論文2003-40SD-1-5

변형된 레지스터 교환 방식의 비터비 디코더 설계

(Design of Viterbi Decoders Using a Modified Register Exchange Method)

李 燦 豪 *, 盧 承 孝 **

(Chanho Lee and Seung-hyo Noh)

요 약

본 논문에서는 비터비 디코더의 디코딩과정에서 trace-forward 과정이후, trace-back 동작 없이 decision bit를 결정 가능한 구조로 설계하여 사용 메모리 크기와 동작 cycle에서 이득을 가지는 변형된 레지스터 교환(modified register exchange) 방식을 제안하였다. 제안된 구조는 시뮬레이션에 의해 trace-back이 있는 기존의 방식과 동일한 결과를 나타냄을 확인하였으며, 변형된 레지스터 교환 방식과 기존의 레지스터 교환 방식, 그리고 trace-back 방식과 비교하였다. 제안한 방식은 다른 방식들에 비해 메모리를 1/(5 x constraint length)로 줄일 수 있고, trace-back 방식에 비해 throughput을 2배 향상시켰다. 변형된 레지스터 교환 방식을 적용한 비터비 디코더의 동작을 검증하기 위해 code rate 2/3, constraint length, K가 3인 디코더를 radix-4 구조의 1 bit 디코딩 방식으로 설계하여 FPGA(field programmable gate array)를 이용하여 구현하고, 측정을 통해 오류 정정 작용을 확인하였다. 또한 블록 디코딩 방식에도 적용할 수 있음을 보였다.

Abstract

This paper proposes a Viterbi decoding scheme without trace-back operations to reduce the amount of memory storing the survivor path information, and to increase the decoding speed. The proposed decoding scheme is a modified register exchange scheme, and is verified by a simulation to give the same results as those of the conventional decoders. It is compared with the conventional decoding schemes such as the trace-back and the register exchange scheme. The memory size of the proposed scheme is reduced to 1/(5 x constraint length) of that of the register exchange scheme, and the throughput is doubled compared with that of the trace-back scheme. A decoder with a code rate of 2/3, a constraint length, K=3 and a trace-back depth of 15 is designed using VHDL and implemented in an FPGA. It is also shown that the modified register exchange scheme can be applied to a block decoding scheme.

Keyword : Viterbi decoder, register exchange, traceback, VLSI, block decoding, TCM

* 正會員, 崇實大學校 情報通信電子工學部
(Soongsil University, School of Electronic Engineering)
** 正會員, 三星電子 시스템 LSI 事業部 LSI 開發 1팀
(Smart card design team, System LSI division, Samsung Electronics Co. inc.)
接受日字:2001年7月31日, 수정완료일:2002年12月17日

Ⅰ. 서 론

오늘날, 통신시스템에서 오류 정정 코딩은 중요한 역할을 한다. 오류 정정 코딩은 실제 전송과정에서 필연적으로 존재하는 오류를 감지하고 정정하도록 전송자료에 redundancy를 추가하는 기능을 가지고 있다. 전송자료에 redundancy를 추가하고, 연판정(soft-decision)

디코딩을 이용하여 값을 얻으므로, 전송전력의 증가 없이 BER(bit-error rate)을 상당히 줄일 수 있다. 이와 같은 코딩 기술들은 GSTNs(general switched telephone networks)와 같은 유한 전력 채널에서의 전송에 특히 유용하다. 송신단에서 길쌈(convolutional) 인코더가 사용되면, 수신단에서는 전송자료를 추정하기 위해 비터비 알고리즘(Viterbi algorithm)을 사용한다. 비터비 알고리즘은 수신한 데이터의 판단과정에서, 에러확률을 적게 하기 위해 가능성이 높은 값으로 디코딩 하는 것이다.[1,2] 만약, L-bit codeword가 있다면, 그에 따른 가능한 값은 2L 개가 되는데 그중 가장 유사한 값을 찾아내는 것이다.

비터비 알고리즘의 구현에 사용되는 일반적인 디코딩방법으로 레지스터 교환 방법(register exchange)과 trace-back 방법이 있다.[3~7] 레지스터 교환 방식은 trace-forward 과정에서 생존 경로(survivor path)를 기록하기 위한 decision bit를 저장할 때 현재의 stage에 해당되는 decision bit를 포함한 모든 decision bit을 멀티플렉서(multiplexor)와 읽기/쓰기가 동시에 가능한 메모리를 이용하여 재정렬시켜 trace-back 과정 없이 디코딩 데이터를 구하는 방식이다. 이 방식은 상태(state) 수와 생존 경로 길이(depth)의 곱만큼의 멀티플렉서, 그리고 dual port 메모리가 필요하다. 그러므로, 많은 면적과 전력소비가 뒤따르게 되어 현재는 trace-back 방식이 주로 사용되고 있다. Trace-back 방식은 trace-forward 과정에서 저장한 decision bit를 이용하여 생존 경로 길이만큼 진행한 후 생존 경로를 따라 trace back을 하여 디코딩 데이터를 얻는 방식이다. Trace-back은 저장되어 있는 decision 값을 이용하여 생존 경로를 역추적 하므로, trace-forward 과정 속에서 발생한 모든 decision 값이 저장되어 있어야 한다. 또한, 디코딩 값을 구한 후, 다시 trace-forward 과정이 시작될 상태가 결정되므로, trace-back 과정을 통하여 디코딩 데이터가 결정되기 전까지 다음 디코딩과정을 실행할 수 없다. 레지스터 교환 방식에 비해 동작 cycle은 증가하나, 메모리부분에서의 이득으로 인해 가장 많이 사용되고 있는 방식이다.

비터비 디코더에서 입력값이 연속적으로 들어오는 경우 디코딩된 값을 얻기 위해서는 초기 stage의 값들만 필요하고, 중간에 저장된 값은 필요가 없다. 따라서 trace-forward 이후 초기 상태와 관련값만 알 수 있다면, 두 번째 이후 stage의 값은 저장할 필요가 없고, 따라서 많은 메모리가 필요하지 않게 된다. 즉, 레지스터 교환 방식에서 (trace-back depth) x (상태수) 만큼의 메모리에 중간값을 모두 저장하는 대신, 상태 수만큼의 메모리에 초기값 만을 저장하고 이 값들에 대해서만 path metric 값의 계산 결과에 따라 레지스터 교환 작용과 동일한 방법으로 수행하면 원하는 동작을 하게 된다. 따라서 trace-back 동작이 없는 장점과 적은 메모리로 동작하는 장점을 갖는 비터비 디코더를 설계할 수 있다. 이러한 구조의 '변형된 레지스터 교환 방식(modified register exchange)'에서는, 생존 경로 길이만큼 진행한 후 출력될 디코딩 데이터를 trace-forward 과정에서 살아남는 path를 따라 고정된 동일한 메모리를 반복 사용하여, 두 번째 stage 이후의 decision 값의 저장 공간이 필요하지 않으며, 생존 경로가 결정되면 바로 디코딩 데이터가 결정되므로 trace-back 실행시간이 단축되어 디코딩 시간이 줄어들게 된다. 반면에 ACS(Add-Compare-Select) 블록이 조금 복잡해지고 입력 데이터의 크기가 작을 경우에는 일반적인 디코딩 방식에 비해 디코딩 시간이 길어지는 단점이 있다. 그러나, 복잡성대신 얻어지는 속도향상과 메모리 면적 감소의 이득이 더 크고, 일반적인 시스템의 데이터는 수십 bit를 넘어가는 경우가 대부분이므로 데이터의 크기에 따른 단점은 큰 문제가 안 된다.

본 논문에서는 trace-back이 없는 레지스터 교환 방식의 장점을 취하면서 필요한 메모리 크기를 크게 줄인 변형된 레지스터 교환 방식의 비터비 디코더의 구조를 제안하였다. 또한 이를 적용한 디코더를 IDEC에서 지원한 MAX+plusII를 이용하여 설계하고 FPGA에 구현하여 그 동작을 확인하였다.

Ⅱ. 변형된 레지스터 교환 방식

1. 동작원리와 구조

Trace-back 동작이 없는 변형된 레지스터 교환 방식의 비터비 디코딩 방식은 trace-forward를 해 나가면서 시작 상태에서의 생존 경로에 의해 생성된 값만을 저장하고 survivor path를 따라 저장위치를 바꾸어준다. Trace-forward가 끝나면서 생존 경로가 결정되는 순간 디코딩 데이터를 결정하므로 trace-back이 필요 없다. 따라서, <그림 1>과 같은 구조를 가지며, 블록도가 단순해진 만큼 core의 크기를 줄일 수 있다. 여기서

transition memory unit은 입력 버퍼로서 입력된 심볼이 여러 번 사용되므로 이 값을 저장하는 역할을 하고 레지스터에는 path metric 값과 decision bit가 저장된다.

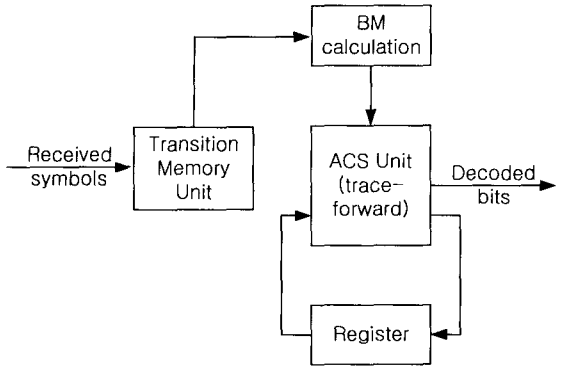
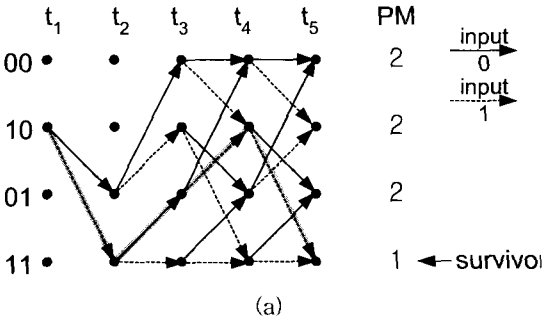


그림 1. 변형된 레지스터 교환 방식의 Viterbi decoder 구조. Trace-back 구조와는 달리 trace-back unit이 없다

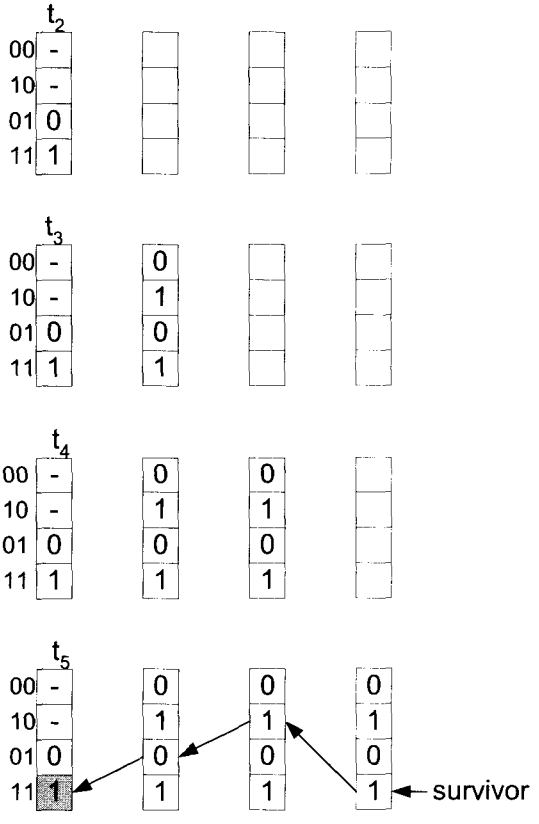
Fig. 1. Structure of a Viterbi decoder using the modified register exchange scheme. It is found that the trace-back unit is eliminated.

<그림 2>에 디코딩 방법에 따른 메모리 크기와 내용의 변화가 나타나 있다. <그림 2(a)>는 디코딩을 위한 trellis diagram으로 시작 상태는 '10'이고 trace-back depth는 4이므로 시간 t_1부터 t_5까지 진행하면서 생존 경로를 결정한다. <그림 2(b)>의 Trace-back 방법의 경우 메모리는 4 × 4 = 16개가 필요하다. Trellis diagram상에서 한 단계씩 진행하면서 메모리에 decision bit이 차례로 채워져 t_5에 이르면 모든 메모리가 값을 가지게 된다. 이 때 생존 경로가 결정되고 저장된 decision bit 값에 따라 한 단계씩 역추적하여 디코딩 데이터와 다음 시작 상태를 결정한다. <그림 2(c)>의 레지스터 교환 방식은 사용하는 메모리의 크기와 trace-forward 과정에서 모든 값을 저장하는 것은 trace-back 방식과 같지만 path metric이 결정될 때마다 레지스터에 저장된 값들을 재배열하여 마지막에 선택된 상태와 같은 열에 생존 경로가 일직선으로 위치하도록 하여 trace-back 필요성을 없앤 것이다. 그러나 레지스터의 재배열 과정에서 많은 전력 소모가 일어나는 단점이 있다.

만일 연속된 입력값이 들어온다면 첫 번째 행의 메모리에 저장된 값만이 디코딩 값을 결정하는데 이용되고 나머지 저장된 값들은 모두 버리게 되므로 생존 경로상의 데이터를 모두 저장할 필요가 없어진다. 따라서 4개의 메모리만 설정하고 레지스터 교환 방식에서의 첫 번째 행에 저장된 값들만 trace-forward를 진행하면서 메모리의 데이터 위치를 계속 바꾸어 주면 같은 결과를 얻을 수 있다. 이 경우에도 trace-forward 과정이 끝나면 선택된 메모리로부터 시작 상태 정보와 디코딩 데이터를 알 수 있다. 이것이 <그림 2(d)>에 나타난 변형된 레지스터 교환 방식이다.



(a)



(b)

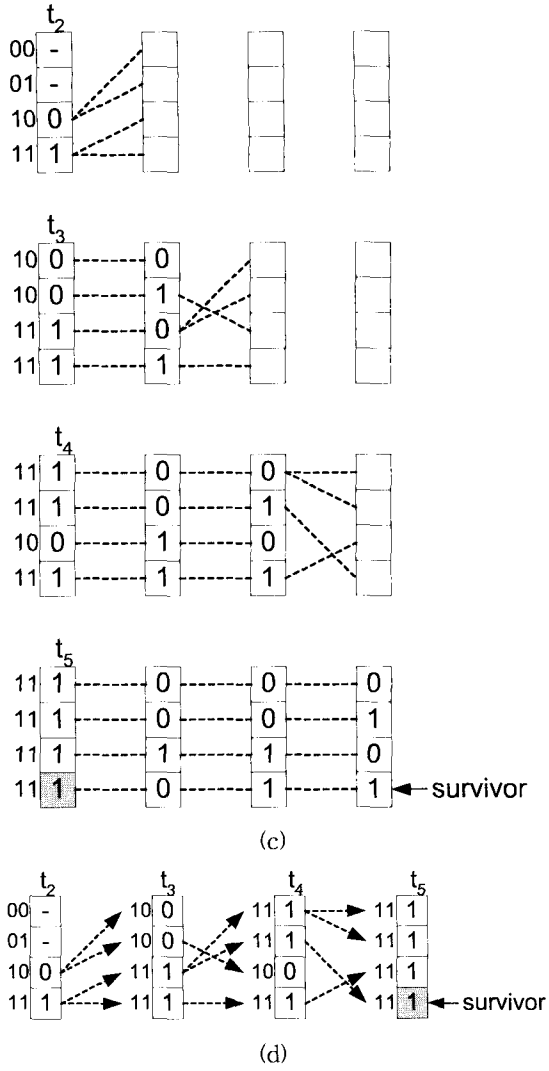


그림 2. 디코딩 방법에 따른 메모리 크기와 t_2-t_5 사이의 메모리 내용의 비교. (a) Trellis diagram (b) Trace-back 방법 (c) 레지스터 교환 방법 (d) 변형된 레지스터 교환 방법

Fig. 2. Comparison of the memory size and the contents of memory for operation cycles of t_2-t_5 (a) Trellis diagram (b) Trace-back scheme (c) Register exchange scheme (d) modified register exchange scheme.

Trace-back 동작이 없이 디코딩이 이루어짐을 확인하기 위하여, code rate 1/2, K=3인 8-level soft decision 비터비 디코더를 변형된 레지스터 교환 디코딩 방식과 기존의 trace-back 디코딩 방식을 이용하여 2,000,000개의 데이터에 대해 시뮬레이션 하여 <그림 3>과 같이 비슷한 성능을 가짐을 확인하였다.

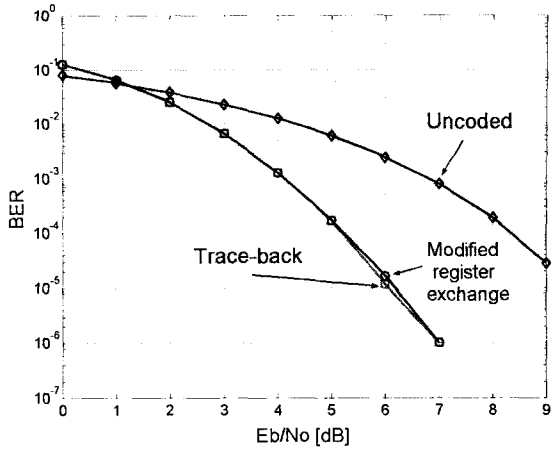


그림 3. K=3, R=1/2인 Viterbi decoder의 BER 시뮬레이션 결과

Fig. 3. BER simulation result of a Viterbi decoder with K=3 and R=1/2.

2. 성능 비교 및 장단점

변형된 레지스터 교환 방식의 장점을 알아보기 위해 기존의 trace-back 방식과 레지스터 교환 방식에 대해 메모리 크기와 동작 cycle, 그리고 throughput에 대한 비교를 <표 1>에 나타내었다. 비터비 디코더는 constraint length K의 4~5 배 정도면 에러 정정에 거의 문제가 없으므로, trace-forward 동작의 생존 경로 길이는 4K~5K가 되고, 그 동작에 따른 decision 값을 저장하여야 한다[1-2]. 여기서 디코더의 입력 버퍼는 생존 경로 길이와 동일한 수의 입력값을 저장하고 있어야 한다. 또한, 다음 디코딩 동작에서 입력으로 받아놓은 생존 경로 길이 만큼의 데이터 중 디코딩된 영역을 제외한 나머지 부분을 제거하고 새로 들어온 데이터와 함께, 다음 survivor path를 구할 때 사용한다. Trace-forward 동작이 끝난 후, 최종 생존 경로가 결정되면 이 결과에 의해 trace-back 동작이 시작되며, 생존 경로 길이 만큼 동작하여야 하므로 생존 경로 길이 만큼의 clock을 필요로 한다. 따라서, trace-back 동작이 있는 일반적인 비터비 디코더를 IS95 reverse link(code rate=1/3, K=9)에 맞추어 구현하면, trace-forward 동작 중에 발생하는 decision 값을 저장하기 위해서는 40(survivor path depth) × 256(state) × 1(decision bit) = 10,240 bit의 저장공간과 80 clock cycle이 필요하다. 이에 반해, 변형된 레지스터 교환 비터비 디코더는 trace-forward 동작을 위한 40 clock 만에 디코딩 결과를 얻을 수 있으며, trace-forward 동작 중에 발생

하는 decision bit를 모두 저장할 필요가 없으므로, 256bit만을 필요로 한다. 이에 따라, 변형된 레지스터 교환 방식의 디코딩이 trace-back 디코딩 방식에 비해 survivor path memory 크기는 1/L(IS95 reverse link의 경우 1/40)만큼 감소하고, 동작 cycle은 1/2로 감소하며, 디코딩 속도(throughput)는 2배의 향상을 기대할 수 있다. 이러한 디코딩 방식은 블록 디코딩방식에[8] 대해서도 적용할 수 있다. <표 1>의 괄호 안에 본 논문에서 설계한 K=3, code rate 2/3인 디코더와 IS95 reverse link에 이용되는 code rate 1/3, K=8인 디코더의 메모리, 하나의 디코딩 데이터를 얻는데 필요한 동작 clock 수, throughput이 나타나 있다.[9~11] K값이 커질수록 메모리에서 얻는 이득이 크고 동작 cycle에서의 이득도 크게 증가함을 알 수 있다. <표 1>의 결과는 branch metric(BM) 값과 path metric(PM) 값을 매번 갱신하고 trace-forward를 시작할 때 초기값을 아는 상태에서 출발하는 조건에서 구한 값들이다. 이 경우가 BER이 가장 좋은 경우이다. 그러나 동작 속도를 증가시키기 위해서 초기값이 결정되기 전에 pipeline 구조 등을 이용하여 다음 trace-forward를 시작하는 방법이 있다. 이 경우 throughput은 증가하나 BER이 나빠지는 단점이 있다. 변형된 레지스터 교환 방식도 이러한 방법으로 구현이 가능하다. 즉 초기값을 모르는 상태에서 trace-forward를 시작하여도 decision bit을 결정하는데는 아무 지장이 없다.

표 1. 비터비 디코더의 구현 방식 비교
Table 1. Comparison of implementation methods of Viterbi decoders.

구현 / 비교항목	Trace-back decoding	Register Exchange	Modified Register Exchange
Survivor path memory	$L\times S\times d$ $= 5K\cdot 2^K\cdot k$ (240/10240)*	$L\times S\times d$ $= 5K\cdot 2^K\cdot k$ (240/10240)*	$S\cdot k$ $=2^K\cdot k$ (16/256)*
Number of clock to obtain a decoded bit	L+L = 2L = 10K (30/80)*	L = 5K (15/40)*	L = 5K (15/40)*
Throughput (decoded data per one clock cycle)	1/2L $(\frac{1}{30}/\frac{1}{80})$*	1/L $(\frac{1}{15}/\frac{1}{40})$*	1/L $(\frac{1}{15}/\frac{1}{40})$*

* : (Code rate 2/3, K=3 / Code rate 1/3, K=8)
L : survivor path length(=5×K), K : Constraint length, S : # of state(= 2^K)
d : decision bit, k : decoded data (code rate = k/n)

그러나, 변형된 레지스터 교환 디코딩은 항상 하나의 trace-forward path에 대해 하나의 디코딩 결과만을 출력하므로, 적은 량의 입력 데이터에 대해서는 불리하다. 즉 기존의 디코더에서 survivor path depth보다 작거나 같은 수의 입력에 대해서 한번의 디코딩 동작으로 모든 출력을 얻을 수 있으나 변형된 레지스터 교환 방법에서는 항상 입력 수만큼 디코딩 동작을 반복하여야 한다. <표 2>는 입력값의 크기에 따른 디코딩 효율을 계산한 결과이다. 계산 결과 생존 경로 길이의 2배 이상에 해당하는 연속적인 데이터량에 대하여는 변형된 레지스터 교환 방식의 디코딩 효율이 좋았다. 실제로 수십 bit미만의 데이터만 전송하는 경우는 거의 없으므로 대부분의 경우에 변형된 레지스터 교환 방식은 그 장점을 살릴 수 있다.

표 2. 디코딩 데이터 크기에 따른 효율 비교
Table 2. Comparison of decoding efficiency according to the decoding data size.

구현 / 비교항목		Trace-back decoding	Register Exchange	Modified Register Exchange
Decoding 크기에 따른 필요 cycle 수식		(5K+n)/ (10K×n)	(5K+n)/ (5K×n)	(5K+n) /(10K× (5K+n))
Decoding 크기 (5K+n) 에 따른 효율 비교(K=3가정)	15	0.5	1	0.0667
	28	0.0667	0.1333	0.0667
	100	0.0387	0.0775	0.0667
	3000	0.0335	0.0669	0.0667

K : Constraint length, n : decoding 동작 횟수

Ⅲ. 변형된 레지스터 교환 방식의 Viterbi Decoder 설계

1. PLD를 이용한 구현 및 측정결과

변형된 레지스터 교환 방식을 이용하여 비터비 디코더를 구현하여 디코딩 동작을 확인하기 위해 MAX+plusⅡ의 VHDL 기능을 이용하여 code rate 2/3, K=3인 비터비 디코더를 설계하였다. Radix-4 ACS 구조의 trace-back 동작이 있는 비터비 디코더는 trace-forward를 위한 15 clock과 trace-back을 위한 15 clock을 합쳐 30 clock을 필요로 한다. 이에 비해 변형된 레지스터 교환 비터비 디코더는 15 clock만을 필요

로 하므로, 2배에 가까운 속도향상을 얻을 수 있었다. 설계한 비터비 디코더는 Altera사의 FPGA 10K70RC 240-4에 구현하였으며, 전체 logic cell의 28%를 사용하고, 디코딩 동작에 필요한 메모리는 16bit만을 사용하였다. 비터비 디코더를 radix-2 ACS 구조로 구현하면, logic cell의 19%를 사용하여 radix-4 방식에 비해 사용하는 logic cell 수를 30% 정도 줄일 수 있다. 반면에 필요한 동작 cycle은 30 clock으로 늘어난다. <표 3>에 결과를 정리하였다.

표 3. 변형된 레지스터 방식의 비터비 디코더의 설계 결과. (Code rate 2/3, K=3, Altera FPGA 10K70RC240-4에 구현)

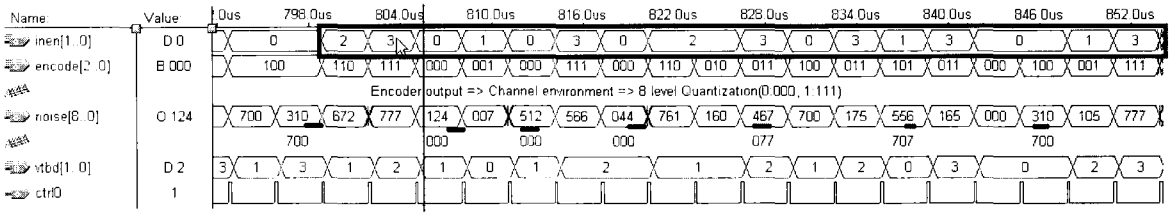
Table 3. Implementation results of a Viterbi decoder using the modified register exchange (Code rate 2/3, K=3, implemented on Altera FPGA 10K70RC240-4).

구현 / 비교항목	변형된 레지스터 교환 방식		Trace-back 방식	
	radix-4	radix-2	radix-4	radix-2
Survivor path memory	16bit	16bit	240bit	240bit
Cycle	15	30	30	60
FPGA cell 사용률	28%	19%	N/A	N/A

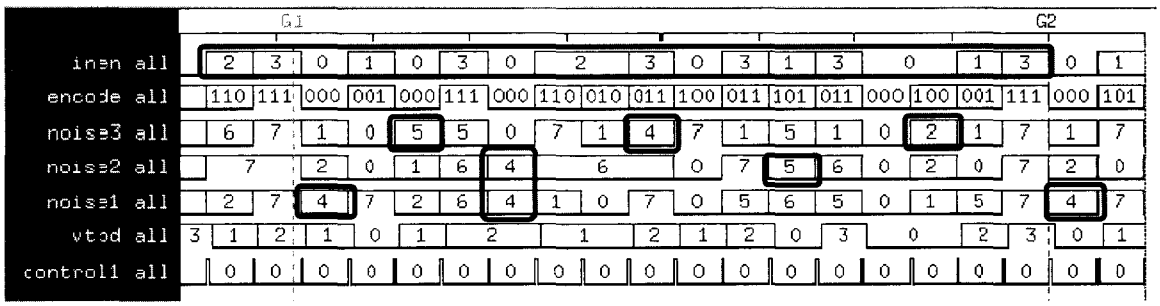
<그림 4>와 <그림 5>는 radix-4 비터비 디코더의 시뮬레이션 결과와 logic analyzer(HP 16702B)를 이용한 측정결과이다. <그림 5>에서 TCM(Trellis coded modulation) 인코더에 임의의 값을 입력하고 그에 따른 인코더의 출력 bit을 '0'은 '000', '1'은 '111'로 양자화 시킨 뒤 AWGN(Additive White Gaussian Noise) 분포에 따라 랜덤하게 변화(잡음)를 주어 출력값을 변형시킨 부분을 보여주고 있다. 이때 '0'이 '100'에서 '111', '1'이 '000'에서 '011' 사이의 값을 갖게되면 에러가 발생한 것으로 간주된다. <그림 6>은 변형된 출력값을 받아 디코더에서 디코딩되어 출력되는 부분을 보여주고 있다. <그림 4(a)>에서 표시된 부분에 noise[8..0]의 열의 아래에 표시된 8진수값이 송신단에서 출력된 값이다. 송신단의 값과 noise[8..0] 열의 값 사이에 오류가 발생했으나, 오류가 정정되어 <그림 5(b)>의 decode.all과 같이 출력되고 있다. 이 값은 <그림 4(b)>의 인코더 입력값 inen.all과 같음을 알 수 있다.

2. Block decoding Viterbi Decoder에의 적용

블록 디코딩은 trace forward 동작을 survivor path



(a)



(b)

그림 4. Radix-4 Viterbi decoder의 시뮬레이션 결과 및 측정결과 1 (a) MAX+plusII 시뮬레이션 결과 (b) Logic analyzer 측정결과

Fig. 4. Simulation and measure results of a radix-4 Viterbi decoder (1) (a) Simulation result by MAX+plusII (b) Measured result by a logic analyzer.

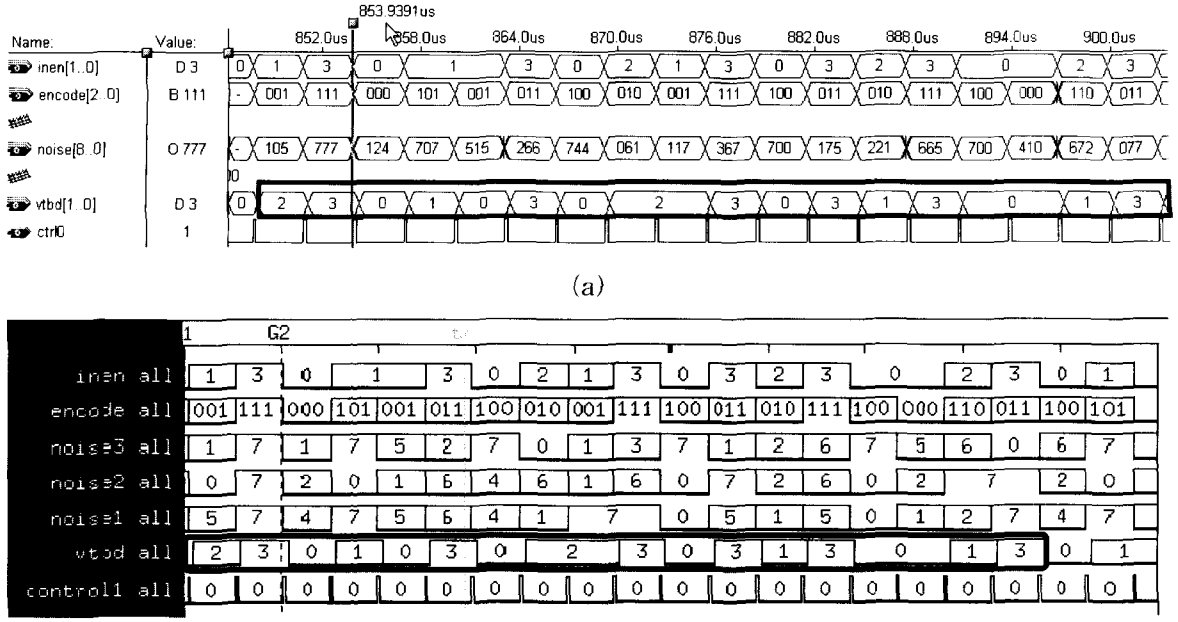


그림 5. Radix-4 Viterbi decoder의 시뮬레이션 결과 및 측정결과 2 (a) MAX+plusII 시뮬레이션 결과 (b) Logic analyzer 측정결과

Fig. 5. Simulation and measure results of a radix-4 Viterbi decoder (2) (a) Simulation result by MAX+plusII (b) Measured result by a logic analyzer.

depth와 디코딩 하고자 하는 디코딩 depth의 합만큼을 진행시킨 후, 생존 경로 길이 만큼 trace-back 시킨다. 이후로는 디코딩 depth만큼 trace-back하여 동일한 수의 데이터를 한번에 얻을 수 있다.[8] 비터비 디코더에서는 생존 경로 길이 만큼의 영역에 대해 trace-forward 동작을 하면, 오류에 의한 영향이 제거된다. 오류정정에 필요한 생존 경로 길이 이상 trace-forward 동작을 진행하면, trace-forward를 진행한 전체 영역에서 생존 경로 길이를 제외한 부분이 디코딩 길이가 되고, 이 영역에서 디코딩값이 오류를 가지고 있을 가능성은 매우 낮아진다. 따라서, 디코딩 depth에 해당하는 생존 경로가 가지고 있는 값은 디코딩이 완료된 값으로 출력된다. 이렇게 디코딩 동작을 시킬 경우, 디코딩 depth만큼 메모리가 추가되고, 한번의 디코딩 동작을 위한 클럭은 증가하지만, 디코딩되어 출력되는 bit수가 증가하므로, 클럭당 디코딩 속도가 증가하게 된다. 예를 들어 생존 경로 길이가 15이고, 디코딩 길이가 15라면, 한번의 디코딩 동작 즉, trace-forward와 trace-back 동작이 모두 끝난 후, 15 길이만큼의 디코딩 결과가 출력되고, 클럭당 디코딩 효율은 1/2에 해당되며, 일반적인 디코딩방식에 비해 7.5배 정도의 이득을 얻게 된다.

여기서 trace-back 동작대신 변형된 레지스터 교환 방식을 적용하여 블록 디코딩을 구현할 수 있다. 즉, 어떤 비터비 디코더이든지 trace-back 동작을 변형된 레지스터 교환 방식으로 구현할 수 있다. 이를 보이기 위해 블록 디코딩 방식을 trace-back 동작을 이용한 방식과 변형된 레지스터 교환 방식을 적용한 방식으로 그 장단점을 비교하였다.

<그림 6>은 두 가지 방식으로 3번의 디코딩 동작을 실행할 경우를 비교하여 필요한 클럭수를 비교해 본 것이다. 각각의 방식이 최적의 동작이 가능하도록 하기 위해 trace-back을 통하여 디코딩 값을 구하는 방식에서는 trace-forward 동작을 통해 최종 생존 경로를 찾은 후, 저장되어 있는 모든 decision bit에 대해 trace-back 동작을 진행한다. 이 과정 중에, 15 stage를 trace-back하면, 다음 디코딩 동작이 시작될 위치에서의 decision 값에 의한 상태가 추출되어, 다음 디코딩 동작을 위한 trace-forward 동작이 실행 가능하다. 따라서, trace-forward 동작이 계속 이루어지면서 동시에 디코딩 값을 출력할 수 있다. 변형된 레지스터 교환 방식을 이용한 블록 디코딩에서는 trace-forward 과정 중에 디코딩 영역(15 clock cycle)에서는 trace-back 디코

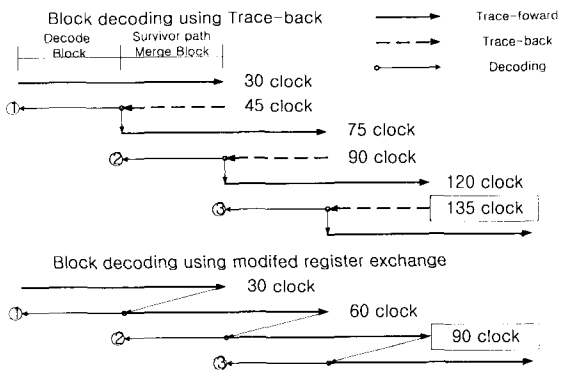


그림 6. trace-back 방법과 변형된 레지스터 교환 방식에 의한 블록 디코딩의 동작 비교

Fig. 6. Comparison of block decoding operation between the trace-back scheme and the register exchange scheme.

딩 방식처럼 decision 값을 저장하고, 디코딩 영역이 끝나고 trace 영역이 시작되는 위치에서는 변형된 레지스터 교환 방식이 적용되어 다음 디코딩 동작을 시작할 시작 상태 값만을 넘겨준다. 따라서 trace-forward 동작이 모두 끝나면, 바로 다음 디코딩 동작을 시작할 수 있으며, 동시에 디코딩 값을 가지고 있는 최종 생존 경로에서의 디코딩 영역으로 이동하여 trace-back 동작을 통한 디코딩 출력을 시작할 수 있다. 두 방식 모두 처음에 입력된 데이터 블록에 대해서는 디코딩 블록과 생존 경로 merge block 만큼 진행한 후 다음 동작을 진행하므로 그 이후에 입력된 데이터에 비해 15 clock 이 더 필요하다. 따라서 처음 들어온 입력과 그 이후에 연속적으로 들어오는 입력은 throughput이 다르다. <표 4>에 나타난 바와 같이 두 가지 방식의 비교 결과, 변형된 레지스터 교환 방식을 이용한 블록 디코딩 방식이 메모리는 1/2로 감소하고, 동작에 필요한 클럭은 최대 50%가 증가하여 더 우수함을 보여 주었다.

Ⅳ. 결 론

본 논문에서는 비터비 디코더의 일반적인 구현 방식인 trace-back 방식을 사용하지 않고, 메모리와 latency에서 이득을 가지는 변형된 레지스터 교환 방식을 제안하였다. 기존의 trace-back 방식과 레지스터 교환 방식, 변형된 레지스터 교환 방식과의 비교를 통하여 변형된 레지스터 교환 방식의 장단점을 알아보았으며, 시뮬레이션을 통해 디코딩 동작을 확인하였다. 제안한 방식은 기존 trace-back 방식에 비해 trace-back 과정이 필요 없어서 2배의 throughput과 매우 작은 메모리 크기(1/5K)로 우수하였고 레지스터 교환 방식에 비해서 1/5K의 메모리 크기와 구조의 복잡도에서 우수함을 보여주었다. 또한 제안된 방식을 이용하여 radix-4 ACS를 갖는 단일 bit 비터비 디코더를, VHDL을 이용하여 설계하고 FPGA(10K70RC 240-4, logic cells 28% 사용)로 구현하였다. 설계한 비터비 디코더는 code rate

표 4. 블록 디코딩 방식으로 구현한 비터비 디코더의 비교

Table 4. Comparison of implementation methods of Viterbi decoders with block decoding scheme.

구현 / 비교항목	Trace-back decoding	Modified Register Exchange	Block decoding (with trace-back)	Block decoding (Modified Register Exchange)
Survivor path memory [bit]	$L \times S \times d$ $= 5K \cdot 2^K \cdot k$ (240/10240)*	$S \cdot k$ $=2^K \cdot k$ (8/256)*	$(L+M) \cdot S \cdot k$ (480/20480)*	$(M \cdot k+K) \cdot S$ (264/12288)*
number of clock to obtain decoded bit(s)	$L + L = 2L = 10K$ (30/80)*	$L = 5K$ (15/40)*	$2(L+M)$ (45/120)*	$L + 2M$ (30/80)*
Throughput (decoded data bits per one clock cycle)	$1/2L$ $(\frac{1}{30}/\frac{1}{80})$*	$1/L$ $(\frac{1}{15}/\frac{1}{40})$*	$M/(2L+2M)$ $(\frac{1}{4}/\frac{1}{4})$*	$M/(L+2M)$ $(\frac{1}{3}/\frac{1}{3})$*
연속적인 data에 대한 throughput (bit/cycle)	$1/2L$ $(\frac{1}{30}/\frac{1}{80})$*	$1/L$ $(\frac{1}{15}/\frac{1}{40})$*	$M/(2L+M)$ $(\frac{1}{3}/\frac{1}{3})$*	$M/(L+M)$ $(\frac{1}{2}/\frac{1}{2})$*

* : (Code rate 2/3, K=3, M=L=15 / Code rate 1/3, K=8, M=L=40)

L : survivor path length(=5×K) , K : Constraint length, S : # of state(= 2K)

d : decision bit, k : decoded data (code rate = k/n), M: size of decoding block

2/3이고 constraint length K가 3인 디코더이며, 측정을 통해 오류 정정 기능과 예상했던 성능을 검증하였다. 제안된 방식은 trace-back을 이용한 모든 디코더에서 이용 가능하다. 이를 보이기 위해 블록 디코딩 방식에 변형된 레지스터 교환 구조를 적용하였고, 이 경우 블록 디코딩 속도를 최대 50% 향상시킴을 확인하였다.

참 고 문 헌

[1] Peter Sweeney, *Error Control Coding an introduction*, Prentice Hall, 1991.

[2] Shu Lin, Daniel J. Costello, Jr, *Error Control Coding : Fundamentals and Applications*, Prentice Hall, 1983.

[3] Feygin, G., Chow, P., Gulak, P.G., Chappel, J., Goodes, G., Hall, O., Sayes, A., Singh, S., Smith, M.B., Wilton. S., "A VLSI implementation of a cascade Viterbi decoder with traceback", Circuits and Systems, 1993., ISCAS '93, 1993 IEEE International Symposium on, May 1993 Page(s): 1945-1948 Vol. 3.

[4] Jens Sparso, Henrik N. Jorgensen, Erik Paaske, Steen Pedersen, and Thomas Rü'bner-Petersen, "An Area-Efficient Topology for VLSI Implementation of Viterbi Decoders and Other Shuffle-Exchange Type Structures", *IEEE J. Solid-State Circuits*, Vol. 26, No. 2, pp. 90~97, FEBRUARY 1991.

[5] Montse Bóo, Francisco Argü'ello, Javier D. Bruguera, Ramón Doallo and Emilio L. Zapata, "High-Performance VLSI Architecture for the Viterbi Algorithm", *IEEE Trans. Communication*, Vol. 45, pp. 168~176, FEBRUARY 1997.

[6] Peter J. Black and Teresa H. Meng, "A 140-Mb/s 32-state, Radix-4 Viterbi Decoder" *IEEE J. Solid-State Circuits*, Vol. 27, pp. 1877-1885, DECEMBER 1992.

[7] K. Hu, W.Lin, and M. Caldwell, "A Viterbi Decoder Memory Management System Using Forward Traceback and All-Path Traceback" *Proceedings of the IEEE 1999 International Conference on Consumer Electronic*, pp. 68~69.

[8] Peter J. Black and Teresa H.-Y. Meng, "A 1-Gb/s, Four-State, Sliding Block Viterbi Decoder", *IEEE J. Solid-State Circuits*, Vol. 32, No. 6, pp. 797~805, JUNE 1997.

[9] Mansoor A. Chirstie, "Viterbi Implementation on the TMS320C5x for V.32 Modems", Digital Signal Processing Applications-Semiconductor Group, Document #SPRA099.pdf, Texas Instruments Incorporated, Texas, 1996.

[10] A. Mark Earnshaw, Steven D. Blostein, "A Combined Soft-Decision Deinterleaver/Decoder for the IS95 Reverse Link", *IEEE Trans. Vehicular Technology*, Vol. 49, No. 2, pp. 448~452, March 2000.

[11] Vijay K. Garg, *IS-95 CDMA and cdma2000 : Cellular/PCS Systems Implementation*, Prentice Hall, 2000.

저 자 소 개

李 燦 豪(正會員)

1987년 2월 서울대학교 전자공학과 졸업 (공학사). 1989년 2월 서울대학교 대학원 전자공학과 졸업 (공학석사). 1994년 6월 University of California, Los Angeles 전자공학과 졸업(공학 박사). 1994년 8월~1995년 2월 삼성전자 반도체연구소 선임연구원. 1995년 3월~현재 숭실대학교 전자공학과 부교수. <주관심분야 : 채널 코덱의 VLSI 구현, 저전력 프로세서 설계, 암호 프로세서 설계, 데이터 통신용 전송 모듈 설계, SOC 설계 방법론 등>

盧 承 孝(正會員)

1998년 2월 호서대학교 전자공학과 졸업 (공학사). 2001년 2월 숭실대학교 대학원 전자공학과 졸업 (공학석사). 2001년 2월 - 현재 삼성전자 시스템 LSI 사업부 LSI 개발 1팀의 스마트 카드 설계팀에 근무중