論文98-35C-7-4

고속 Viterbi 복호기를 위한 메모리 관리

(Memory Management in High-Speed Viterbi Decoders)

任 敏 中 *

(Min-Joong Rim)

요 약

메모리를 어떻게 관리하는가 하는 것은 Viterbi 복호기를 실리콘 위에 구현하는데 가장 중요한 문제들 중 하나다. 이 논문은 고속 Viterbi 복호기의 메모리를 관리하는 새로운 traceback scheme을 제안한다. 제안된 방법은 traceback process에 dummy write를 추가하여 읽고 쓰기 동작의 균형을 맞춤으로서 메모리 사용을 보다 간단히 하게 한다. 제안된 방법은 최소의 메모리를 사용하면서도 메모리 내용을 읽고 쓰기 위한 address generation scheme이 간단하고 global interconnection이 존재하지 않아 VLSI 구현에 적합하다.

Abstract

Memory management is one of the most important problems in implementing Viterbi decoders. This paper introduces a novel traceback scheme for memory management of high-speed Viterbi decoders. The new method balances the read and the write operations by inserting dummy write operations into the traceback process, resulting in simpler memory access schemes. It is suitable for VLSI implementation since it uses minimal memory requirements, it does not need global interconnections, and its address generation scheme for accessing memory contents is very simple.

I. 서 론

최근의 영상 신호 압축 기술의 발달은 좁은 대역에 다량의 영상신호를 전송하는 것을 가능하게 하였다. 압축된 디지털 신호는 매우 적은 bit error rate를 필요로 하므로 송신측에서 신호에 redundancy를 넣고 수신측에서 이를 이용하여 오류를 정정하는 forward error correction (FEC) 기술이 중요하게 되었다. Viterbi 복호기는 디지털 TV 등의 통신 시스템에서 쓰이는 FEC 복호기의 핵심부분중의 하나이다. 이 논문은 direct broadcasting satellite receiver[1] 등의 고속으로 동작하는 시스템의 Viterbi 복호기에 적용될 수 있는 아키텍쳐에 관한 것이다. 그러한 Viterbi 복호기는 60Mbps 정도 이상의 고속을 요하고, constraint length K는 7정도이며 traceback depth L도 96이상으로 크다는 특징을 가지고 있다.

Viterbi 복호기의 주요부분들은 branch-metric (BM) unit, add-compare-select (ACS) unit, traceback unit 등이 있다[2] [3]. BM unit은 각 정해진 심벌이 수신되었을 상대적인 확률을 나타내는 숫자인 branch metric을 생성한다. ACS unit은 branch metric을 받아들여 각 data sequence의 상대 확률을 가리키는 path metric을 계산한다. ACS unit에서 각 심벌이 수신될 때마다 생성되는 2^{K-1} bit의 정보는 traceback unit의 메모리에 저장된다. 최적의 결과를 얻기 위해서는 모든 데이터를 다 받은 후 traceback이라고 불리는 decoding process를 시작해야 한다. 데이터가 packet 단위로 나뉘어져 전송되는

* 正會員, 三星電子 半導體 總括 System LSI 본부
(Semiconductor System LSI Business, Samsung Electronics, Co. LTD.)

接受日字:1998年2月9日, 수정완료일:1998年5月27日

packet mode에서는 packet 단위로 traceback을 하면 되지만 데이터가 연속적으로 들어오는 continuous mode인 경우에는 모든 데이터를 다 받은 후 복호하는 것이 불가능하다. 다행히도 모든 데이터를 다 받은 후 traceback을 할 필요는 없으며 일정 숫자 (traceback depth L) 이상의 데이터를 받은 후 traceback을 하면 결과에 거의 차이가 없다는 것이 알려져 있다[3].

Viterbi 복호기에서 traceback unit의 메모리는 무시할 수 없는 영역을 차지하며 따라서 메모리 관리를 어떻게 하느냐 하는 것이 중요하다. 이 논문은 고속 Viterbi 복호기의 메모리 관리에 관한 것이다. 2장에서는 traceback 과정을 수행하는 알고리즘을 설명하며 3장에서는 traceback 알고리즘을 구현하는 여러 가지 방법들을 비교 설명한다.

Ⅱ. Traceback 알고리즘

기존의 traceback 구조와 새로운 traceback 구조를 유도하고 비교 설명하기 위해 traceback process에서 사용되는 3개의 단순한 operation을 정의한다. 여기서 X_n은 시간 n에서 ACS unit으로부터 온 2^{K-1} bit의 정보이다.

(1) W_n은 X_n을 메모리에 저장하는 write operation이다.

(2) R_n은 X_n을 메모리로부터 읽는 read operation이다.

(3) T_n은 읽어낸 X_n과 현재의 state address로부터 다음의 state address를 계산하는 operation이다.

Traceback process는 다음과 같은 3개의 stage를 가지고 있다[3] [4]. Write stage는 메모리에 path정보들을 저장하는 단계로서 이 때 사용되는 단위 operation은 W_n이다. Traceback stage는 메모리로부터 데이터를 읽은 다음 이것과 현재의 state address를 이용하여 다음의 state address를 만들어 내는 단계로서 이 때 사용되는 단위 operation은 R_n과 T_n이다. Decode stage는 복호된 출력결과를 내보낸다는 것을 제외하면 traceback stage와 같은 동작을 한다. Traceback 알고리즘은 다음과 같이 기술할 수 있다.

for n = 1 to N step M

begin

W_{n+2M} -- *M write*

W_{n+2M+1}

:

W_{n+3M-1}

R_{n+3M-1} T_{n+3M-1} -- *L traceback*

R_{n+3M-2} T_{n+3M-2}

: :

R_{n+M} T_{n+M}

R_{n+M-1} T_{n+M-1} -- *M decode*

R_{n+M-2} T_{n+M-2}

: :

R_n T_n

end

여기서 M은 한 번의 L traceback을 한 후 몇 개의 복호된 출력 결과를 내보내는가를 나타낸다. 일반적인 저속 Viterbi decoder의 경우에는 continuous mode로 동작할 때 매 데이터를 받은 후 traceback을 하여 하나의 데이터를 복호해내는 방법을 사용할 수 있다. 이 때에는 M = 1이 된다. 그러나 수십 Mbps를 만족시키는 고속 Viterbi decoder에서는 한 심벌 싸이클 동안 L개의 메모리 operation을 하는 것은 여러 개의 메모리를 사용하여 병렬성을 증가시키더라도 매우 어려운 일이다. 따라서 고속 Viterbi decoder에서는 packet mode에서 동작하는 것과 유사하게 한 번의 traceback을 수행할 때 여러 개의 데이터를 복호하는 방법을 사용한다. M을 증가시키면 하나의 데이터를 얻기 위한 메모리 operation이 감소하지만 복호 지연시간과 메모리 요구량이 증가하게 된다. M을 감소시키면 메모리 operation이 증가하여 여러 개의 메모리 operation이 병렬적으로 수행되어야 하므로 다수의 메모리를 사용하게 되어 전체 메모리 요구량의 감소에도 불구하고 메모리 면적의 증가를 초래할 수 있다. 따라서 적절한 M의 선택이 필요하다. 이 논문에서는 설명을 용이하게 하기 위해 M과 함께 m을 사용하는데 m은 L/M으로서 L개의 데이터를 복호하기 위해 몇 번의 L traceback이 필요한가를 나타낸다. m의 증가는 불필요하게 메모리의 수를 증가시킬 수 있으므로 실제에서 m은 매우 작은 수를 사용하게 될 것이다. 이 논문의 예에서는 m을 2로 즉 M = L/2로 하겠다. 유사한 논의가 임의의 m에 대해서 성립할 수 있다.

주어진 알고리즘을 VLSI로 구현하기 위해서는 다

음과 같은 3개의 task가 필요하다. Scheduling은 operation들이 수행될 control time-step을 지정하는 것이고, resource allocation은 필요한 module의 수를 정하는 것이며 resource binding은 operation을 module에 mapping하는 것이다.

고속 Viterbi 복호기에서 output을 매 싸이클마다 내보내기 위해서는 scheduling에서 병렬성이 고려되어야 한다. Loop의 성능을 향상시키기 위한 잘 알려진 technique으로 software pipelining이 있다[5]. Software pipelining이 적용된 후 schedule이 된 알고리즘은 다음과 같다.

for n = 1 to N+3M step M
begin

W_{n+2M} R_{n+2M-1} T_{n+2M-1} R_{n-1} T_{n-1} R_{n-2M-1} T_{n-2M-1}
W_{n+2M+1} R_{n+2M-2} T_{n+2M-2} R_{n-2} T_{n-2} R_{n-2M-2} T_{n-2M-2}
: : : : : : :
W_{n+3M-1} R_{n+M} T_{n+M} R_{n-M} T_{n-M} R_{n-3M} T_{n-3M}

end

위의 스케줄이 된 알고리즘에서 각 열은 한 심벌 싸이클에 수행되는 operation들을 나타낸다. Software pipelining technique은 다음과 같이 loop를 unroll하면 쉽게 이해할 수 있다.

$\underline{W_{n+2M}}$ R_{n+2M-1} T_{n+2M-1} R_{n-1} T_{n-1} R_{n-2M-1} T_{n-2M-1}
$\underline{W_{n+2M+1}}$ R_{n+2M-2} T_{n+2M-2} R_{n-2} T_{n-2} R_{n-2M-2} T_{n-2M-2}
: : : : : : :
$\underline{W_{n+3M-1}}$ R_{n+M} T_{n+M} R_{n-M} T_{n-M} R_{n-3M} T_{n-3M}
W_{n+3M} $\underline{R_{n+3M-1}}$ $\underline{T_{n+3M-1}}$ R_{n+M-1} T_{n+M-1} R_{n-M-1} T_{n-M-1}
W_{n+3M+1} $\underline{R_{n+3M-2}}$ $\underline{T_{n+3M-2}}$ R_{n+M-2} T_{n+M-2} R_{n-M-2} T_{n-M-2}
: : : : : : :
W_{n+4M-1} $\underline{R_{n+2M}}$ $\underline{T_{n+2M}}$ R_n T_n R_{n-2M} T_{n-2M}
W_{n+4M} R_{n+4M-1} T_{n+4M-1} $\underline{R_{n+2M-1}}$ $\underline{T_{n+2M-1}}$ R_{n-1} T_{n-1}
W_{n+4M+1} R_{n+4M-2} T_{n+4M-2} $\underline{R_{n+2M-2}}$ $\underline{T_{n+2M-2}}$ R_{n-2} T_{n-2}
: : : : : : :
W_{n+5M-1} R_{n+3M} T_{n+3M} $\underline{R_{n+M}}$ $\underline{T_{n+M}}$ R_{n-M} T_{n-M}
W_{n+5M} R_{n+5M-1} T_{n+5M-1} R_{n+3M-1} T_{n+3M-1} $\underline{R_{n+M-1}}$ $\underline{T_{n+M-1}}$
W_{n+5M+1} R_{n+5M-2} T_{n+5M-2} R_{n+3M-2} T_{n+3M-2} $\underline{R_{n+M-2}}$ $\underline{T_{n+M-2}}$
: : : : : : :
W_{n+6M-1} R_{n+4M} T_{n+4M} R_{n+2M} T_{n+2M} $\underline{R_n}$ $\underline{T_n}$

여기서 밑줄 치어진 부분들을 보면 처음의 알고리즘과 같은 것을 알 수 있다. 스케줄이 된 알고리즘은 write, traceback 1, traceback 2, decode의 4개의 pipeline stage를 가지는 형태로 되어 있다.

이와 같이 스케줄이 된 알고리즘은 resource allocation과 binding에 의해 실제 구현에서 다양한 형태를 가질 수 있다. 먼저 기존의 방법들이 어떠한 형태로 resource allocation과 binding을 했는지 살펴본다.

Ⅲ. 기존의 Traceback 구조

기존의 방법들과 새로운 방법을 비교하기 위해 한 심벌 싸이클 동안 한 개의 메모리에 하나의 읽기와 하나의 쓰기가 가능하다고 가정한다. 만약 심벌 싸이클이 길어서 한 심벌 싸이클 동안 여러 개의 읽고 쓰기가 가능하다면 그에 따른 구현 구조의 수정이 필요하지만 전체적인 토의는 비슷하다.

스케줄링이 된 알고리즘을 보면 한 열에 한 개의 쓰기와 3개의 읽기가 있다. 다시 말해서 한 심벌 싸이클 동안 한 개의 쓰기와 3개의 읽기를 행하여야 한다. 이를 각각 W1, R1, R2, R3라고 하자. 이 논문에서는 한 심벌 싸이클 동안 한 개의 메모리에 하나의 읽기와 하나의 쓰기가 가능하다고 가정하였으므로 이들을 하나의 메모리를 사용하여 읽고 쓸 수는 없으며 병렬성을 증가시키기 위하여 복수의 메모리를 사용하여야 한다. 복수개의 메모리가 주어졌을 때 W1을 어떤 특정한 하나의 메모리에 binding되었다고 하면 다른 메모리에는 데이터를 쓰지 않는 것이므로 데이터를 읽는 것도 의미가 없으며 따라서 알고리즘을 실현하지 못한다. 위의 알고리즘에서 여러 개의 메모리를 사용할 때의 binding은 iteration마다 다르게 binding이 되는 periodic binding을 해서 모든 메모리에서 쓰고 읽는 것이 행해지도록 해야 한다. [6]과 [7]에 있는 방법은 3개의 메모리를 사용할 경우 표 1에 나와 있는 것과 같이 6개의 iteration을 한 period로 하는 periodic binding을 한다. 표 1에서 iteration 1의 RAM1은, R3으로 소모된 부분은 더 이상 필요하지 않으므로 W1의 쓰기가 사용할 수 있어서, 다른 RAM에 비해 적은 메모리를 필요로 한다. 그러나 iteration에 따라서 RAM의 용도가 바뀌므로 세 개의 메모리는 그 크기가 같아야 하며 따라서 RAM1을 다른 RAM보다 작게 만들지는 못한다. 이 방법은 필요한

메모리 요구량보다 많은 메모리를 사용하므로 메모리의 낭비가 있다.

표 1. [6]과 [7]에 의한 binding
Table 1. Binding by [6] or [7].

	RAM 1	RAM 2	RAM 3
iteration 1	W1, R3	R2	R1
iteration 2	R1, W1	R3	R2
iteration 3	R1	W1,R3	R2
iteration 4	R2	R1,W1	R3
iteration 5	R2	R1	W1,R3
iteration 6	R3	R2	R1,W1

[8]과 [9]에 나와 있는 방법은 메모리 요구량을 줄이기 위해 5개의 메모리를 사용하여 10개의 iteration을 한 period로 하는 periodic binding을 한다 (표 2). Binding의 pattern이 두 번씩 반복되는 이유는 앞에서의 방법과는 달리 W1과 R3이 최소 크기의 메모리를 공유함으로 인해 읽고 쓰기의 방향이 순방향과 역방향으로 번갈아 가며 진행되기 때문이다. [8]과 [9]의 방법은 각 메모리의 크기가 [6]과 [7]에서의 메모리 크기의 반인데 반해 메모리의 수는 두 배보다 하나 적으므로 전체적인 메모리 요구량은 줄었다. 그러나 메모리의 수가 증가했으며 더 복잡한 컨트롤이 필요하다. 일반적으로 메모리 크기가 반이 되더라도 어드레스 디코딩 부분 등은 반이 되기 어려우므로 메모리의 면적은 반이 되지 않으며 메모리의 수가 증가하면 interconnection 부분도 증가할 수 있다. 복수개의 작은 메모리의 사용은 비용면에서 불리할 수 있다.

표 2. [8]과 [9]에 의한 binding
Table 2. Binding by [8] or [9].

	RAM1	RAM2	RAM3	RAM4	RAM5
iteration 1	W1,R3		R2		R1
iteration 2	R1	W1,R3		R2	
iteration 3		R1	W1,R3		R2
iteration 4	R2		R1	W1,R3	
iteration 5		R2		R1	W1,R3
iteration 6	W1,R3		R2		R1
iteration 7	R1	W1,R3		R2	
iteration 8		R1	W1,R3		R2
iteration 9	R2		R1	W1,R3	
iteration 10		R2		R1	W1,R3

Ⅳ. 새로운 Traceback 구조

기존의 traceback 구현에서 binding이 복잡해지는 이유는 한 심벌 싸이클 동안 여러 개의 읽기가 있지만 오직 하나의 쓰기만이 존재하기 때문이다. 저장된 데이터는 여러 iteration 동안 사용되어야 하므로 다음 iteration의 쓰기가 같은 영역을 사용할 수 없어서 periodic binding의 필요성이 생기며 periodic binding은 memory scheme을 복잡하게 만든다. 이 논문에서는 보다 간단한 binding을 얻기 위해 dummy write를 추가하여 읽고 쓰기 동작의 균형을 맞추는 방법을 제안한다. 추가된 쓰기 동작은 특정 메모리에서 다른 메모리로의 데이터 이동을 발생시켜 여러 iteration동안 데이터를 저장하기 위해 periodic binding을 해야 할 필요성을 제거한다. 아래의 알고리즘에서 굵게 표시된 부분이 추가된 dummy write이다.

for n = 1 to N+3M step M
begin
$W_{n+2M}R_{n+2M-1}T_{n+2M-1}$ $\boldsymbol{W_{n+2M-1}}$ $R_{n-1}T_{n-1}$ $\boldsymbol{W_{n-1}}$ $R_{n-2M-1}T_{n-2M-1}$
$W_{n+2M+1}R_{n+2M-2}T_{n+2M-2}$ $\boldsymbol{W_{n+2M-2}}$ $R_{n-2}T_{n-2}$ $\boldsymbol{W_{n-2}}$ $R_{n-2M-2}T_{n-2M-2}$
: : : : : : : : :
$W_{n+3M-1}R_{n+M}T_{n+M}$ $\boldsymbol{W_{n+M}}$ $R_{n-M}T_{n-M}$ $\boldsymbol{W_{n-M}}$ $R_{n-3M}T_{n-3M}$
end

이제 각 열에 3개의 쓰기 동작과 3개의 읽기 동작이 존재한다. 이를 각각 W1, W2, W3, R1, R2, R3라고 하자. 3개의 메모리를 사용한 binding은 표 3과 같다.

표 3. 제안된 binding
Table 3. Proposed binding.

RAM1	RAM2	RAM3
W1,R1	W2,R2	W3,R3

제안된 방법에서 첫 번째 쓰기 W1은 어드레스가 *n+2M, n+2M+1, ...* 과 같이 매 step마다 하나씩 증가하고 있고 읽기 R1은 어드레스가 *n+2M-1, n+2M-2, ...* 과 같이 매 주기마다 가장 최근의 데이터부터 시작하여 매 step마다 하나씩 감소하고 있다. 이와 같이 데이터를 하나씩 쓰면서 M step마다 가장 최근의 데이터부터 역순으로 읽어내는 것은 LIFO (Last In First Out buffer)라고 할 수 있다. LIFO

의 크기는 최대 한 주기의 데이터, 즉 M개의 데이터를 기억하고 있어야 하므로 M (= L/m) 이다. 두 번째 쓰기 W2는 어드레스가 *n+2M-1, n+2M-2, ...* 과 같고 두 번째 읽기 R2는 어드레스가 *n-1, n-2, ...* 과 같으므로 쓰여진 데이터는 정확히 2M step 후에 읽어지는 것이며 이 것은 2M의 크기를 가지는 FIFO에 해당된다. 세 번째의 쓰기와 읽기도 이와 유사하며 역시 2M의 크기를 가지는 FIFO에 해당된다. 첫 번째 FIFO의 입력은 LIFO의 출력과 같으므로 LIFO와 첫 번째 FIFO는 직렬로 연결된다. 마찬가지로 두 번째 FIFO의 출력과 첫 번째 FIFO의 입력이 같으므로 두 개의 FIFO도 직렬로 연결된다. 따라서 세 개의 메모리는 모두 직렬로 연결된 형태가 된다.

Traceback 알고리즘은 데이터를 메모리에 쓰고 쓴 반대의 순서로 데이터를 읽어내면서 traceback하는 것이므로 기본적으로 LIFO와 유사하다고 할 수 있다. 특히 packet mode로 동작할 때에는 한 packet의 데이터가 모두 쓰여지고 쓴 반대의 순서로 traceback을 하므로 LIFO와 일치한다. 고속 Viterbi decoder의 경우 continuous mode로 동작할 때에도 packet mode와 유사하게 한 번의 L traceback을 통해서 여러 개의 데이터를 복호해낸다. Traceback 알고리즘에서 처음 M 쓰기를 하고 M 읽기를 하는 부분은 M의 크기를 가지는 LIFO와 유사하다. Packet mode로 동작하는 경우와의 차이점은 데이터의 쓰기를 하는 부분보다 traceback과 decode를 하는 부분이 길다는 것이다. 즉 traceback은 단지 M 읽기에 그치지 않고 계속적으로 이전 데이터를 읽어낸다. 이 것은 처음 M 쓰기를 한 데이터가 다음 iteration을 위해 보존되어야 한다는 것을 뜻하며 데이터를 읽어내면 메모리 영역에서 사라지는 LIFO와는 차이가 있다. 제안된 방법은 처음 M 쓰기와 M 읽기를 LIFO로 만들기 위해 읽어낸 데이터를 다시 쓰기를 하여 다른 장소에 저장한다. 이렇게 추가된 쓰기는 다음의 읽기와 함께 FIFO의 형태를 이룬다. 이 번에는 LIFO가 아니라 FIFO가 된 이유는 LIFO에서 읽기의 순서가 이미 뒤집어졌기 때문이다. 즉 처음의 쓰기와 traceback의 읽기는 그 순서가 반대이지만 같은 traceback에서 처음부분과 두 번째 부분은 읽기의 순서가 같으므로 traceback의 처음부분을 그 순서대로 쓰기를 행하면 두 번째 읽기와는 같은 순서가 되는 것이다. 같은 방법으로 나머지 부분들도 모두 FIFO의 형태로 변환되어 한 개의 LIFO와 m개의 FIFO를 가지는 형태가 된다.

그림 1에 제안된 메모리 관리 방법에 따른 메모리 구조가 나와 있다. 위 부분에 앞에서 설명한 하나의 LIFO와 두 개의 FIFO가 있다. 이들 3개의 메모리는 한 심벌 싸이클동안 하나의 데이터를 쓰고 하나의 데이터를 읽는 매우 간단한 구조의 LIFO와 FIFO이다. 하나의 LIFO와 두 개의 FIFO는 앞에서 설명한 것과 같이 서로 직렬로 연결되어 있다. 아래 부분은 메모리 읽기 쓰기를 제외한, traceback을 하기 위해서 필요한 동작들로서 다른 모든 메모리 관리 방법에서도 필수적으로 들어가는 부분이다. 그림의 trace logic은 traceback 알고리즘에서 설명한 Tn에 해당하는 부분으로서 읽어낸 2^{K-1}비트의 정보와 현재까지 계산된 state address를 이용하여 다음의 state address를 계산해낸다. 반복적으로 state address를 계산함으로서 traceback 동작과 Viterbi 복호가 행해지며 이를 위한 데이터는 LIFO와 FIFO에 의해 공급받는다. 윗줄에 있는 LIFO의 결과는 traceback 동작을 하기 위해 trace logic으로 가는 동시에 뒤에 연결되어 있는 FIFO의 입력으로 들어간다. 마찬가지로 FIFO의 결과도 trace logic과 다음의 FIFO의 입력으로 들어간다. 여기서 LIFO의 크기는 M = L/m이며 FIFO의 크기는 2M = 2L/m이다. Traceback 알고리즘은 데이터 입력순서의 역순으로 데이터를 복호하는 것이므로 LIFO를 사용하여 sequence의 순서를 뒤집는 작업이 필요하다. Trace logic 뒤에 연결되어 있는 마지막의 LIFO는 이 역할을 하며 그림의 다른 메모리들의 word length가 2^{K-1}비트인데 반해 word length가 1이므로 그 크기는 매우 작다.

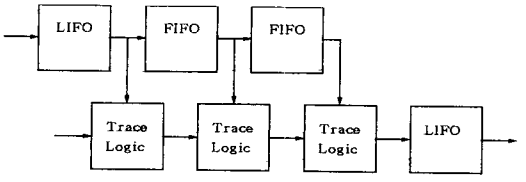

그림 1. 새로운 traceback 구조
Fig. 1. New traceback structure.

V. 토 론

제안된 traceback 구조와 기존의 traceback 구조의 비교가 표 4에 나와있다. VLSI 구현에서 비용, 즉 면적은 컨트롤 로직의 면적, 메모리의 면적,

interconnection 면적 등으로 나눌 수 있다. 컨트롤 로직의 경우 면적은 알고리즘의 복잡도에 비례하는데 [6], [7], [8], [9] 의 방법은 periodic binding을 요구하여 복잡한 메모리 어드레스 발생을 필요로 한다. 이에 반해 제안된 알고리즘은 심벌 싸이클당 한 개의 데이터를 받아들이고 내보내는 간단한 구조의 FIFO와 LIFO로 구성되므로 구현이 간단하고 복잡도가 적다. 메모리의 면적은 메모리 요구량에 비례하며 적은 수의 메모리를 사용하는 것이 유리한데 [6], [7] 의 방법은 불필요하게 메모리 요구량을 증가시켰으며 [8], [9] 의 방법은 최소의 메모리 요구량을 사용하지만 메모리의 수를 많이 증가시킴으로서 메모리 면적의 증가를 초래한다. 이에 반해서 제안된 방법은 최소의 메모리 요구량과 최소의 메모리 수를 동시에 만족시킨다. 이 것이 가능해진 것은 periodic binding을 사용하지 않음으로서 모든 메모리가 같은 크기일 필요가 없기 때문이다. 그리고 [6], [7], [8], [9] 의 방법이 2^{K-1}비트의 버스가 모든 메모리에 연결되는 global interconnection 구조를 가지는 데 반해 제안된 메모리 관리 방법은 local interconnection만을 가짐으로서 불필요한 interconnection 면적이나 interconnection delay의 증가를 억제한다.

표 4. 제안된 방법과 기존의 방법과의 비교
Table 4. Comparisons of traceback methods.

	proposed	[6] [7]	[8] [9]
# of RAMs	m+1	m+1	2m+1
memory length per RAM	L/m, 2L/m	2L/m	L/m
total memory length	2L+L/m	2L+2L/m	2L+L/m
bit width	2^{K-1}	2^{K-1}	2^{K-1}
binding period	1 iteration	2m+2 iterations	4m+2 iterations
interconnection	local	global	global

[10] 의 방법은 이 논문에서 제안된 방법에서 m = L, 즉 M = 1인 특별한 경우이다. 이 경우 LIFO의 깊이는 1이며 FIFO의 깊이는 2이다. 이 방법은 전체 메모리 요구량과 복호 지연시간을 줄인 반면 interconnection과 trace logic의 지나친 증가를 초래하여 VLSI 구현에는 적합하지 않다.

제안된 방법은 두 가지 크기의 메모리를 사용하여 메모리의 regularity를 감소시켰다. 그러나 두 가지 크기의 메모리를 사용하는 것은, design flow의 발달로 인해, 커다란 문제가 되지는 않는다. RAM generator의 발달은 여러 가지 크기의 메모리를 쉽게 만들어낼 수 있게 하였으며 auto-layout tool의 발달은 standard cell 영역이 직사각형일 필요가 없게 하여 두 가지 크기의 메모리를 사용하여도 unused area가 발생하지는 않게 되었다.

VI. 결 론

이 논문에서는 고속 Viterbi 복호기를 위한 새로운 traceback 구조를 소개했다. 새로운 traceback 구조는 최소의 메모리 요구량과 최소의 메모리 수를 사용하면서도 메모리 어드레스 발생 방법이 간단하며 global interconnection이 존재하지 않아 VLSI 구현에 적합하다.

참 고 문 헌

[1] The European Digital Video Broadcasting Project, Digital Video Broadcasting.

[2] G.D. Forney, JR, "The Viterbi Algorithm", *Proc. of IEEE*, vol. 61, no. 3, march 1973.

[3] G.C. Clark and J.B. Cain, Error Correction Coding for Digital Communications, Plenum Press, 1981.

[4] C.M. Rader, "Memory management in a Viterbi Decoder", *IEEE Trans. on Communications*, vol. 29, no. 9, September 1981.

[5] M. Lam, "Software Pipelining: An Effective Scheduling Technique for VLIW Machines", *Proc. of the ACM SIGPLAN'88 Conference on Programming Language Design and Implementation*, June 1988.

[6] H.A. Bustamante et al., "Stanford Telecom VLSI Design of a Convolutional Decoder", *IEEE Conference on Military Communications*, Boston, Massachusetts, October 1989.

[7] R. Cypher and C.B. Shung, "Generalized

Trace Back Techniques for Survivor Memory Management in the Viterbi Algorithm", *Globecom*, December 1990.

[8] O. Collins and F. Pollara, "Memory Management in traceback Viterbi Decoders", *TDA Progress Report 42-99, jet Propulsion Laboratory*, Pasadena, California, 1989.

[9] G. Feygin and P.G. Gulak, "survival Sequence Memory Management in Viterbi Decoders", *Proc. 3rd Workshop on ECC*, September 1989.

[10] T.K. Troung et al, "A VLSI Design for a Trace-Back Viterbi Decoder", *IEEE Trans. on Communications*, vol. 40, no. 3, March 1992.

저 자 소 개

任 敏 中(正會員)

1965년 1월 30일생. 1987년 서울대학교 전자공학과 졸업. 1990년 Wisconsin 주립대 Electrical and Computer Engineering 석사. 1993년 Wisconsin 주립대 Electrical and Computer Enineering 박사. 1993년 ~ 1998년 현재 삼성전자 반도체 총괄 System LSI 본부 선임연구원. 주관심분야는 VLSI CAD and design for DSP and communications