 빠른 주 클럭으로부터 메모리 쓰기 클럭과 읽기 클럭을 이끌어 내어 k의 값을 높일 수 있으나, 처리속도가 매우 높은 현대의 비터비 복호기를 설계하고자 하는 경우 k의 값을 높이는 것은 생존경로 메모리의 특별한 설계가 요구되므로 매우 어렵다. 따라서, 현실적으로 k의 값은 작은 값이 되며 성능 향상에는 한계가 있다.

4) SNNDC를 사용한 알고리즘

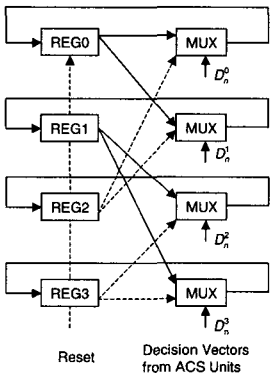
Kamada[7]는 위에서 설명한 Stanford Telecom[5]의 생존경로 메모리 관리 방법을 개선한 방법을 제안하였다. 문헌[5]에서는 이전 메모리 뱅크의 복호(D)를 시작할 상태번호를 결정하기 위하여 현재 메모리 뱅크에 L개의 결정벡터를 모두 저장(W) 한 이후에서야 역추적(T)을 시작할 수 있었다. 반면, 문헌[7]에서는 ACS로부터 출력되는 결정벡터를 생존경로 메모리에 저장(W)함과 동시에 각 ACS가 선택한 경로에 해당되는 이전 상태번호를 SNNDC (Starting Node Number Deciding Circuit)라는 장치를 사용하여 별도의 레지스터에 저장하도록 하였다. 이를 통해 L개의 결정벡터를 모두 저장(W)하고 나면 별도의 역추적(T)과정 없이 복호(D)를 시작할 상태번호를 바로 결정할 수 있다.

이 과정을 그림 4(a)에 보인 트렐리스도의 예를 통하여 자세히 설명한다. 시각 $t=nL$인 순간에 SNNDC내의 각 레지스터는 자기 자신의 상태번호로 초기화(reset) 되었다가 다음 시간 단계에 ACS가 선택한 경로에 해당되는 상태번호로 갱신 된다. 그림에서 각 노드위에 적힌 숫자는 SNNDC 레지스터내의 상태번호 값을 나타낸다. 이 동작은 $t=nL+(L-1)$단계까지 새로운 결정벡터가 저장될 때마다 한번씩 반복적으로 수행된다. 이 예에서는 $t=nL+(L-1)$에서 SNNDC내의 레지스터는 모두 0으로 바뀐 것을 볼 수 있는데 이것은 생존경로들이 모두 L 단계의 시간이 경과하여 하나의 경로로 합병되었고 이 합병된 생존경로가 $t=nL$에서 상태번호 0을 지나가기 때문이다. 따라서, $t=nL+(L-1)$에서는 SNNDC내의 임의의 레지스터를 택하고 그 속에 저장된 상태번호를 추출하여 이 상태번호로부터 시작하여 이전 메모리 뱅크에 기록된 정보를 복호(D)할 수 있다. 그림 4(b)에는 r=1/2, K=3인 길쌈코드를 사용하는 경우 필요한 SNNDC의 구현 예를 보였다. 일반적으로 SNNDC는 상태번호를 저장할 수 있는 N개의 레지스터와 N개의 2:1 MUX로 구현될 수 있으며, 부호화기의 트렐리스도와 같은 배선 형태를 갖는다.

Kamada가 문헌[7]에서 제안한 생존경로 메모리를 관리하는 방법과 SNNDC의 제어 방법을 그림 5(a)

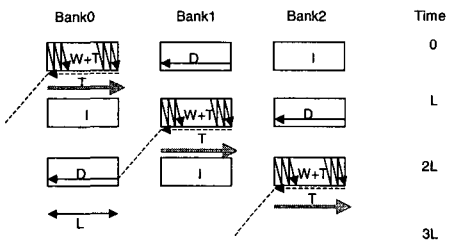


(a)

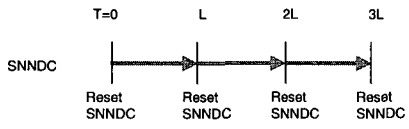


(b)

그림 4. (a) 시작노드번호의 결정방법의 예, (b) 시작노드번호 결정 장치(SNNDC)의 구현 예 (r=1/2, K=3인 경우).



(a)



(b)

그림 5. (a) Kamada[7]의 SNNDC를 사용하는 생존경로 메모리 관리 방법과 (b) SNNDC의 제어 방법.

와 (b)에 각각 설명하였다. 그림 5(a)에서 'W+T' 표시는 생존경로 메모리에 결정벡터를 저장(W)함과 동시에 SNNDC 내의 이전 상태번호 레지스터를 갱신하면서 역추적(T)을 동시에 행한다는 것을 표시한다. Bank0에 저장된 결정벡터를 복호(D)를 하기위해서는 Bank1에 L개 결정벡터의 저장을 끝냄과 동시에 SNNDC에 저장되어 있는 임의 레지스터의 값을 꺼내어 Bank0의 복호(D)를 시작할 상태번호로 사용하면 된다. SNNDC는 그림 5(b)에 보인 것 처럼 매 L 주기마다 초기화된다.

각 메모리 뱅크의 크기는 L이며 3개의 메모리 뱅크가 사용되었으므로 필요한 총 생존경로 메모리의 크기는 $3L$이고 생존경로 메모리에서 최종 복호 비트의 결정에 소요되는 처리시간 지연은 $3L$로 문헌[5]의 방법에 비하여 필요한 메모리의 양과 처리시간이 L만큼 개선되었다.

Ⅲ. 제안하는 생존경로 관리 기법

위에서 설명한 기존의 각 방법들은 다음과 같은 문제점이 있다. 즉, k-포인터 알고리즘의 경우 k개의 읽기 포인터와 1개의 쓰기 포인터가 필요한데 성능 향상을 위하여 k의 값으로 큰 값을 사용하여야 하나 이 경우 메모리 접근 구조가 지나치게 복잡해지므로 k의 값을 임의로 크게 할 수가 없다. 1-포인터 알고리즘의 경우에는 입력 데이터를 저장하는 클럭 보다 k배 더 빠른 읽기 클럭을 사용하여야 하고 k의 값이 클수록 성능이 개선되므로 큰 값의 k가 바람직하다. 그러나, 고속의 비터비 복호기 설계에 있어 읽는 클럭과 쓰는 클럭의 동작 주파수 비를 임의로 높이는 것은 불가능하므로 역시 성능 향상에는 한계가 있다. 문헌[7]에서는 SNNDC 구조를 사용하여 문헌[5]에서 제시된 구조를 개선할 수 있음을 보였다. 그러나, 문헌[7]에서 제시된 방법은 여전히 많은 양의 생존경로 메모리 크기와 처리지연시간을 가지며 본 논문에서 제안하는 방법으로 더 개선되어 질 수 있다.

본 논문에서는 k-개의 읽기 포인터와 1개의 쓰기 포인터 대신, 동작 속도가 똑같은 1개의 읽기 포인터와 1개의 쓰기 포인터만을 사용하면서 위에서 언급된 기존의 방법들 보다 구조가 간단하면서도 더 작은 생존경로 메모리 크기와 짧은 처리시간 지연을 갖는 새로운 생존경로 관리 방법을 제안한다. 제안된 방법은 문헌[7]에서 제시된 방법을 일반화시킨 방법으로 생각할 수 있다.

3.1 제안하는 즉시 역추적(ITB: Immediate Traceback) 방법

1) 2개의 SNNDC를 사용하는 즉시 역추적

본 논문에서 제안하는 즉시 역추적 방법을 그림

6(a)(b)에 예를 들어 보였다. 그림 6(a)에는 제안된 생존경로 메모리의 관리 방법을 보이고, 그림 6(b)에는 제안된 방법에서 부가적으로 사용하는 SNNDC의 제어 방법을 나타내었다.

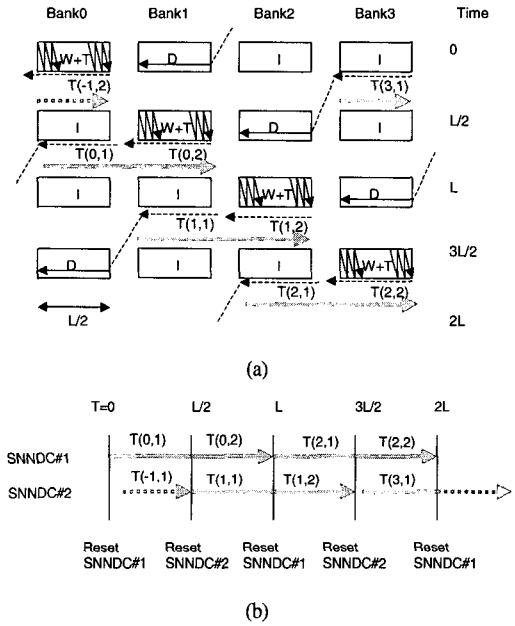


그림 6. (a) 제안된 생존경로 메모리 관리 방법: 2개의 SNNDC를 사용하는 경우, (b) 해당되는 SNNDC의 제어 방법

이 예에는 2개의 SNNDC가 부가적으로 사용되며 본 논문에서 제안하는 독창적인 방법으로 제어되고 있다. 앞에서와 마찬가지로 생존경로가 L 단계의 시간이 경과한 후 하나의 경로로 합병된다는 가정을 한다. 각 메모리 뱅크는 $L/2$개이므로 L 단계의 역추적을 하기위해서 적어도 2개의 메모리 뱅크에서 역추적을 수행하여야 한다. 자세한 제어 방법을 다음에 설명하였다.

(i) 시간 $t=0$에서 SNNDC#1을 초기화 시킨 후, 시간 $t=(L/2)-1$까지 ACS에서 출력되는 결정벡터를 Bank0에 저장하면서 동시에 SNNDC#1 레지스터의 갱신을 반복한다('W+T'로 표시).

(ii) 시간 $t=L/2$에서 SNNDC#2를 초기화 시킨 후, Bank1에 ACS로부터 출력되는 결정벡터를 저장하고 이 결정벡터를 사용하여 SNNDC#1과 SNNDC#2의 레지스터를 동시에 갱신한다. 이때 $t=L/2$에서 SNNDC#1을 재 초기화하지 않고 $t=L-1$까지 SNNDC#1의 갱신을 계속한다. 이것은 SNNDC#1을 사용하여 $t=0$부터 $L-1$까지 L 단계의 역추적 결과 (그림 6(a)에서 T(0,1), T(0,2)로 표시)를 즉시(immediately) 얻기 위한 것이다. SNNDC#1과 SNNDC#2로 제공되는 결정벡터의 값은 동일하지만 각 SNNDC#1과 SNNDC#2는 초기화 되는 시점이 서로 다르므로 각 SNNDC의 레지스터는 다른 상태번호로 갱신된다.

(iii) SNNDC#2를 사용하여 $t=L/2$에서 $t=(3L/2)-1$ 까지 L 단계의 역추적 결과 (그림 6(a)에서 T(1,1), T(1,2)로 표시)를 즉시 얻을 수 있다. 이때 $t=L$에서 SNNDC#2을 재 초기화하지 않고 $t=(3L/2)-1$ 까지 SNNDC#2의 갱신을 계속한다.

(iv) 시간 $t=L-1$에서 각 SNNDC의 갱신이 끝난 후 SNNDC#1에 속한 임의의 레지스터를 택하고 이 값을 시작상태번호로 사용하여 Bank3의 복호(D)를 시작할 수 있다. (생존경로 메모리를 원형(circular)형태로 접근하므로 Bank3에는 Bank0에 저장된 결정벡터보다 먼저 기록된 결정벡터가 저장되어 있다.) 그리고 SNNDC#1은 $t=L$에서 재 초기화 되어 위에서 설명한 동일한 방법에 따라 T(2,1), T(2,2)의 즉시 역추적을 수행한다.

(v) 시간 $t=(3L/2)-1$에서 각 SNNDC의 갱신이 끝난 후 SNNDC#2에 속한 임의의 레지스터를 택하고 이 값을 시작상태번호로 사용하여 Bank0의 복호을 시작할 수 있다. 그리고 SNNDC#2는 $t=(3L/2)$에서 재 초기화 되어 위에서 설명한 동일한 방법에따라 T(3,1), T(3,2)의 즉시 역추적을 수행한다.

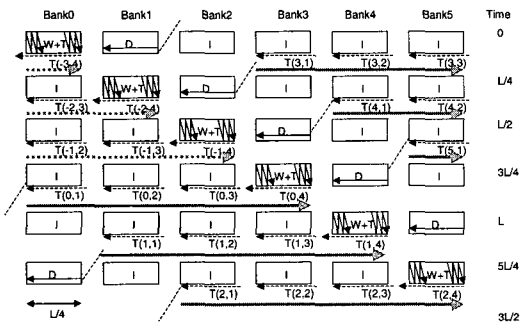
우리는, 단계 (iv)와 (v)에서, 기존 방법에서 필요했던 추가적인 역추적 과정 없이 결정벡터의 저장이 끝나는 순간에 바로 SNNDC의 임의 레지스터 값을 꺼내는 것만으로 복호를 위한 시작상태번호를 즉시 역추적(immediate traceback) 할수 있음을 알 수 있다.

제안된 방법에서는 $L/2$의 크기를 갖는 4개의 메모리 뱅크를 사용하였으므로 필요한 총 생존경로 메모리의 크기는 $2L$이고 처리시간 지연은 $2L$이 된다. 이때 2개의 SNNDC 회로가 부가적으로 필요하다. 제안된 방법에서는 기존 방법[5][6]과는 달리 생존경로 메모리에 접근하기 위하여 서로 속도가 같은 1개의 읽기 포인터와 1개의 쓰기 메모리 만이 필요함을 알수 있다.

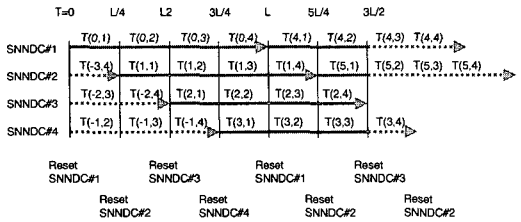
2) k개의 SNNDC를 사용하는 즉시 역추적

위에서 편의상 2개의 SNNDC 회로를 사용하는

경우를 예를 들어 설명하였으나, 제안된 방법은 일반적인 경우로 확장할 수 있다. 4개의 SNNDC 회로를 사용하는 경우에 제안된 방법을 적용하는 예를 그림 7(a)(b)에 설명하였다. 그림 7(a)에서는 크기가 $L/4$인 6개의 메모리 뱅크가 사용되었으며 4개의 독립된 SNNDC 회로가 필요하다. 그림 7(b)에서 굵게 표시된 화살표를 이용하여 각 SNNDC가 초기화되어 갱신이 지속되는 구간을 표시하였다. 그림에서 각 SNNDC는 굵은 화살표가 시작되는 시점에서 자기 상태번호로 초기화되며 위에서 설명한 방법에 따라 SNNDC가 매 단계마다 갱신되다가 화살표가 끝나는 시점에서 갱신이 중지된다. 이때 SNNDC 내의 임의 레지스터 값을 꺼내어 이전 단계의 복호(D)를 시작하는 상태번호로 사용한다. $L/4$의 크기를 갖는 6개의 메모리 뱅크를 사용하였으므로 필요한 총 생존경로 메모리의 크기는 $1.5L$이고 처리시간 지연은 $1.5L$이 된다. 이 경우에도 제안된 방법을 사용하면 서로 속도가 같은 1개의 읽기 포인터와 1개의 쓰기 메모리만이 필요함을 알 수 있다. 제안된 방법은 다음과 같이 일반화 될 수 있다. 즉, k를 사용하는 SNNDC 회로의 개수라 할 때 L/k 크기의 메모리 뱅크 $(k+2)$개를 사용하



(a)



(b)

그림 7. (a) 제안된 생존경로 메모리 관리 방법: 4개의 SNNDC를 사용하는 경우, (b) 해당되는 SNNDC의 제어 방법

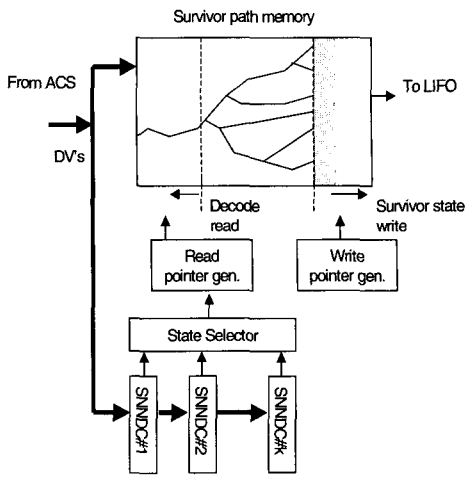


그림 8. k개의 SNNDC를 사용하는 제안된 생존경로 메모리 관리를 위한 하드웨어 구성도

면, 총 생존경로 메모리 크기는 $(k+2)L/k$ 가 되며, 이때 처리지연시간은 $(k+2)L/k$ 단위시간이 된다.

그림 8은, k개의 SNNDC 회로를 사용하는, 제안된 생존메모리 관리 방법을 수행할 수 있는 하드웨어의 구성도이다. ACS 부에서 출력되는 결정벡터는 생존경로 메모리에 쓰기 포인터 발생기가 지시하는 장소에 저장된다. k개의 SNNDC는 서로 독립적으로 수행되며 매 시간단계 마다 각 SNNDC 내부의 레지스터에 현재 ACS가 선택한 경로에 해당하는 상태번호를 갱신하여 저장한다. 이때 i-번째 $(i=1,2,\ldots,k)$ SNNDC는 $t_s(i)= mL+(i-1)L/k$ 시각 $(m=0,1,2,\ldots)$에 초기화되어 $t_e(i)=mL+(i-1)L/k+(L-1)$ 시각까지 SNNDC의 갱신 과정을 반복하며 $t_e(i)$ 시각에 SNNDC 내부의 임의 레지스터를 선택하여 저장되어 있는 상태번호를 꺼내어 시작상태번호로서 출력한다. 상태선택기는 시각 $t_e(i)$에서 i-번째 SNNDC에서 출력되는 시작상태번호를 읽기 포인터 발생기에게 전달한다. 복호과정에서는 읽기 포인터 발생기가 지시하는 장소로부터 생존경로 메모리를 읽어 복호된 비트를 출력하고 이 복호비트들은 LIFO로 전달하여 역순으로 재배열한다.

3.2 성능 비교 분석

제안된 구조와 기존의 구조의 성능차이를 표 1에 정리하여 비교하고 또 그림 9에 생존경로 메모리의 크기와 처리지연시간을 정규화된 파라미터 k에 대

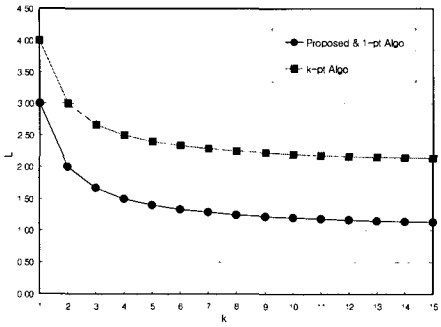


그림 9. 생존경로 메모리의 크기와 처리지연시간의 추이의 비교

하여 그래프로 나타내어 보였다. 가로축의 정규화된 k값에서 각 알고리즘에서 사용하는 설계 기준 파라미터들을 각각 $k_e=k+1$, $k_o=k+1$, $k_p=k$ 의 변환관계를 사용하여 얻을 수 있다.

표 1과 그림 9의 그래프에 보인 근사곡선을 비교해 보면 제안된 알고리즘이 k-포인터 알고리즘에 비하여 생존경로 메모리의 크기와 처리지연시간 면에서 모두 성능이 뛰어남을 알수 있다. 더불어, 성능을 높이기 위하여 k_e값을 키우는 경우, k-포인터 알고리즘은 복잡한 k_e개의 읽기 포인터와 1개의 쓰기 포인터를 갖는 메모리를 사용하여야 하므로 구현이 매우 복잡해지지만 제시된 방법은 1개의 쓰기 포인터와 읽기 포인터만을 가지므로 표준적인 구조의 이중포트 메모리를 사용하여 용이하게 구현할 수 있다. 그림 9를 보면 제안된 방법에서 2개의 SNNDC를 사용하면 어떤 k-포인터 알고리즘보다도 항상 성능면에서 우수함을 알수 있다.

표 1. 생존경로 메모리 관리 방법들의 비교

	k-pointer algorithm	One pointer algorithm	Proposed ITB algorithm
Definition of parameter k	k_e ($\geq$ 2): no. of read pointers	k_o ($\geq$ 2): no. of traceback recursions per write clock cycle	k_p ($\geq$ 1): no. of SNNDCs
No. of read pointers	k_e	1	1
No. of write pointers	1	1	1
No. of banks × Bank size	$2k_e \times \frac{L}{(k_e-1)}$	$(k_o+1)\times \frac{L}{(k_o-1)}$	$(k_p+2)\times \frac{L}{k_p}$
SPM size (columns)	$\frac{2k_e}{k_e-1}L$	$\frac{k_o+1}{k_o-1}L$	$\frac{k_p+2}{k_p}L$
Latency	$\frac{2k_e}{k_e-1}L$	$\frac{k_o+1}{k_o-1}L$	$\frac{k_p+2}{k_p}L$
Memory structure	(k_e+1)-ported memory	dual ported memory with k_o-times faster read capability	standard dual-ported memory
Limiting factor	mutiple access capability	no. of traceback recursions per write clock cycle	the complexity of k_p SNNDCs

한편, 제안된 방법과 1-포인터 알고리즘의 경우 생존경로 메모리의 크기와 처리시간 지연이 같은 근사곡선을 가짐을 볼 수 있다. 생존경로 메모리의 크기와 처리지연시간은 이론적 극한값이 L이며 각 방법 모두 이 극한값에 점근적으로 접근하고 있다. 그러나, 고속 비터비 복호기를 설계하기 위해서는 제안된 방법이 훨씬 더 유리하다. 왜냐하면, 1-포인터 알고리즘에서는 결정벡터를 저장하는 쓰기 클럭 보다 역추적 및 복호를 위해 k_o배 더 빠른 읽기 클럭을 사용해야 하고 저속 비터비 복호기를 설계하는 경우 k_o의 값을 충분히 크게 할 수 있으므로 1-포인터알고리즘을 사용하여 성능을 크게 높일 수 있지만 근래의 비터비 복호기에서 요구되는 높은 데이터 처리속도를 얻고자 할 때 읽기 클럭의 속도를 쓰기 클럭 속도에 비해 임의 배수 k_o로 높이는 것은 회로의 동작 주파수가 높은 관계로 매우 어렵기 때문이다. 1-포인터 알고리즘의 경우 동작 주파수가 높을 경우 $k_o=2\sim3$(이때, 정규화된 파라미터 $k=1\sim2$이다) 정도의 값 이외에는 실용적으로 사용할 수 없으므로 성능 향상에는 한계가 있다. 반면, 제안된 알고리즘에서는 k개의 SNNDC를 구현하는데 필요한 하드웨어의 증가를 허용한다면 요구되는 처리시간지연 조건을 용이하게 만족시킬 수 있다.

다만, 제안된 방법에서는 처리지연시간 개선을 위하여 복수의 SNNDC를 사용하려면 SNNDC 구현에 추가되는 하드웨어가 증가하게 되므로, 생존경로 메모리의 크기가 줄어들어 얻을 수 있는 이득과 처리지연시간을 줄여 얻을 수 있는 이득을 상쇄시킬 수 있으므로 요구되는 처리지연시간을 만족하는 최소 개수의 SNNDC를 사용하도록 하여야 한다. 일반적으로 처리지연시간을 결정하는 파라미터 L의 값은 적어도 구속장 K의 4~5배 이상이며 페이딩이 있는 채널의 경우나 코드 펑처링(puncturing)을 사용하는 경우 이 기준값보다 훨씬 더 큰 값을 사용하여야 한다[5]. 고속의 비터비 복호기 설계에 있어, 기존의 방법들은 L의 값이 큰 경우 요구되는 처리시간지연 요구사항을 만족 시키기가 매우 어려운 반면, 제안된 방법에서는 SNNDC를 구현하는데 필

요한 하드웨어 비용을 지불하는 대신 요구되는 처리지연시간을 용이하게 만족시킬 수 있다. 예를 들어, $k=4$인 경우 제안된 구조에서 4개의 SNNDC가 필요한데 이 경우 처리시간 지연이 $1.5L$로 기존 방법들 보다 크게 줄어들게 되므로 4개의 SNNDC를 구현하는데 필요한 하드웨어의 증가를 충분히 고려할 수 있다.

Ⅳ. 결 론

본 논문에서는 고속 비터비 복호기 설계에 적합한 새로운 생존 메모리 관리 및 역추적 방법을 제안하였다. 제안된 방법은 Kamada[7]가 제시한 방법을 일반화 시킨 것이다. Kamada가 설명한 방법은 본 논문에서 제안된 방법에서 $k=1$인 특별한 경우가 된다. 본 논문에서는 $k\geq 2$인 경우로 Kamada의 결과를 일반적인 경우로 확장하였으며 이때 요구되는 SNNDC의 제어 방법을 새롭게 제시하고 이에 따른 생존경로 메모리 관리를 위한 하드웨어 구조를 제시하였다. 제안된 방법을 사용하여 얻을 수 있는 장단점은 다음과 같다.

먼저, 제안된 방법을 사용하면 표준적인 이중포트를 갖는 하나의 통합된 메모리를 사용하여 용이하게 생존경로 메모리를 구현할 수 있다. 이것은 기존 TB 방법들에서 생존경로 메모리 구현시 k-개의 읽기 포인터와 1개의 쓰기 포인터를 사용하거나, 1개의 쓰기 포인터와 k-배 더 빠른 1개의 읽기 포인터를 사용하는 반면 제안된 방법에서는 동작 속도가 똑같은 1개의 읽기 포인터와 1개의 쓰기 포인터만이 필요하기 때문이다. 기존의 방법에서는 k-개의 읽기 포트를 확보하기 위하여 생존경로 메모리의 각 메모리 뱅크를 분리된 메모리로 구현해야 하거나, 읽기/쓰기 속도제어를 위한 설계가 추가로 필요하다.

다음으로, 제안된 방법은 기존에 알려진 TB 방법들에 비하여 더 작은 생존경로 메모리의 크기를 가지며 또 더 짧은 처리지연시간을 가진다. 그러나, 제안된 방법에서 짧은 처리지연시간을 얻기 위해서는 설계 파라미터 k의 값을 증가 시켜야 하며, k 값의 증가에 따라 SNNDC 회로구현을 위한 추가의 하드웨어가 요구되므로 작은 크기의 생존경로 메모리를 사용하는 이득을 상쇄시킬 수 있다. 따라서, 비터비 복호기에서 요구되는 처리지연시간을 만족하는 가장 작은 k값을 선택하도록 하여야 한다.

만약, 고속의 처리속도와 짧은 처리 지연시간이 동시에 필요한 경우 본 논문에서 제안된 방법이 처리지연시간을 줄이는 데 가장 효과적으로 적용될 수 있다. 그림 9에서 보였듯이 1-포인터 알고리즘만이 제안된 방법과 똑같은 성능 추이를 보이나 처리속도가 높은 경우 적용하기 어렵기 때문이다.

참 고 문 헌

[1] Bernard Sklar, Digital communications: Fundamentals and applications, Prentice Hall, 1988.

[2] H.-L. Lou, "Implementing the Viterbi algorithm: Fundamentals and real-time issues for processor designers," IEEE signal processing mag., vol. 12, pp. 42-52. Sept. 1995.

[3] Y.-N. Chang, H. Suzuki, and K. K. Parhi, "A 2-Mb/s 256-state 10-mW rate-1/3 Viterbi decoder," IEEE J. Solid-State Circuits, vol. 35, no. 6, Jun. 2000.

[4] C. M. Rader, "Memory management in a Viterbi decoder," IEEE Trans. Commun, vol. 29, no. 9, Sep. 1981.

[5] H. A. Bustamante et al., "Standford Telecom VLSI design of a convolutional decoder," Proc. IEEE Conference on Military Communications, vol. 1, pp. 171-178, Boston, Massachusetts, Oct. 1989.

[6] G. Feygin and P. G. Gulak,"Architectural tradeoffs for survivor sequence memory management in Viterbi decoders," IEEE Trans. Commun, vol. 41, no. 1, pp. 425-429, Mar. 1993.

[7] T. Kamada, Viterbi decoder and Viterbi decoding method, US Patent number US6041433A, Mar. 2000.

[8] S.-J. Jung, M.-W. Lee, and H.-J. Choi, "A new survivor memory management method in Viterbi decoders," Proc. IEEE GLOBECOM, pp. 126-130, 1996.

[9] 정의석, 조용수, "IEEE 802.11a 고속 무선 LAN 모뎀 기술," 한국통신학회지, vol. 16, no.10, pp.42-63, 1999.

김 진 율(Jin-Yul Kim) 정회원

1986년 2월 : 서울대학교 전자공학과 학사
1988년 2월 : 한국과학기술원 전기 및 전자공학과 공학석사
1993년 2월 : 한국과학기술원 전기 및 전자공학과 공학박사
1994년 3월~현재 : 수원대학교 전자공학과 조교수
<주관심 분야> 디지털신호처리, 신호처리를 위한 VLSI 설계, 영상 추적

김 범 진(Beomjin Kim) 정회원

1993년 2월 : 서울대학교 전자공학과 학사
1995년 2월 : 서울대학교 전자공학과 공학석사
1995년 2월 ~ 2000년 1월 : LG 전자기술원
2000년 1월 ~ 현재 : (주)MMC 테크놀로지 선임연구원
<주관심 분야> 무선모뎀, 신호처리, 무선랜 시스템