고속 Huffman Codec 설계

°김병규, 김윤홍, 박종태, 이강현
조선대학교 전자정보통신공학부 MEDAL
http://medal.chosun.ac.kr
khrhee@vlsi.chosun.ac.kr

A Design on the High Speed Huffman Codec

°Byung Guy Kim, Yun Hong Kim, Jong Tae Park, Kang Hyeon Rhee
School of Electronics, Information and Communications Eng.
Multimedia ASIC Lab., Chosun University

요 약

멀티미디어 응용분야에 사용하는 데이터는 매우 크다. 이러한 대용량의 데이터를 관리하기 위해서 압축은 반듯이 필요하다. 본 논문에서는 허프만 부호화, 복호화 과정의 불규칙한 정렬구조와, 검색과정을 없애서 고속의 허프만 부호화 및 복호화가 가능한 허프만 코덱의 하드웨어 구조를 제안하였다.

1. 서 론

정보화 시대에 사용되는 데이터는 매우 크므로 이를 저장 매체에 저장하거나, 통신환경에 사용할 경우 매우 큰 대역폭을 필요로 한다. 데이터 표현을 작게 할 수 있다면 시스템의 경제적, 시간적 비용을 줄일 수 있다.

데이터 압축은 데이터의 표현 방식을 변환시킴으로써 이러한 문제에 대한 해결을 시도한다. 압축방식은 크게 무손실 압축과 유손실 압축 두 가지로 구분된다. 무손실 압축은 복호 후에 데이터가 원시 데이터와 완전히 같은 것으로 에러가 전혀 없다. 유손실 압축은 복호 후에 원시 데이터에 비해 어느 정도 에러를 허용하는 방식이다. 무손실 압축은 주로 에러를 허용하지 않는 텍스트 문서에서 사용되고, 유손실 압축은 어느 정도 에러가 허용되는 이미지, 음성 데이터 같은 아날로그 데이터를 디지털 변조한 데이터를 압축하는데 주로 사용한다. 유손실 압축은 특정 경우에 매우 효율적이나 모든 경우에 적용이 불가능하다. 그러나 무손실 압축은 압축이 필요한 모든 데이터에 적용이 가능하다.

허프만 코덱(huffman codec)을 하드웨어로 구현할 때 비교횟수의 불규칙함과, 불규칙한 타이밍으로 인해 하드웨어로 설계하는데 어려움이 있었다.

본 논문에서는 허프만 코덱을 하드웨어로 구현할 때, 정렬과정에 규칙성을 갖게 하고, 또한 정확한 타임을 예측하여 분산 파이프라인으로 구성함으로써 고속의 허프만 코덱을 구현할 수 있는 하드웨어 구조를 제안한다.

2. 이론적 배경

데이터 압축이란 원래의 데이터 보다 작은 부호로 표현하는 것이다.

심벌의 정보량 $I(s_i)$는 식(1)로 표현된다.

$$I(s_i) = \log \frac{1}{P(s_i)} \qquad (1)$$

$P(s_i)$: 심벌의 발생확률

즉, 심벌의 정보량은 심벌의 발생 확률과 반비례한다.

심벌이 모여서 만들어진 시스템의 에너지량을 "엔트로피(entropy)" 라하며 심벌의 엔트로피 $H(s)$는 식(2)과 같이 표현된다.

$$H(s) = \sum_{i=0}^{n} P(s_i) I(s_i)$$

$$= \sum_{i=0}^{n} P(s_i) \log \frac{1}{P(s_i)} \quad (2)$$

Shannon의 정보이론에 의해 평균 부호길이 $L(s)$는 엔트로피보다 적게 압축할 수는 없다[2]. 즉 $H(s) \le L(s)$관계에 있다. 그러나 $L(s)$를 최소화하여 부호화 함으로써 데이터를 압축한다.

1952년 D. A Huffman에 의해 제안된 허프만 코드는 가장 빈번히 발생하는 심벌에 가장 작은 부호를 할당하여 가장 빈번한 심벌의 엔트로피를 작게 하여 압축한다.

3. 고속 허프만 코덱 설계

3.1 허프만 부호화 과정.

허프만 부호화는 먼저 압축할 데이터 군에서 심벌의 발생 빈도를 카운트하고, 발생 빈도에 따라 허프만 테이블을 생성한 후 이 테이블에 근거하여 허프만 부호화를 한다. 그림 1은 허프만 부호화 과정을 나타낸다.

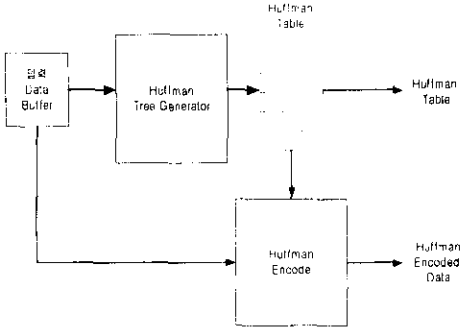

그림 1. 허프만 부호화 과정

본 논문에서는 통계적 기법의 허프만 부호화를 사용한다. 통계적 허프만 부호화에서는 입력 데이터의 통계적 발생빈도를 카운트하여 발생빈도에 따라 허프만 테이블을 생성하여 부호화를 한다.

3.2 발생빈도 카운트(probability counter)

입력 심벌 8bit, 블록 크기 8Kbyte로 하는 허프만 코덱을 설계하였다. 부호화할 데이터의 발생빈도를 카운트함으로써 허프만 부호화는 시작된다. 그림 2는 발생빈도 카운터의 구성 블록도 이다.

버퍼의 데이터를 고속으로 전송하기 위해서 버스 대역폭은 64bit로 하였다. 입력 데이터는 메모리 디코더를 지나면 발생빈도 카운터를 선택하는 선택신호 1비트로 바뀌진다. 즉 심벌의 값이 주소를 지정하는 포인터가 된다. 발생빈도 카운터에는 동시에 8개의 선택신호가 발생할 수 있으므로 발생빈도 카운터는 동시에 8개의 심벌에 대한 발생빈도를 카운트 해야한다. 또한 1클럭에 8개씩 입력 심벌을 처리함으로 같은 심벌이 연속적으로 있을 경우 1클럭에 최대 8까지 발생빈도가 증가될 수 있어야 한다. 이러한 병렬 고속 처리를 위해 다음과 같은 구조의 발생 빈도 카운터를 설계하였다. 여기서 카운트 레지스터의 크기는 8Kbyte 블록에 대해 테이블을 작성함으로 13bit이다. 그림 2는 8개의 심벌에 발생빈도를 동시에 카운트할 수 있도록 발생빈도 카운터를 심벌의 종류만큼 병렬로 배치한 것이다.

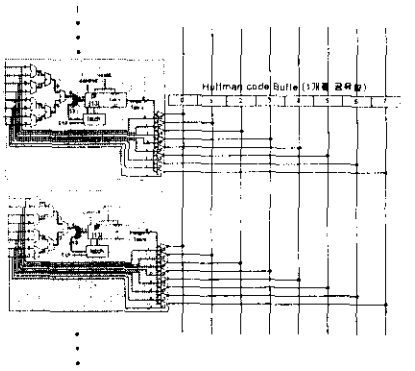

그림 2. 발생빈도 카운터 구성

3.3 발생빈도에 따른 허프만 트리의 생성

8Kbyte의 입력에 대한 발생빈도가 카운트되면 발생빈도를 기준으로 정렬하여 허프만 트리를 구성하고 허프만 테이블을 생성한다. 정렬 방법은 3개씩 그룹으로 비교하고 이것을 다시 9개의 그룹으로 묶어서 4단계로 나누어 비교 정렬한다. 여기서 3개씩 그룹으로 비교하는 것을 "3sort"이라고 표현하고 이렇게 3개씩 묶어진 무리를 "3sort 그룹"이라고 표현한다. 그림 3은 3sort 알고리즘의 비교 방법을 보여준다.

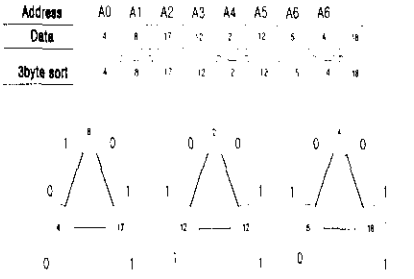

그림 3. 3sort 비교 알고리즘

9개의 심벌에 대해 3sort하면 플래그 값은 (00, 10, 11), (11, 00, 11), (10, 00, 11)이 되고, 플래그 값에 따라 정렬하면 (17, 8, 4), (12, 12, 2), (18, 5, 4)이 된다. 단 발생빈도가 같은 값도 플래그를 1로 설정하고, 발생빈도가 같은 경우 보다 하위 주소에 존재하는 발생빈도 값이 우선 순위를 갖게 했다. 정렬된 3sort 그룹을 다시 순위별로 정렬하여 9sort를 시행한다. 순위별 정렬이 되면 이것을 다시 3sort 그룹으로 나누어 3sort를 시행하면 가장 왼쪽에 9개의 발생빈도 카운트 값 중 최대치를 찾아낼 수 있다. 이 과정은 발생빈도 값이 모두 쉬프트될 때 까지 시행한다. 그림 4는 9sort의 처리 과정이다.

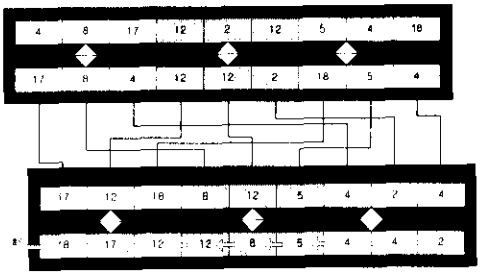

그림 4. 9sort 처리과정

9sort과정은 먼저 재 정렬된 3sort 그룹에 대해 3sort를 시행하고, 왼쪽으로 쉬프트 하여 9개의 발생빈도 카운트 값 중에서 발생빈도가 최대인 값을 출력하고 다시 3sort를 시행하여 나머지 8개의 발생빈도 카운트 값 중 최대 값을 출력한다. 이러한 과정을 9회 거치면 9sort 그룹에 대한 정렬은 모두 끝난다. 또한 매 클럭 마다 발생빈도가 높은 값에서 낮은 값 순으로 출력되므로 파이프라인을 적용할 수 있다. 이렇게 출력된 값은 다음 단계에서 다시 9sort과정을 거쳐 또 다시 9sort 과정을 발생빈도가 최대인 값을 찾아낸다. 이러한 9sort 과정은 각 단계 내에 식(3)과 같이 존재한다.

$$N = \frac{S}{9} \qquad (3)$$

N : 각 단계 9sort수.

S : 각 단계의 비교할 발생빈도 카운트 수.

따라서 각 단계별 9sort수는 다음과 같다.

1 단계 : $\frac{256}{9} = 28.44..$ 올림해서 29회,

2 단계 : $\frac{29}{9} = 3.22..$ 올림해서 4회,

3 단계 : $\frac{4}{9} = 0.44..$ 올림해서 1회

결과적으로 256개의 발생빈도 카운트 값을 정렬하는데 파이프라인으로 구성하여 단계별로 각각 29, 4, 1 총 34개의 9sorter가 필요하다. 또 최초로 출력이 발생하기까지 총 3단계이고 9sort는 3sort Clock의 1/2 클럭을 사용하여 3클럭 후에 최초의 값이 나온다. 그리고 각 단계가 파이프라인으로 구성되므로 이후 매 클럭마다 결과가 출력된다. 출력은 발생빈도가 가장 높은 값에서 가장 낮은 값 순으로 출력되므로 출력순서에 따라 허프만 부호를 할당한다. 이러한 발생빈도 값에 따라 실제로 정렬하는 것은 아니고 주소 값을 저장하여 메모리를 디코딩할 뿐이다. 그리고 디코딩된 위치에 해당하는 발생 빈도 카운터 내부에 있는 허프만 테이블 레지스터에 값을 할당한다. 여기까지 해서 허프만 테이블이 생성되었다. 그림 5는 발생빈도 카운터와 발생빈도 정렬기(probability sorter)의 구성을 보여준다.

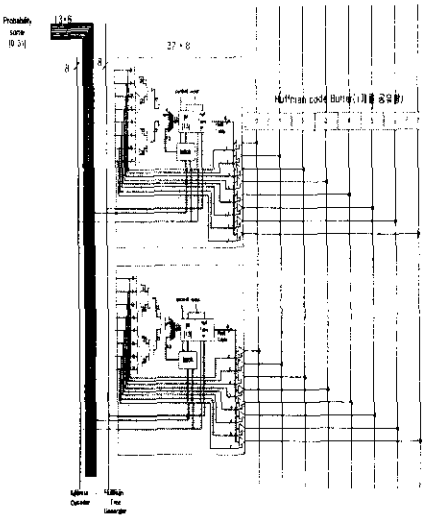

그림 5. 발생빈도 카운터와 발생빈도 정렬기 구성

이렇게 해서 허프만 테이블이 생성되면 다시 압축하려는 8Kbyte의 블록문자들을 입력하여 이번에는 빈도를 카운트하지 않고 해당번지의 허프만 부호를 출력한다. 이런 과정으로 허프만 부호화기가 구현되었다.

허프만 복호화기는 허프만 부호화기의 기능을 재 사용하여 구현한다. 복호화기는 먼저 부호화기로부터 허프만 테이블을 전송 받고, 전송 받은 데이터는 주소 순서대로 되었음으로 차례로 주소를 카운트하면서 저장하면 된다. 그리고 허프만 부호화기와 똑같은 방식으로 허프만 코드의 값이 허프만 테이블의 주소와 같게 하여 검색과정을 없앴다. 그림 6은 전체 허프만 부호화기의 구조를 보여준다.

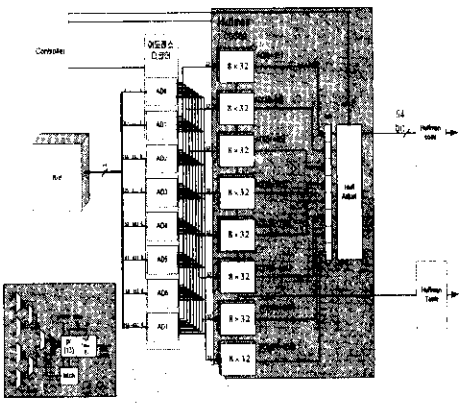

그림 6. 허프만 코덱의 전체 구조

4. 시뮬레이션 결과 및 고찰

본 논문에서 구현한 허프만 코덱은 압축이 필요한 모든 경우 사용할 수 있는 범용 허프만 코덱이다. 시뮬레이션 결과, 이미지 파일의 경우 평균 36%의 효율을 보였고 텍스트 파일의 경우 평균 44%의 압축효율을 보였다. 바이너리 파일의 경우 평균 43%의 압축률을 나타냈다. 압축 결과는 허프만 테이블이 포함된 결과이다. 또한 일반 상용 프로그램에 비해 2~7%정도 압축률이 떨어지는 경향을 보인다. 일반적인 보통 압축 프로그램은 여러 가지 압축 알고리즘을 혼용하여 사용함으로 압축효율을 향상 시키고 있으나, 본 논문의 허프만 코덱은 단지 허프만 부호화 알고리즘만을 사용함으로 이러한 결과가 나왔다. 그러므로 본 코덱의 압축효율을 높이기 위해서는 그러한 여러 가지 알고리즘을 혼합하여 사용함으로써 좀더 좋은 결과를 얻을 수 있다. 그러나 본 논문의 구조로 하드웨어를 설계할 경우 기존의 정렬과정의 불규칙한 비교횟수가 없어지고 비교횟수와 비교 타이밍이 정확하여 파이프라인 구조로 설계가 가능하고, 또한 검색 과정이 없으므로 고속의 허프만 코덱이 구현되었다.

본 논문의 허프만 코덱은 외부 클럭을 1/2 분주한 내부 클럭를 사용하고 최초의 허프만 테이블 출력은 외부 클럭기준으로 6 클럭후에 출력되고 매 2클럭마다 출력이 된다. 따라서 6클럭후 테이블전송이 가능하고 모두 512클럭 후에는 모든테이블을 전송할 수 있게된다. 그리고 513클럭부터 매 1클럭에 허부만 부호화된 출력이 8개씩 출력된다. 따라서 8Kbyte를 부호화하여 전송하는데 소요되는 클럭은 모두 $512+\frac{8\text{Kbyte}}{8}\times 2$ 으로 3024클럭이 소요된다.

본 허프만 코덱은 대부분 레지스터로 구성되어 있으므로 외부 메모리를 사용하여 칩의 크기를 줄일 수 있다.

5. 결 론

본 논문에서는 고속 허프만 코덱을 설계했다. 제안된 허프만 코덱은 압축할 데이터의 종류를 미리 정해두고 입력 문자의 2진 코드를 포인터로 사용하도록 하여 심벌의 검색과 빈도 카운트 오버헤드를 없애고, 내부 버스를 64비트 병렬처리하고, 허프만 테이블 생성시 3sort, 다시 9개의 그룹을 분산정렬 하여 분산 파이프라인 정렬함으로써 고속의 허프만 코덱을 설계하였다. 복호기는 부호화기를 재사용함으로써 기능블럭의 중복성이 제거되었다.

참고문헌

[1]. D. A. Huffman. "A Method for the Construction of Minimum Redundancy Codes," Proceedings of the Institute of Radio Engineers, 40 (1951), 1098-1101.

[2]. C. E. Shannon, "Mathmatical Theory of Communication," Bell System Thechnical Jouranl, Vol.27, No.3, July 1948, pp.379-423.

[3]. D. E. Knuth. "Dynamic Huffman Coding," Journal of Algorithms, 6 (1985), 163-180.

[4]. James D. Murray and William Vanryper "GRAPHICS FILE FORMATS SECOND EDITION" O' REILLY & ASSOCIATES, INC.