논문 2007-44SD-7-11

비교 연산을 개선한 SPEC-T 비터비 복호기의 구현

(A SPEC-T Viterbi decoder implementation with reduced-comparison operation)

방 승 화*, 임 종 석**

(Seung-Hwa Bang and Chong-Suck Rim)

요 약

비터비 복호기는 디지털 통신 시스템에서 순방향 오류 정정을 위해서 사용하는 핵심적인 부분으로 최우 추정 복호 방식의 알고리즘을 사용한다. 비터비 복호기는 복호기 상태의 개수만큼의 경로를 계산하고 역 추적하는 특성 때문에 저 전력화가 상당히 어렵다. 본 논문에서는 기존의 SPEC-T 알고리즘을 구현하는데 있어서 비교기의 동작을 최소화할 수 있는 효율적인 방법을 제안하고 ACS(Add-Compare-Select) 구조와 MPMS(Minimum Path Metric Search) 구조에 이를 적용하였다. 실험 결과, 제안한 ACS 구조와 MPMS 구조는 기존의 구조보다 전력 소모량이 임계 값 26에서 각각 최대 약 10.7%와 11.5% 감소하였고 SPEC-T 구조보다는 전력 소모량이 임계 값 26에서 각각 약 6%와 1.5% 더 감소하였다.

Abstract

The Viterbi decoder, which employs the maximum likelihood decoding method, is a critical component in forward error correction for digital communication system. However, lowering power consumption on the Viterbi decoder is a difficult task since the number of paths calculated equals the number of distinctive states of the decoder and the Viterbi decoder utilizes trace-back method. In this paper, we propose a method which minimizes the number of operations performed on the comparator, deployed in the SPEC-T Viterbi decoder implementation. The proposed comparator was applied to the ACSU(Add-Compare-Select Unit) and MPMSU(Minimum Path Metric Search Unit) modules on the decoder. The proposed ACS scheme and MPMS scheme shows reduced power consumption by 10.7% and 11.5% each, compared to the conventional schemes. When compared to the SPEC-T schemes, the proposed ACS and MPMS schemes show 6% and 1.5% less power consumption. In both of the above experiments, the threshold value of 26 was applied.

Keywords: VLSI design, Viterbi decoder, low-power, SPEC-T algorithm

I. 서 론

비터비 복호기는 디지털 통신 시스템에서 순방향 오류 정정(forward error correction)을 위해 사용하는 핵심 부분으로 대략 모바일 모뎀 전력 소모량의 1/3을 차지한다[1]. 일반적으로 ACSU(Add-Compare-Select Unit)은 덧셈기, 비교기, 선택기의 사용으로 복호기의 가장 큰 면적을 차지하며, 복호기 상태 개수만큼의 생존 경로(survivor path)를 찾는 특성은 회로간의 복잡한 연결로 이어져 상당한 전력을 소모한다[2]. 이와 같이 ACSU는 비터비 복호기의 회로 면적과 전력 소모를 결정하는 중요한 부분으로 효율적인 구조가 반드시 필요하다[2~3].

일반적으로 비터비 복호기의 전력 소모량을 최소화하는 방법은 크게 두 가지로 나누어 볼 수 있다. 첫째, 면적을 줄이는 방법으로 ACS(Add-Compare-Select) 연산을 재 정렬 하여 덧셈기의 수를 줄이는 방법[2~3], 비트-시리얼 구조를 이용하여 연산기의 크기를 줄이는 방법[4~5] 등이 있다. 둘째, 제한적인 탐색 알고리즘(limited search algorithm)의 사용으로 실제 계산에 참

* 정회원, (주)인테그마 연구개발팀
(R&D Team, INTEGMA Co., Ltd)
** 평생회원, 서강대학교 컴퓨터공학과
(Department of Computer Science and Engineering, Sogang University)
접수일자: 2007년4월6일 수정완료일: 2007년6월21일

여하는 연산의 횟수를 줄이는 방법으로 각 복호 단계마다 생존 경로의 개수를 M개 이하로 고정 시키는 M-알고리즘[6], 각 복호 단계마다 임계 조건(threshold condition)을 이용하여 동적으로 생존 경로의 개수를 유지하는 T-알고리즘(Threshold algorithm)[7]이 있다. T-알고리즘의 전력 소모량에 관한 효율성은 입증[8] 되었지만 매 복호 단계마다 생존 경로를 선택하기 위한 비교식에서 최소 경로 값(minimum path metric)을 찾기 위한 지연 시간이 필요하고, 선택한 경로 값(path metric)의 불규칙성은 하드웨어 구현을 까다롭게 한다.

이러한 T-알고리즘의 문제점을 개선한 SPEC-T 알고리즘(Speculation-T algorithm)이 최근 제안되었다[9]. SPEC-T 알고리즘은 매 복호 단계마다 예측한 최소 경로 값을 사용하여 최소 경로 값을 찾기 위한 지연 시간을 없애고, 가지 값(branch metric)을 정규화하고 생존 경로의 경로 값을 음수로 만들어 ACS 연산에서 덧셈 연산의 횟수를 줄이는 방법이다. 반면에 덧셈 연산 다음으로 빈번히 수행하는 비교 연산의 횟수를 줄이지 못하고 있다.

이에 본 논문에서는 SPEC-T 알고리즘을 구현하는데 있어서 전력 소모량을 더욱 줄이기 위해서 비교 연산의 횟수를 최소화할 수 있는 방법을 제안하고 이의 구조를 보인다. 제안한 비교기를 적용하여 구속장(constraint length) K가 7이고 8단계 연판정(soft decision) 입력에 동작하는 개선된 SPEC-T 구조의 비터비 복호기를 구현하였다.

본 논문의 구성은 다음과 같다. Ⅱ장에서 비터비 알고리즘과 관련된 이론을 소개한다. Ⅲ장에서는 SPEC-T 알고리즘을 구현하는데 있어서 비교기 사용을 최소화하는 방법과 구조를 제시하고 전력 소모량을 개선한 SPEC-T 비터비 복호기의 구조를 보인다. Ⅳ장에서는 실험 결과를 통해 제안한 구조와 기존의 구조들 사이에 전력 소모량을 비교해 본다. 마지막으로 Ⅴ장에서는 본 논문의 결론을 내린다.

Ⅱ. 기본 정리

1. 비터비 알고리즘

비터비 알고리즘(Viterbi algorithm)은 최우 추정 방식(maximum likelihood)의 알고리즘으로 길쌈 부호(convolutional code)를 복호하기 위해 1967년에 Viterbi가 처음으로 제안 하였다[10]. 일반적으로 비터비 알고리즘의 VLSI 구현은 그림 2.1과 같이 크게 세 가지 부분

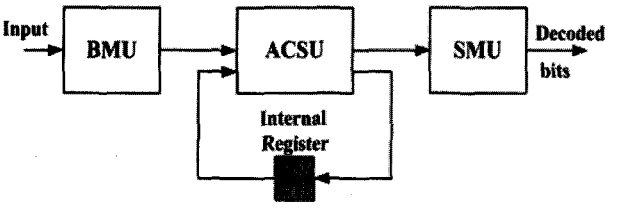


그림 2.1. 비터비 복호기의 기본 블록도
Fig. 2.1. Basic block diagram of the Viterbi decoder.

으로 나누어진다. BMU(Branch Metric Unit)은 각 입력 부호어(code-word)에 대한 가지 값을 계산한다. ACSU은 각 상태의 모든 경로 값의 갱신과 생존 경로를 결정한다. SMU(Survivor Management Unit)는 ACSU에서 계산한 결정 비트(decision bit)를 이용하여 생존 경로를 따라 존재하는 복호 비트를 찾는다. 일반적으로 복호 비트를 찾는 방법은 역추적(trace-back: TB) 방법과 레지스터 교환(register-exchange: RE) 방법이 있다[11].

비터비 알고리즘은 매 복호 단계에서 각 상태마다 생존 경로를 찾기 위해 식 (2.1)과 같은 ACS 연산을 수행한다.

$$PM_t^k = min\left\{PM_{t-1}^i + BM_t^{i,k}, PM_{t-1}^j + BM_t^{j,k}\right\} \qquad (2.1)$$

식 (2.1)과 같이 시간 t의 상태 k의 경로 값 PM_t^k는 시간 t-1의 상태 i와 j의 각각의 경로 값 PM_{t-1}^i와 PM_{t-1}^j에 상태 i와 j에서 상태 k로의 천이가 갖는 각각의 가지 값 $BM_t^{i,k}$와 $BM_t^{j,k}$을 각각 더하고 최소 경로 값을 갖는 경로를 선택하여 얻는다. 비터비 알고리즘은 이와 같은 ACS 연산을 모든 입력에 대하여 매번 격자도(trellis)가 갖는 상태의 개수만큼 수행한다.

2. T-알고리즘

T-알고리즘은 격자도 상의 각 단계에서 모든 생존 경로를 구하고 임계 조건을 이용하여 유사성이 낮은 생존 경로를 제거하는 너비 우선 탐색 복호(breadth first

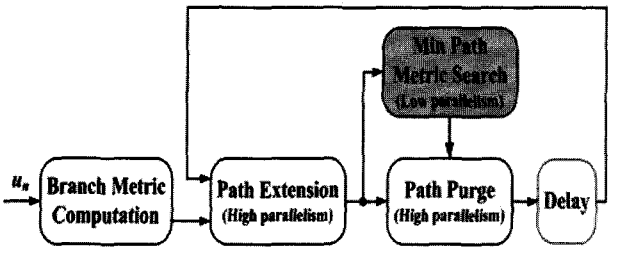


그림 2.3. T-알고리즘 데이터 흐름도
Fig. 2.3. Data-flow diagram of the T-algorithm.

search decoding) 계열의 알고리즘이다[7]. T-알고리즘은 매 복호 단계마다 최소 경로 값과 임계값을 더한 임계 조건과 모든 후보 경로의 경로 값을 비교하여 임계 조건보다 크거나 같은 후보 경로의 경로 값을 제거한다. 즉, 복호하는 동안 생존 경로의 평균 개수를 줄여 실제 계산에 참여하는 ACS 연산의 횟수가 줄어들게 한다. 이와 같이 임계 조건을 이용하여 생존 경로의 평균 개수를 줄이는 비교 연산은 식 (2.2)과 같다.

$$PM_i^{(n)} < PM_m^{(n)} + T, \quad (T: threshold) \tag{2.2}$$

식 (2.2)에서 $PM_i^{(n)}$는 n번째 단계의 모든 후보 경로의 경로 값이고, $PM_m^{(n)}$는 $PM_i^{(n)}$중 최소의 값이다. 임계 값 T는 미리 정해진 값이다. 각 복호 단계에서 식 (2.2)는 그림 2.3 T-알고리즘의 데이터 흐름을 이용하여 다음과 같이 설명할 수 있다.

1) 가지 값 계산과 후보 경로 값 계산(Branch Metric Computation and Path Extension)

   임의의 n번째 복호 단계에서 입력 u_n에 대한 모든 가지 값 $BM_j^{(n)}$을 계산하고 (n-1)번째 단계의 모든 생존 경로의 경로 값 $PM_i^{(n-1)}$과 더하여 n번째 단계의 모든 후보 경로의 경로 값 $PM_i^{(n)}$을 계산한다.

2) 최소 경로 값 탐색(Minimum Path Metric Search)

   임의의 n번째 복호 단계에서 후보 경로의 경로 값 $PM_i^{(n)}$중 최소 경로 값 $PM_m^{(n)}$을 찾는다. 이 과정은 상태의 개수보다 하나 적은 비교 연산이 필요하므로 상당한 지연 시간이 필요하다.

3) 경로 제거(Path Purge)

   임의의 n번째 복호 단계에서 식 (2.2)을 만족하지 않는 경로를 제거한다. 또한 선택한 생존 경로의 개수가 $N_{\max}$을 초과할 경우에 임계 값 T를 줄이고 처리 과정을 반복하여 매 복호 단계마다 생존 경로의 개수를 $N_{\max}$이하로 유지한다[7]. $N_{\max}$는 미리 정의된 값이다.

T-알고리즘은 격자도상에서 한 단계의 전체 상태 개수보다 적은 평균 생존 경로를 가지면서 ML 복호 성능에 근접한 알고리즘이다[7~8]. 반면에 T-알고리즘과 같이 탐색을 제한하는 알고리즘의 VLSI 구현을 위해서는 다음과 같이 존재하는 문제를 해결해야 한다[9].

1) 알고리즘 고유의 병목현상(bottleneck)

   그림 2.3에서 보듯이 생존 경로의 계산과 제거는 약간의 덧셈과 비교만으로 매 단계마다 병렬 처리가 가능하지만 최소 경로 값을 찾는 연산은 시리얼(serial) 요소가 존재하기 때문에 상대적으로 큰 지연 시간을 초래한다.

2) 불규칙한 데이터 읽기/쓰기

   T-알고리즘은 생존 경로의 상태와 생존 경로의 개수가 각 복호 단계에서 불규칙하게 나타난다. 이는 병렬 처리를 방해하는 요소이며 성능 감소와 전력 소모의 증가로 이어진다.

다음 절에서는 이와 같이 언급한 T-알고리즘의 문제점을 해결한 SPEC-T 알고리즘을 설명할 것이다.

3. SPEC-T 알고리즘

SPEC-T 알고리즘은 가장 큰 지연 시간을 초래하는 최소 경로 값을 찾는 연산을 T-알고리즘의 주 데이터 흐름에서 제거한 방법으로 2005년 Sun이 제안하였다[9]. SPEC-T 알고리즘은 매 복호 단계마다 최소 경로 값이 최소 가지 값을 갖는 가지(branch)를 따라서 존재한다고 예상하여 실제 최소 경로 값을 사용하지 않고 예측 최소 경로 값을 사용하여 임계 조건 비교식을 만든다. 최소 경로 값을 예측하는 방법은 식 (2.3)을 수행하여 얻을 수 있다.

$$\Gamma_m^{(n)} = \begin{cases} \Gamma_m^{(n-1)} + BM_m^{(n)}, & if\ n \bmod v \neq 0 \\ \Gamma_m^{(n-1)} + BM_m^{(n)} + E, & if\ n \bmod v = 0 \end{cases} \tag{2.3}$$
$$E = min\left\{PM_i^{(n-v)} - \Gamma_m^{(n-v)}\right\}$$

식 (2.3)에서 $\Gamma_m^{(n)}$는 n번째 단계의 예측 최소 경로 값이고 $BM_m^{(n)}$는 n번째 단계의 실제 최소 가지 값이다. E는 (n-v)번째 단계의 생존 경로의 실제 최소 경로 값 $PM_m^{(n-v)}$와 예측 최소 경로 값 $\Gamma_m^{(n-v)}$간에 차이다. v는 n번째 단계의 예측 최소 경로 값 $\Gamma_m^{(n)}$의 오차를 보상해주기 위한 단계이다.

식 (2.3)에서 $n \bmod v \neq 0$인 경우는 (n-1)번째 단계의 예측 최소 경로 값 $\Gamma_m^{(n-1)}$에 n번째 단계의 최소 가지 값 $BM_m^{(n)}$을 더하여 n번째 단계의 예측 최소 경로 값 $\Gamma_m^{(n)}$을 구한다. $n \bmod v = 0$의 경우는 예측에 따른 오차를 보상해 주기 위해서 매 v단계 마다 E를 추가로 더하여 n번째 단계의 예측 최소 경로 값 $\Gamma_m^{(n)}$을 구한

다. 실제 최소 경로 값을 찾을 수 있는 지연 시간 v마다 오차를 보완해 주는 것은 $(n-v)$단계에서 예측 최소 경로 값이 아닌 실제 최소 경로 값을 사용하는 것을 의미한다. 즉, 예측에 필요한 단계는 항상 일정한 간격 v를 유지하므로 오차 또한 일정한 값 이상을 벗어나지 않아 오차 범위를 최소화 할 수 있다.

Ⅲ. 저 전력을 위한 비터비 복호기의 구조

1. SPEC-T구조의 비교 연산 최소화

SPEC-T ACS 연산에서 비교 연산은 덧셈 연산 다음으로 빈번히 수행하는 연산이다. 또한 실제 최소 경로 값을 찾기 위한 MPMSU(Minimum Path Metric Search Unit)는 복호기 상태의 개수보다 한 개 적은 비교 연산을 수행한다. 즉, 비터비 복호기의 비교 연산은 덧셈 연산 다음으로 가장 빈번하게 수행하는 연산중의 하나로 상당한 전력을 소모하지만 SPEC-T 구조의 복호기는 비교 연산의 동작과 정지는 고려하지 않고 있다. 이에 본 논문에서는 SPEC-T 구조의 비터비 복호기의 비교 연산을 최소화하는 방법을 제안한다.

SPEC-T ACS 연산에서 n번째 단계의 상태 p의 경로 값 $PM_p^{(n)}$을 갱신하기 위한 비교 연산은 식(3.1)과 같다.

$$PM_i^{(n-1)}+(BM_{i,p}^{(n)}-\Gamma_m^{(n)})<PM_j^{(n-1)}+(BM_{j,p}^{(n)}-\Gamma_m^{(n)}).$$
$$\Gamma_m^{(n)}=\begin{cases}BM_m^{(n)}, & \text{if } n \bmod v \neq 0.\\ BM_m^{(n)}+PM_m^{(n-v)}+T, & \text{if } n \bmod v = 0.\end{cases} \quad (3.1)$$

식 (3.1)에서 보듯이 두 입력 경로 값 $PM_i^{(n-1)}$와 $PM_j^{(n-1)}$는 각각 생존 경로일 경우 음수이므로 MSB가 1이고 생존 경로가 아닐 경우는 0이므로 MSB는 0이다. 이 특성을 이용하여 입력 경로 값의 MSB가 1일 경우는 각각의 경로 값과 가지 값을 선택하여 덧셈 연산을 수행하고 입력 경로 값의 MSB가 0일 경우는 가지 값에 0을 입력하고 경로 값 또한 0이므로 덧셈 연산을 수행하지 않는 효과를 볼 수 있다.

표 3.1. 각 구조에서 비교 연산의 동작과 정지
Table 3.1. Compare operation of different architectures.

MSBs	비교 연산의 동작과 정지		
	기존의 방법	SPEC-T 방법	제안한 방법
00	동작(2)	정지(2)	정지(2)
01	동작(2)	동작(2)	정지(2)
10	동작(2)	동작(2)	정지(2)
11	동작(2)	동작(2)	동작(2)
Total	동작(8)/정지(0)	동작(6)/정지(2)	동작(2)/정지(6)

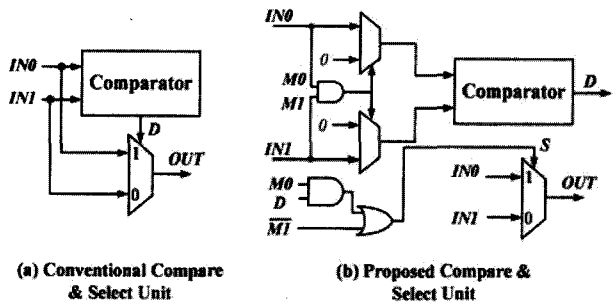


그림 3.1. 기존의 비교기 구조와 제안한 비교기 구조
Fig. 3.1. Conventional CS(Compare&Select) architecture and proposed CS architecture.

이때 덧셈 연산을 수행한 결과인 두 후보 경로의 경로 값 중 적어도 하나가 0이라면 0이 아닌 쪽을 선택하면 되므로 비교 연산을 수행하지 않고도 새로운 경로 값을 선택할 수 있다. 표 3.1과 같이 비교 연산의 두 입력 경로 값의 MSB가 00, 01, 10, 11인 경우로 나누어 보자. 기존의 방법은 모든 경우에 비교 연산을 수행하고 SPEC-T 방법은 두 입력 경로 값의 MSB가 00일 때를 제외하고는 모두 비교 연산을 수행한다. 반면에 제안한 방법은 적어도 하나의 입력 경로 값의 MSB가 0이면 비교 연산을 수행하지 않고도 두 값의 크기를 비교할 수 있으므로 연산의 횟수를 상당히 줄일 수 있다.

그림 3.1은 기존의 비교기 구조와 제안한 비교기 구조를 보이고 있다. 그림 3.1 (b)을 보자. 생존 경로의 경로 값을 음수로 만드는 특성에 따라서 비교기의 두 입력 *IN0*과 *IN1*은 음수 아니면 0이다. 이에 두 입력의 MSB *M0*과 *M1*은 0 또는 1이다. 입력 단의 두 0-선택기는 *IN0*과 *IN1*이 모두 음수일 경우는 *IN0*과 *IN1*을 선택하여 비교 연산을 수행한다. 반면에 적어도 하나의 입력이 0일 경우는 두 입력의 MSB *M0*과 *M1*을 AND한 결과는 0이므로 두 0-선택기에서 모두 0을 선택한다. 이는 비교기의 두 입력으로 0을 입력하여 비교기의 동작을 멈출 수 있는 효과를 볼 수 있다.

표 3.2 기존의 비교기 구조의 진리표
Table 3.2 Truth table of conventional CS architecture.

Input		D	OUT
IN0	IN1		
0	0	0	×
0	음수	0	IN1
음수	0	1	IN0
음수	음수	0	IN1
음수	음수	1	IN0

표 3.3 제안한 비교기 구조의 진리표
Table 3.3 Truth table of proposed CS architecture.

Input					S	OUT
IN0	IN1	M0	M1	D		
0	0	0	0	0	1	×
0	음수	0	1	0	0	IN1
음수	0	1	0	0	1	IN0
음수	음수	1	1	0	0	IN1
음수	음수	1	1	1	1	IN0

그림 3.1 (a)에서 보듯이 기존의 구조는 비교 결과를 이용하여 선택기에서 최소의 값을 선택하지만 제안한 구조에서는 비교기의 두 입력 중 적어도 하나의 입력이 0인 경우 비교 연산을 수행하지 않으므로 최소의 값을 선택하기 위한 추가적인 회로가 필요하다.

그림 3.1 (b)을 보면 입력 *IN0*의 MSB *M0*과 비교기의 비교 결과인 *D*를 AND한다. 이 결과는 또 다른 입력 *IN1*의 MSB *M1*을 NOT한 결과와 OR한다. 이 결과는 출력 선택기에서 적절한 값을 선택하여 기존의 구조와 동일한 결과를 얻을 수 있다. 표 3.2와 표 3.3은 각각 기존의 비교기 구조와 제안한 비교기 구조의 진리표(truth table)를 보이고 있다.

비교 연산의 두 입력 값의 MSB가 가질 수 있는 4가지 값의 비율이 서로 모두 일정하다고 가정하자. 기존의 방법의 비교 연산을 100%로 보았을 때 SPEC-T 방법은 기존의 방법보다 최대 25%의 비교 연산을 줄일 수 있다. 반면에 제안한 방법은 기존의 방법보다 최대 75%의 비교 연산을 줄일 수 있고 SPEC-T 방법보다 최대 50%의 비교 연산을 더 줄일 수 있다.

이와 같이 제안한 비교기 구조는 기존의 비교기 구조에 AND 2개와 OR 1개 그리고 선택기 2개를 추가하여 약간의 면적 증가는 있으나 비교기의 동작을 최소화할 수 있는 좀 더 효율적인 구조이다.

2. SPEC-T 구조를 개선한 비터비 복호기의 구조

가. 구현한 복호기의 상위 블록도

구현한 비터비 복호기는 3비트로 구성된 부호어 2개를 입력으로 받고 복호된 데이터를 1비트로 출력한다. 이에 대한 비터비 복호기의 상위 블록도를 그림 3.2에 보인다. 구현한 비터비 복호기의 동작을 위한 입력 신호는 회로의 동기화를 위한 *Clk*, 회로의 초기화를 위한 *Reset*, 비터비 복호기의 동작 신호를 알리는 *Start*로 구성되어 있다. 데이터 입력 신호는 2개의 3비트 부호어 *CW0*과 *CW1*이 있다. 출력 신호는 복호된 1-비트 (Decoded bit)이다.

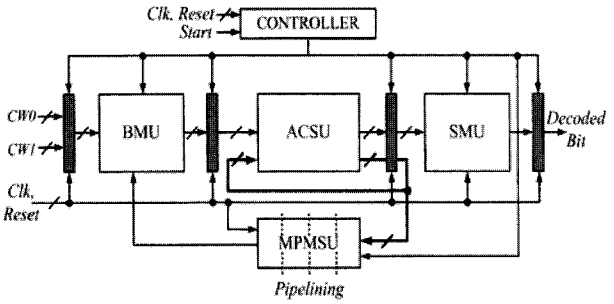


그림 3.2. 구현한 비터비 복호기의 상위 블록도
Fig. 3.2. Top-level block diagram of Viterbi decoder.

그림 3.2에서 보듯이 구현한 비터비 복호기는 총 4개의 모듈로 이루어져 있다. BMU는 두 입력 부호어 *CW0*과 *CW1*을 이용하여 계산한 가지 값과 예측 최소 경로 값을 이용하여 가지 값을 정규화 한다. ACSU는 현재 단계의 경로 값을 갱신하기 위해 이전 단계의 경로 값과 정규화한 가지 값을 이용한다. SMU는 생존 경로를 역추적하기 위하여 ACSU로부터 전달 받은 생존 경로 결정 비트를 이용한다. 역추적 구조는 write, search, decode 연산으로 이루어져 있다. 세 연산은 동시에 수행하고 역추적 생존 경로의 길이는 48이다. MPMSU는 예측에 따른 오차를 보완해 주는 단계 v마다 최소 경로 값을 BMU에 전달한다.

나. 정규화한 가지 값을 계산하기 위한 BMU

식 (3.1)과 같이 ACS 연산을 하기 전에 정규화한 가지 값을 미리 계산하는 구조를 그림 3.3에 보이고 있다.

그림 3.3 (a)은 예측 최소 경로 값을 찾기 위한 구조이다. 두 입력 부호어는 거리를 나타내는 값이므로 본래의 값과 1의 보수를 취한 값 중 MSB가 0인 쪽이 작은 값이다. 그러므로 $CW0$과 $\overline{CW0}$ 사이에 MSB가 0인 값을 선택하고 $CW1$과 $\overline{CW1}$ 사이에 MSB가 0인

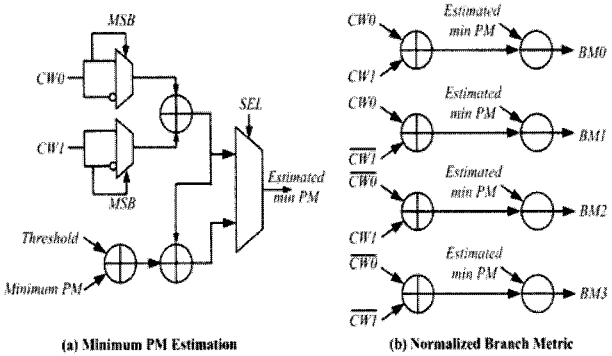


그림 3.3. 정규화한 가지 값을 계산하기 위한 BMU
Fig. 3.3. Normalized Branch Metric Unit(BMU).

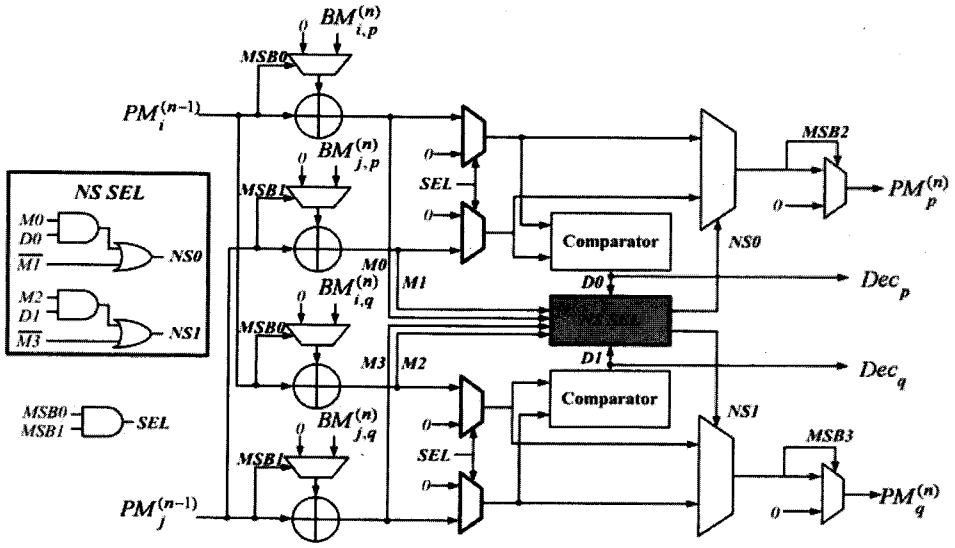


그림 3.4. 제안한 SPEC-T Radix-2 Butterfly ACS 구조
Fig. 3.4. Proposed SPEC-T Radix-2 Butterfly ACS architecture.

값을 선택하여 두 값을 더하면 최소 가지 값을 구할 수 있다. 임계값은 미리 정해진 값이고 최소 경로 값은 MPMSU로부터 입력 받는다. 출력 선택기에서 *SEL*신호는 예측에 따른 오류를 보완해주는 단계 v마다 1로 바뀌고 그 외는 0이 된다.

그림 3.3 (b)은 가지 값을 정규화하기 위한 구조를 보이고 있다. 첫째 단의 덧셈기는 일반적인 가지 값을 계산하고 둘째 단의 뺄셈기는 일반적인 가지 값과 예측 최소 경로 값을 입력으로 정규화한 가지 값을 계산한다. 일반적인 BMU에 비해서 덧셈기 3개, 선택기 3개, 뺄셈기 4개를 추가로 사용하지만 면적과 전력 소모 증가는 극히 미미한 수준이다.

다. 개선된 SPEC-T Radix-2 Butterfly ACS 구조

본 논문에서는 기억 장치 접근 횟수를 줄이기 위해 참고 문헌 [9]에서 제안한 SPEC-T ACS 연산을 기존의 Radix-2 Butterfly ACS 연산에 적용하고 이를 SPEC-T Radix-2 Butterfly ACS 연산이라 한다.

SPEC-T Radix-2 Butterfly ACS 연산에서 경로 값 $PM_p^{(n)}$와 $PM_q^{(n)}$을 갱신하기 위한 비교 연산 식은 식 (3.2)과 같다.

$$\begin{aligned} &PM_i^{(n-1)}+(BM_{i,p}^{(n)}-\Gamma_m^{(n)})<PM_j^{(n-1)}+(BM_{j,p}^{(n)}-\Gamma_m^{(n)}),\\ &PM_i^{(n-1)}+(BM_{i,q}^{(n)}-\Gamma_m^{(n)})<PM_j^{(n-1)}+(BM_{j,q}^{(n)}-\Gamma_m^{(n)}).\\ &\Gamma_m^{(n)}=\begin{cases} BM_m^{(n)}, & \text{if } n \bmod v \neq 0.\\ BM_m^{(n)}+PM_m^{(n-v)}+T, & \text{if } n \bmod v = 0. \end{cases} \end{aligned} \quad (3.2)$$

식 (3.2)에서 보듯이 SPEC-T Radix-2 Butterfly ACS 연산은 $(n-1)$번째 단계의 모든 생존 경로의 경로 값과 괄호 부분의 정규화한 가지 값을 더해 각 상태의 후보 경로의 경로 값을 구하고 서로 비교하여 n번째 단계의 생존 경로의 경로 값을 구한다. 이 때 일반적인 ACS 연산과 달리 $(n-1)$번째 생존 경로의 경로 값과 정규화한 가지 값을 더한 결과를 n번째 단계의 후보 경로의 경로 값으로 사용한다. 이것은 $(n-1)$번째 단계의 예측 최소 경로 값 $\Gamma_m^{(n-1)}$이 n번째 생존 경로의 경로 값에 누적되어 있다는 것을 의미한다. 이로 인해 n번째 단계에서 예측 최소의 경로 값 $\Gamma_m^{(n)}$은 $n \bmod v = 0$일 때는 n번째 단계의 최소 가지 값 $BM_m^{(n)}$이고 예측에 따른 오류를 보완해 주는 $n \bmod v \neq 0$일 때는 n번째 단계의 최소 가지 값 $BM_m^{(n)}$와 $(n-v)$번째 실제 최소 경로 값 $PM_m^{(n-v)}$과 임계 값 T를 더한 것이다.

그림 3.4는 SPEC-T Radix-2 Butterfly ACS 구조에

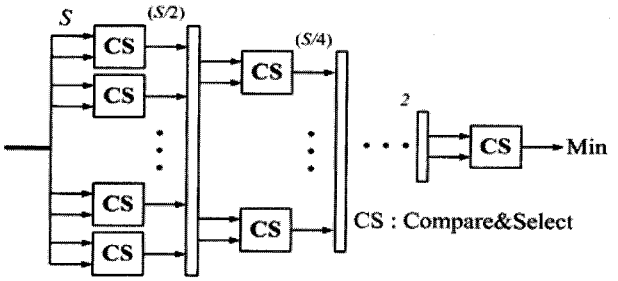


그림 3.5. S개의 입력을 갖는 비교기 트리
Fig. 3.5. S-input comparator tree.

제안한 비교기를 적용한 구조이다. 그림 3.4에서 보듯이 두 입력 경로 값 $PM_i^{(n-1)}$와 $PM_j^{(n-1)}$은 모두 생존 경로가 아닐 경우 0이므로 MSB는 0이 되고 가지 값 0-선택기도 0을 선택하여 덧셈기가 동작하지 않는다. 이 두 입력 경로 값이 모두 생존 경로일 경우에는 음수이므로 MSB도 1이 되고 가지 값 0-선택기도 가지 값을 선택하여 덧셈기가 동작한다. 반면에 적어도 하나의 입력 경로 값이 0이면 butterfly 구조의 특성에 따라 4개의 덧셈기 중 2개만 동작하고 2개는 정지한다. 이 때 비교기의 입력은 덧셈기의 출력을 사용하므로 두 개의 비교기 입력을 선택하는 4개의 0-선택기는 모두 0을 선택하여 비교기는 모두 동작하지 않는다.

*NS0*과 *NS1*은 상태 p와 q의 후보 경로 중 생존 경로의 경로 값을 선택하기 위해서 그림 3.4의 *NS SEL*과 같은 블록 회로를 이용하여 만든다. *NS0*과 *NS1*에 의해 각각 선택된 상태 p와 q의 생존 경로의 경로 값은 음수가 아닐 경우에 MSB는 0이므로 마지막 0-선택기에서 0을 선택하여 생존 경로를 제거한다. Dec_p와 Dec_q는 생존 경로의 결정 비트로 SMU에서 생존 경로를 복호하기 위해서 사용한다.

라. 제안한 비교기를 적용한 MPMSU

MPMSU은 최소 경로 값을 찾기 위해서 사용하는 유닛(unit)이다. 그림 3.5에서 보듯이 이 유닛은 복호기의 상태의 개수가 S일 때 S-1개의 비교기와 선택기를 각각 사용하고 최대 $\log_2 S$개의 파이프라인 단계로 나누어진다. 이에 따라 비터비 복호기에서 ACSU 다음으로 비교 연산을 가장 빈번하게 수행하므로 본 논문에서 제안한 비교기 구조를 이 유닛에 적용하여 상당한 전력 소모량을 줄일 수 있다. 간단히 CS 블록을 제안한 비교기로 대신하면 된다.

표 4.1. 복호하는 동안 각 ACSU의 비교기 사용률
Table 4.1. Comparator operation-rate of different ACSUs.

		복호하는 동안 비교기 사용률			
		2.5dB	3.0dB	3.5dB	4.0dB
Conventional ACSU		100%	100%	100%	100%
SPEC-T ACSU	T=29	90.59%	88.56%	86.46%	84.23%
	T=28	86.53%	83.98%	81.42%	78.74%
	T=27	82.06%	79.12%	76.27%	73.38%
	T=26	76.92%	73.65%	70.53%	67.39%
Proposed ACSU	T=29	72.11%	67.52%	63.18%	58.94%
	T=28	65.07%	60.14%	55.63%	51.29%
	T=27	58.14%	53.20%	48.83%	44.74%
	T=26	51.13%	46.32%	42.14%	38.29%

표 4.2. 복호하는 동안 각 MPMSU의 비교기 사용률
Table 4.2. Comparator operation-rate of different MPMSUs.

		복호하는 동안 비교기 사용률			
		2.5dB	3.0dB	3.5dB	4.0dB
Conventional MPMSU		100%	100%	100%	100%
SPEC-T MPMSU	T=29	90.5%	88.9%	85.7%	84.1%
	T=28	85.7%	84.1%	82.5%	79.4%
	T=27	82.5%	81.0%	77.8%	76.2%
	T=26	79.4%	76.2%	73.0%	71.4%
Proposed MPMSU	T=29	61.9%	57.1%	52.4%	49.2%
	T=28	55.6%	50.8%	46.0%	42.9%
	T=27	49.2%	44.4%	39.7%	36.5%
	T=26	42.9%	38.1%	34.9%	31.7%

Ⅳ. 실험 결과

본 논문에서는 기존의 ACS 구조, SPEC-T ACS 구조, 제안한 ACS 구조를 갖는 각 복호기를 VHDL로 구현하였고 Xilinx ISE 8.2i를 이용하여 합성하고 P&R 하였다. 입력 패턴을 이용한 Post-P&R 시뮬레이션을 통해 XPower를 이용하여 전력 소모량을 측정하였다.

1. 시뮬레이션을 통한 비교기 사용률 비교

표 4.1과 표 4.2는 각 구조의 ACSU와 MPMSU에서 동일한 비교기가 2단계 이상 동작하지 않는 경우를 제외한 비교기 사용률의 C-시뮬레이션 결과를 보이고 있다. 표 4.1과 표 4.2에서 보듯이 기존의 구조는 모든 경우에 100%의 비교기 사용률을 보이고 있다. SPEC-T 구조와 제안한 구조는 임계 값 T가 낮을수록 제거되는 경로가 많아지고 SNR(signal-to-noise ratio)이 클수록

표 4.3. 각 ACSU의 전력 소모량 측정 비교
Table 4.3. Power estimation results of different ACSUs.

		Power(mW)			
		2.5dB	3.0dB	3.5dB	4.0dB
Conventional ACSU		108.20 (100%)	108.49 (100%)	108.24 (100%)	108.12 (100%)
SPEC-T ACSU	T=29	110.41 (+2.00%)	110.45 (+1.77%)	111.12 (+2.59%)	110.02 (+1.73%)
	T=28	109.08 (+0.81%)	108.75 (+0.24%)	108.72 (+0.44%)	109.21 (-1.00%)
	T=27	108.09 (-0.10%)	106.42 (-1.95%)	105.89 (-2.22%)	104.54 (-3.42%)
	T=26	106.35 (-1.74%)	104.64 (-3.68%)	103.10 (-4.99%)	103.25 (-4.72%)
Proposed ACSU	T=29	107.90 (-0.28%)	107.42 (-1.00%)	106.52 (-1.61%)	107.21 (-0.85%)
	T=28	107.52 (-0.63%)	106.64 (-1.73%)	104.21 (-3.87%)	103.43 (-4.53%)
	T=27	105.86 (-2.21%)	103.46 (-4.86%)	101.50 (-6.64%)	100.20 (-7.90%)
	T=26	102.46 (-5.60%)	100.57 (-7.88%)	99.42 (-8.87%)	97.67 (-10.70%)

표 4.4. 각 MPMSU의 전력 소모량 측정 비교
Table 4.4. Power estimation results of different MPMSUs.

		Power(mW)			
		2.5dB	3.0dB	3.5dB	4.0dB
Conventional MPMSU		25.21 (100%)	23.98 (100%)	24.42 (100%)	24.16 (100%)
SPEC-T MPMSU	T=29	24.68 (-2.15%)	23.87 (-0.46%)	23.42 (-4.27%)	23.34 (-3.51%)
	T=28	24.45 (-3.11%)	23.51 (-2.00%)	23.12 (-5.62%)	22.89 (-5.55%)
	T=27	24.28 (-3.83%)	23.04 (-4.08%)	22.77 (-7.25%)	22.52 (-7.28%)
	T=26	23.96 (-5.22%)	22.94 (-4.53%)	22.47 (-8.68%)	21.97 (-9.97%)
Proposed MPMSU	T=29	24.46 (-3.07%)	23.76 (-0.93%)	23.11 (-5.67%)	22.97 (-5.18%)
	T=28	24.32 (-3.66%)	23.37 (-2.61%)	22.79 (-7.15%)	22.54 (-7.19%)
	T=27	24.21 (-4.13%)	22.92 (-4.62%)	22.42 (-8.92%)	22.04 (-9.62%)
	T=26	23.47 (-7.41%)	22.21 (-7.97%)	21.94 (-11.30%)	21.67 (-11.49%)

잡음(noise)이 적으므로 비교기 사용률이 낮아지는 추세를 보이고 있다. 특히, T=26, 4.0dB인 경우에 제안한 ACSU와 MPMSU는 비교기 사용률이 각각 약 38%와 32%로 기존의 구조에 비해서 각각 약 62%와 68% 감소하였고 SPEC-T 구조에 비해서는 각각 약 29%와 40% 더 감소하였다.

2. 전력 소모량 측정 비교

표 4.3과 표 4.4는 각각 XPower를 이용하여 세 구조의 ACSU와 MPMSU의 전력 소모량을 측정한 결과를 보이고 있다.

비교기의 동작을 제어하기 위한 회로는 추가적인 전력을 소모하고, 동일한 비교기가 정지하는 단계가 짧은 경우가 많으면 전력 소모량이 크게 감소하지 않으므로 비교기의 사용률에 비해서는 상대적으로 전력 소모량이 크게 감소하지 않는 결과를 보이고 있다.

표 4.3과 표 4.4에서 보듯이 제안한 비교기를 적용한 ACSU와 MPMSU는 임계 값 T가 작을수록, SNR이 클수록 기존의 구조와 SPEC-T 구조보다 전력 소모량이 감소한 결과를 보이고 있다. 특히, T=26, 4.0dB인 경우에 제안한 ACSU와 MPMSU는 기존의 구조에 비해서 전력 소모량이 각각 약 10.7%와 11.5% 감소하였고 SPEC-T 구조에 비해서는 전력 소모량이 각각 약 6%와 1.5% 더 감소한 결과를 보이고 있다. 이는 제안한 비교기를 적용한 SPEC-T 구조의 비터비 복호기의 전력 효율성을 입증하고 있다.

V. 결 론

본 논문에서는 SPEC-T 알고리즘을 구현하는데 있어서 전력 소모량을 더욱 줄이기 위해 비교기의 동작을 최소화할 수 있는 방법과 구조를 제안하고 이를 ACSU와 MPMSU에 적용하였다. 제안한 비교기는 약간의 면적 증가는 있으나 극히 미미한 수준이다. 실험을 통해 임계값 26에서 제안한 비교기를 적용한 ACSU와 MPMSU는 기존의 구조에 비해서 전력 소모량이 최대 각각 약 10.7%와 11.5% 감소한 것을 확인 하였고 SPEC-T 구조에 비해서는 각각 6%와 1.5% 더 감소한 것을 확인 하였다. 이와 같이 제안한 비교기를 적용한 ACSU와 MPMSU는 실험을 통해 비교한 구조 중 전력 소모량이 상대적으로 가장 적은 구조임을 확인할 수 있었다.

참 고 문 헌

[1] I. Kang and A. N. Willson Jr., "Low-power Viterbi decoder for CDMA mobile terminals," *IEEE J. Solid-State Circuits*, Vol. 33, pp. 473-482, Mar. 1998.

[2] Chi-Ying Tsui et. al., "Low Power ACS Unit Design for the Viterbi Decoder," *ISCAS-99*, pp. 137-140.

[3] Ghoneima M., Sharaf K., Ragai H.F., El-Halim Zekry, "Low power units for the Viterbi decoder," *the 43rd IEEE Midwest Symposium on Circuits and Systems*, Vol. pp.412-415, August, 2000.

[4] Kevin Page, Paul M. Chau, "Improved Architectures for the Add-Compare-Select Operation in Long Constraint Length Viterbi Decoding," *IEEE J. Solid-State Circuits*, Vol. 33, No. 1, Jan 1998.

[5] M. A. Bree, D. E. Dodds, "A Bit-Serial Architecture for a VLSI Viterbi Processor," *Conference Proc, WESCANEX 88, Digital Commun*, pp.72-77, May 12, 1988.

[6] J. B. Anderson and S. Mohan, "Sequential coding algorithms: A survey and cost analysis," *IEEE Trans. Commun.*, Vol. 32, pp. 169-176, Feb. 1984.

[7] S. J. Simmons, "Breadth-first trellis decoding with adaptive effort," *IEEE Trans. Commun.*, Vol. 38, pp. 3-12, Jan. 1990.

[8] R. Henning and C. Chakrabarti, "An approach for adaptively approximating the Viterbi

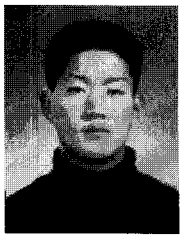
algorithm to reduce power consumption while decoding convolutional codes," *IEEE Trans. Signal Processing*, Vol. 52, no. 5, pp. 1443-1451, May 2004.

[9] Fei Sun and Tong Zhang, "Parallel High-Throughput Limited Search Trellis Decoder VLSI Desigin," *IEEE Trans. Very Large Scale Integr.(VLSI) Syst.*, Vol. 13, no. 9, Sept 2005

[10] Andrew J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. Inform. Theory*, Vol. IT-13, pp. 260-269, April 1967.

[11] Stephen B. Wicker, Error control systems for digital communication and storage, *Prentice Hall*, Inc,. 1995.

저 자 소 개



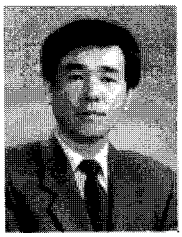
방 승 화(정회원)

2005년 서강대학교 컴퓨터공학과 학사 졸업.

2007년 서강대학교 컴퓨터공학과 석사 졸업.

2007년 현재 (주)인테그마 R&D팀 주임 연구원

<주관심분야 : 통신 알고리즘, VLSI 설계, SoC 설계>



임 종 석(평생회원)

1981년 서강대학교 전자공학과 학사 졸업.

1983년 한국과학기술원 전기 및 전자공학과 석사 졸업.

1989년 University of Maryland, College Park 전기공학과 박사 졸업.

1983년~1990년 8월 한국전자통신연구소 연구원.

1990년 9월~현재 서강대학교 컴퓨터공학과 교수.

<주관심분야 : 알고리즘, CAD, VLSI 설계>