論文96-33B-4-7

한글 한자 비트 맵 폰트의 압축과 복원에 관한 연구

(A Study on Compression and Decompression of Hanguel and Chinese Character Bit Map Font)

趙 璟 衍 *

(Gyoung-Yun Cho)

요 약

본 논문에서는 한글 및 한자 비트 맵 폰트의 압축과 복원을 실시간적으로 수행하는데 적합한 가변 길이 블럭 코드를 제안한다. 제안한 가변 길이 블럭 코드를 명조체와 고딕체 완성형 한글과 바탕체와 돋움체 한자 비트 맵 폰트에 적용하여 압축율이 좋음을 보인다. 그리고 압축과 복원을 수행하는 ASIC(주문형 반도체)을 설계하고 CAD상에서 시뮬레이션하여 동작을 확인한다. ASIC은 0.8 미크론 CMOS 게이트 어레이로 구현하여 약 5,200 게이트가 소요되었으며 33MHz 클럭으로 시뮬레이션하였다. 따라서 제안한 알고리즘은 간단한 하드웨어로 최고 33Mbit/sec로 압축 및 복원을 수행하여 실시간 응용에 적합하다.

Abstract

In this paper, a variable length block code for real time compresssion and decompression of Hanguel and Chinese character bit map font is proposed. The proposed code shows a good compression ratio in complete form of Hangeul Myoungjo and Godik style and Chinese Batang and Doddum style bit map font. Besides, a compression and decompression ASIC is designed and simulated on CAD. The 0.8 micron CMOS Sea of Gate is used to implement the ASIC in amount of 5,200 gates, and it runs at 33MHz in simulator. Therefore, the proposed variable length block code could be implemented with simple hardware and compress and decompress at 33M bit/sec at maximum, which is ideal for real time applications.

I. 서 론

가정용 멀티미디어 기기는 표현 매체에 따라서 TV를 중심으로 하는 기기와 PC를 중심으로 하는 기기로 구분되어 발전하고 있다. TV를 중심으로 하는 기기는 큰 화면이 주는 현장감과 낮은 가격이 장점으로 CD-i, Video-CD, CDG, CDEG, CD-가라오케 등이 있다[1]. 이들 기기의 성능이 발전하면서 다양한 서체의 글자를 표현하려는 요구가 증대되고 있다. 또한 산업용 제어기기등에서도 멘-머신 인터페이스 환경이 그라픽 중심으로 바뀌면서 실시간적으로 다양한 서체를 생성하는 기능이 요구되고 있다. 이들 기기에서는 한정된 몇 가지 종류의 서체를 실시간적으로 표현하는 것이 필요하므로 윤곽선이나 구조적 폰트와 같이 복잡한 하드웨어나 고성능의 프로세서를 요구하는 방식[2,3]보다는 단순한 비트 맵 폰트를 선호하고 있다.

그러나 비트 맵 폰트는 서체 종류별로 작성해야 하므로 데이타 양이 대단히 크다. 예로써 48 X 48 크기 폰트를 비트 맵으로 표현하려면 288 바이트가 소요된다. 완성형 한글이 2,350자이므로 서체 한 벌에 676,800 바이트가 필요하다. 여기에 한자 4,888자를 포함하면 2,084,544 바이트라는 엄청난 양에 이르므로 데이타 압축이 필수적이다. 그리고 멀티미디어 기기의 특성상 실시간 복원이 필요하므로 비교적 간단한 하드웨어에

* 正會員, 釜山水産大學校 工科大學 컴퓨터工學科
(Department of Computer Engineering National Fisheries University of Pusan.)

接受日字:1995年6月16日, 수정완료일:1996年3月21日

의해서 압축 및 복원이 가능한 알고리즘에 관한 연구가 요구된다.

데이타 압축에 관한 연구는 Shannon이 정보를 수량화하여 표현하는 정보이론을 발표[4]하면서부터 여러 분야에서 폭넓게 진행되고 있다. 비트 맵 폰트는 비손실 압축이 필요하므로 엔트로피 압축인 허프만 코드[5]나 산술 코드[6]를 고려할 수 있다. 이들 코드는 발생빈도에 따라서 적합한 코드를 부여하므로 압축효과를 얻을 수 있다. 따라서 코드의 발생빈도를 측정하는 것이 중요하나, 서체에 따라서 코드의 발생빈도가 다르므로 공통적인 코드를 구하기 어렵다. 또 발생 빈도가 낮은 코드는 긴 비트열로 표현되어 대기시간(latency time)이 길어지며, 이를 개선하기 위한 병렬 디코더는 대규모 하드웨어가 요구되므로[7] 비트 맵 폰트의 실시간 압축 및 복원에는 적합하지 않다.

또다른 압축 방법으로 사전식 압축법인 LZ 법 및 LZ를 개량한 여러 방법들이 있다[8]. 그러나 비트 맵 폰트는 글자 단위로 접근이 가능해야 하는데 한 글자의 데이타 양은 수백바이트 이하로 작아서 사전식은 적합하지 않다.

본 논문에서는 비트 맵 폰트의 실시간적 압축 및 복원에 적합하도록 블럭 코드를 개선한 가칭 가변 길이 블럭 코드를 제안한다. 종래 블럭 코드는 가변길이 런 렝스(Run Length) 코드로 자료의 특성에 무관하게 블럭 크기가 결정되므로 압축율이 좋지 않다. 이러한 단점을 개선하기 위하여 자료의 특성에 따라서 블럭 크기를 조정하는 코드가 가변 길이 블럭 코드이다. 제안한 가변 길이 블럭 코드를 명조체와 고딕체 완성형 한글 2,350자와 바탕체와 돋움체 한자 4,888자의 24 X 24, 32 X 32, 40 X 40 및 48 X 48 비트 맵 폰트에 적용하여 압축율을 구하고, 허프만 코드의 압축율과 비교하여 충분히 실용성이 있음을 보인다.

또한 가변 길이 블럭 코드에 의한 압축과 복원을 수행하는 ASIC(주문형 반도체)을 설계하고 이를 CAD 상에서 시뮬레이션하여 동작을 확인한다. 설계한 ASIC은 0.8 미크론 CMOS 게이트 어레이로 구현하여 COMPASS에서 33MHz로 시뮬레이션하였다. 소요된 게이트는 약 5,200 게이트로 85.54 mil X 85.54 mil 크기의 실리콘에 집적하였다. 따라서 허프만 코드에 의한 방식[9-11]보다 하드웨어가 대단히 간단하다. 또한 33MHz 클록으로 동작하므로 최대 33M bps로 압축 및 복원이 가능하여 실시간 응용에 적합하다. 나아가서 허프만 코드나 산술 코드와는 달리 테이블 조사 과정이 없으므로 병렬동작이 가능하여, 동작 속도를 크게 개선시킬 수 있다.

Ⅱ. 가변 길이 블럭 코드

비트 맵 폰트는 비트 '0'과 '1'의 연속으로 해석된다. 이때 연속되는 동일 비트의 길이를 런 렝스라고 하며 런 렝스를 2진수로 표시한 후, MSB부터 일정한 길이로 구분하여 블럭을 만들고 각 블럭의 선두에 영역 비트를 부가하는 방식이 블럭 코드이다[13]. 2 비트 길이 블럭 코드를 표 1에 보인다. W는 블럭을 구분하는 영역비트로 '0' 런에서는 '0', '1' 런에서는 '1'이 되며, x는 런 렝스를 2진수로 표기한 것이다. 예를 들어서 '1' 런이 5, '0'런이 22인 데이타를 2 비트 길이 블럭 코드로 표현하면 '1'런은 'W01W01'로 '101101', '0' 런은 'W01W01W10'으로 '001001010'이므로 코드는 '101101001001010'이 된다. 이와같이 영역 비트 W를 반복해서 사용하므로 여러가지 길이의 런 렝스를 혼돈없이 표현할 수 있다.

표 1. 2 비트 길이 블럭 코드
Table 1. 2 bit length block code.

런 렝 스	코 드	코 드 길 이
1 - 3	Wxx	3 bit
4 - 15	WxxWxx	6 bit
16 - 63	WxxWxxWxx	9 bit
64 - 255	WxxWxxWxxWxx	12 bit

표 1에서 코드 'W00W10'는 런 렝스 2가 되지만 코드 'W10'과 중복되므로 사용하지 않는다. 따라서 코드가 낭비된다. 이러한 단점을 해소하기 위하여 N 블럭의 런 렝스를 식 (1)로 표기한다.

$$MRL = RL - \sum_{i=0}^{N-1} 2^{(i \cdot BLS)} \quad (1)$$

MRL = Modified run length

RL = Run length

BLS = Block length

식 (1)의 누적항은 중복 런 렝스를 합산하여 데이타의 런 렝스에서 삭제하기 위한 것이다. 예를 들어서, 2 비트 길이 블럭 코드에서 'WxxWxxWxx'로 표현되

는 코드는 N=3, BLS=2이므로 식 (1)의 누적항에서 1, 4, 16이 각각 더해져서 RL로부터 빼진다. 그러므로 'W00W00W00'는 런 렝스 21을 나타내며 'W11W11 W11'는 런 렝스 21+63, 즉 84를 나타낸다. 식 (1)을 적용하여 변형된 2 비트 길이 블럭 코드를 표 2에 보인다.

표 2. 변형 2 비트 길이 블럭 코드
Table 2. Modified 2 bit length block code.

런 렝 스	코 드	코 드 길 이
1 - 4	Wxx	3 bit
5 - 20	WxxWxx	6 bit
21 - 84	WxxWxxWxx	9 bit
85 - 340	WxxWxxWxxWxx	12 bit

이러한 변형 블럭 코드에서는 예로써 런 렝스 64-84가 9 비트 코드로 표현되므로 압축율이 개선된다. 그러나 각 블럭의 길이가 일정하게 고정되어 있으므로 원시 자료의 특성을 반영하지 못한다. 그러므로 원시 자료를 분석하여 압축율이 최고가 되도록 각 블럭의 길이를 설정한 코드가 가변 길이 블럭 코드이다. 가변 길이 블럭 코드는 각 블럭의 길이가 다르므로 N 블럭의 런 렝스는 N-1 블럭까지의 중복 런 렝스를 제거하여 표현한다. 그러므로 j 블럭의 길이를 BLSj라 하면 N 블럭의 런 렝스는 식 (2)로 표기된다.

$$VBRL = RL - \sum_{i=0}^{N-1} 2^{\sum_{j=0}^{i} BLS_j} \quad (2)$$

VBRL = Variable length block code run length

RL = Run length

BLSj = jth block length

가변 길이 블럭 코드에서는 각각의 블럭 길이가 자료의 성질에 따라서 압축율이 최대가 되도록 설정하여야 한다. 이러한 알고리즘을 표 3에 보인다.

각 블럭 크기는 1 비트부터 8 비트사이의 값으로 가정하고, 최대 블럭 크기는 32 비트로 한정한다. 32 비트이면 4G 비트의 런 렝스를 표현하므로 실용상 충분하다. 첫번째 블럭 N=1부터 블럭 길이를 산출한다. N=2 블럭 길이를 32 비트로 가정하고, N=1 블럭 길이를 1 비트부터 8 비트까지 변화해 가면서 압축율을 구하여, 가장 높은 압축율을 보이는 블럭 길이를 결정한다. N=1 블럭 길이가 결정되면, N=2 블럭 길이를 동일한 방법으로 구한다. 이러한 과정을 N+1 블럭을 도입하지 않을 때까지 반복한다.

표 3. 가변 길이 블럭 코드 알고리즘
Table 3. Algorithm for variable length block code.

```
Variable block length BLS :

{ N=0 /* N - block number */ ;
  New_block = true ;
  DO while (New_block)
   { N++ /* goto next block */ ;
     Set N + 1 block length to 32 ;
     New_block = false ;
     Set Min_Comp_size to infinity ;
     FOR (i=1, i++, i<=8)
      { Set N block length to i ;
        Calculate compressed code size ;
        If N + 1 block is used then
            New_block = true ;
        If Min_Comp_size> Compressed code
           then { j = i ;
           Min_Comp_size = compress code;
                }
     Set N block length to j ;
   }
}
```

상기 알고리즘은 원시 자료를 수십번이상 파싱하여 시간이 과도하게 요구되므로 페스-1에서 발생 빈도를 포함하는 런 렝스 테이블을 작성하고, 페스-2에서는 작성된 테이블을 참조하여 압축율을 구하는 2 페스 알고리즘으로 변형시켜서 사용한다.

Ⅲ. 시뮬레이션 및 평가

가변 길이 블럭 코드를 고딕체와 명조체 완성형 한글 2,350자의 비트 맵 폰트에 적용하여 블럭 길이와 압축율을 구한 것을 표 4에 보인다.

고딕체 48 X 48 완성형 2,350자의 비트 맵 폰트 크기는 676,800 바이트이다. 이를 가변 길이 블럭 코드로 압축하면 201,407 바이트로 원래 크기의 29.75%가 되며, 압축율은 원시 코드 크기를 압축된 코드 크기로 나눈 값으로 3.36이 된다. 또 명조체 24 X 24 서체의 비트 맵 폰트 크기는 169,200 바이트로 압축하면 84,672 바이트가 되어 압축율은 1.98이고, 원시 코드의 약 50%가 된다. 표 4에서 고딕체가 명조체에 비하여 압축율이 다소 높게 나타나고 있는 데, 이것은 고딕체가 명조체보다 획의 모양이 단순한 것에 기인하는 것으로 보인다. 또한 작은 서체에 비하여 큰 서체의 압축율이 높게 나타나고 있으며, 가장 작은 24 X 24 서체에서도

압축율이 2 정도로 원시 코드 크기의 50% 수준인 좋은 압축율을 보이고 있다. 큰 서체인 48 X 48 서체는 압축율이 3.2 정도로 압축된 코드가 원시 코드의 30% 수준으로 높은 압축율을 보이고 있다.

표 4. 한글의 가변 길이 블럭 코드 압축율
Table 4. Variable length block code compression ratio of Hangeul.

서 체	가변길이 블럭코드	압 축 율
고딕 48 X 48	0xxx0x0x0xx0x0x 1xx1x1x1x	3.36
명조 48 X 48	0xxxx0x0xxx0x 1xx1x1x1x	3.24
고딕 40 X 40	0xxx0x0x0x0x0x 1xx1x1x1x	3.02
명조 40 X 40	0xxx0xx0xxx0x 1xx1x1x1x	2.96
고딕 32 X 32	0xxxx0x0x0x0x 1xx1x1x	2.27
명조 32 X 32	0xxx0x0x0x0x0x 1x1x1x1x	2.39
고딕 24 X 24	0xx0x0x0xx0x 1x1x1x1x	2.02
명조 24 X 24	0xxx0x0xx0x 1x1x1x1x	1.98

한편 가변 블럭 길이는 서체에 따라서 조금씩 다르다. '0' 런은 첫번 블럭의 길이가 3-4 비트이며, 다음 블럭은 대부분이 1 비트이나 서체에 따라서 2 또는 3 비트 길이가 된다. 한편 '1' 런은 첫번 블럭의 길이가 1-2 비트이며, 다음 블럭은 모두 1 비트 길이이다. 이것은 비트 맵 폰트의 바탕이 '0' 런으로 긴 '0' 런 렝스가 많기 때문으로 해석된다. 이에 반하여 '1'런은 글자의 획을 구성하는 성분으로 빈도수도 많지 않고, 런 렝스도 폰트의 가로폭을 넘지 않기 때문에 긴 런 렝스가 존재하지 않는다.

한편 폰트는 행간에 유사성이 높으므로 인접한 행의 변화분만을 '1' 비트로 변화시키면 '0' 런이 증가하게 된다. 이러한 행간 XORing 전처리에 의하여 압축율을 높일 수 있다[12]. 표 5에 행간 XORing 전처리를 한 후에 가변 길이 블럭 코드를 적용한 압축율을 보인다.

표 5로부터 행간 XORing이 압축율을 크게 높이고 있음을 알 수 있다. 고딕 48 X 48 서체에서는 원시 코드 676,800 바이트가 128,851 바이트로 19.03%로 압축되어서 압축율이 5.25가 되었다. 고딕체가 명조체보다 더욱 개선되었는데 이것은 고딕체가 명조체보다 행간 유사성이 높기 때문이다. 또한 가변 길이 블럭 코드도 '0' 런의 종류가 다양화되어서 첫번 블럭의 길이가 고딕체에서는 4 비트, 명조체에서는 3 비트로 고정되었으며, 다음 블럭의 길이도 거의 대부분이 1 비트이다. '1' 런은 그 수가 줄면서 모든 블럭의 길이가 1 비트로 되었다.

표 5. 행간 XORing 전처리 후 한글의 가변 길이 블럭 코드 압축율
Table 5. Varaiable length block code compression ratio of Hangeul with line XORing preprocessing.

서 체	가변길이 블럭코드	압 축 율
고딕 48 X 48	0xxxx0x0xx0x0x 1x1x1x1x1x	5.25
명조 48 X 48	0xxx0xx0x0x0x0x 1x1x1x1x1x	3.77
고딕 40 X 40	0xxxx0x0x0x0x0x 1x1x1x1x	4.51
명조 40 X 40	0xxx0xx0x0x0x0x 1x1x1x1x	3.38
고딕 32 X 32	0xxxx0x0x0x0x 1x1x1x1x	3.21
명조 32 X 32	0xxx0x0x0x0x0x 1x1x1x1x	2.51
고딕 24 X 24	0xxxx0x0x0x 1x1x1x1x	2.56
명조 24 X 24	0xxx0x0x0x0x 1x1x1x1x	2.05

KSC-5601 코드에서 성의한 4,888자 한자에서 행간 XORing 전처리 후의 압축율을 구하여 표 6에 보인다. WINDOW의 바탕체와 돋움체 48 X 48, 40 X 40, 32 X 32, 24 X 24 크기의 비트 맵 폰트를 사용하였다.

바탕체에 비하여 돋움체의 획이 굵으므로 돋움체의 압축율이 좋게 나타나고 있다. 돋움체 48 X 48 한자 4,888자의 비트 맵 폰트는 1,407,744 바이트로 압축하면 465,466 바이트로 33.06%로 줄어든다. 따라서 압축율은 3.02가 된다. 바탕체에 있어서는 1,407,744 바이트가 568,502 바이트로 40.38%로 압축되어 압축율은 2.48이 된다. 압축율이 2를 넘어서면 50% 이하의 크기로 압축되는 것을 나타내며 40 비트 이상의 큰 한자 폰트는 이에 속한다. 가장 압축율이 낮은 바탕체 24 X 24 폰트도 원시 자료 351,936 바이트가 256,251 바이트로 72.81%로 압축된다.

한자의 가변 블럭 길이는 한글과 유사하다. '0' 런의 처음 블럭 길이는 2-3 비트이며 다른 블럭 길이는 모두 1 비트이다. 또한 '1' 런은 한글과 동일하게 모두 1 비트이다. 행간 XORing 전처리는 인접한 행의 차이만을 '1' 비트로 나타내고 있으므로, 가변 길이 블럭 코드

는 서체의 종류에 덜 민감하게 된다. 이에 반하여 전처리를 하지 않으면 한글에서와 동일하게 압축율이 다소 낮아지고 가변 블럭 길이도 좀더 다양하게 구성된다.

표 6. 행간 XORing 전처리 후 한자의 가변 길이 블럭 코드 압축율

Table 6. Variable length block code compression ratio of Chinese characater with line XORing preprocessing.

서 체	가변길이 블럭코드	압 축 율
바탕 48 X 48	0xxx0x0x0x0x0x0x0x 1x1x1x1x1x	2.48
돋움 48 X 48	0xxx0x0x0x0x0x0x0x 1x1x1x1x1x	3.02
바탕 40 X 40	0xx0x0x0x0x0x0x0x 1x1x1x1x1x	2.12
돋움 40 X 40	0xxx0x0x0x0x0x0x 1x1x1x1x	2.49
바탕 32 X 32	0xx0x0x0x0x0x0x0x 1x1x1x1x1x	1.76
돋움 32 X 32	0xx0x0x0x0x0x0x0x 1x1x1x1x1x	2.03
바탕 24 X 24	0xx0x0x0x0x0x0x 1x1x1x1x	1.37
돋움 24 X 24	0xx0x0x0x0x0x0x 1x1x1x1x	1.54

표 7에 완성형 한글 고딕체 48 X 48 서체를 행간 XORing 전처리 후 허프만 코드(Huffman code), 변형 허프만 코드(Modified HC)로 압축한 결과를 보인다.

표 7. 한글 고딕체의 허프만 코드 압축

Table 7. Compression data by Huffman code of Hanguel godik style bit map font.

Huffman Code	압축율 : 5.78 '0' 런 코드 수 : 494 '1' 런 코드 수 : 34 가장 긴 '0' 런 코드 : 17 비트 가장 긴 '1' 런 코드 : 14 비트
Modified Huffman Code	압축율 : 5.53 '0' 런 코드 수 : 77 '1' 런 코드 수 : 34 가장 긴 '0' 런 코드 : 15 비트 가장 긴 '1' 런 코드 : 14 비트

표 7로부터 허프만 코드와 변형 허프만 코드의 압축율은 각각 5.78과 5.53으로 표 5의 가변 길이 블럭 코드의 5.25에 비하여 다소 높은 것으로 나타나고 있다. 압축된 코드 크기는 가변 길이 블럭 코드를 100으로 하였을 때, 허프만 코드와 변형 허프만 코드는 각각 91과 95가 된다. 그러므로 압축된 코드는 가변 길이 블럭 코드가 허프만 코드에 비하여 5% 내지 9% 더 크다.

한편 허프만 코드는 '0' 런 코드의 종류가 494개로 대단히 많다. 따라서 방대한 허프만 코드 테이블이 필요하며, 나아가서 이를 하드웨어로 구성하는 것은 대단히 무리이다. 또 변형 허프만 코드는 메이크 업 코드의 도입으로 코드 수를 줄였으나 아직도 '0' 런 코드의 수가 77개로 상당히 많은 편으로, 하드웨어 구성시 부담이 되는 수치이다. 또한 가장 긴 코드는 허프만 코드에서 17 비트 길이이다. 이것은 최고 17 비트를 파싱해야 압축된 코드를 복원할 수 있는 것을 의미한다. 따라서 화상 처리와 같이 실시간 처리를 요구하는 분야에서는 대기 시간이 과다하게 소요되어 적합하지 않다.

이에 반하여 가변 길이 블럭 코드는 압축율이 변형 허프만 코드의 95%로 별다른 차이가 없으며, 한글 고딕 서체의 경우에는 '0' 런 코드에 7개 블럭, '1' 런 코드에 5개 블럭의 길이만을 정의하면 된다. 명조체 한글 및 두가지 서체의 한자를 포함하여도 '0'런 코드는 8개 블럭, '1' 런 코드는 5개 블럭만으로 정의할 수 있다. 또한 각 블럭 길이는 4 비트이하이므로 2 비트로 표현할 수 있다. 따라서 허프만 코드에 비하여 대단히 간단한 하드웨어만으로 허프만 코드와 비슷한 높은 압축율을 얻을 수 있다.

명조체 한글과 두가지 서체의 한자에 있어서도 표 7과 동일한 현상을 보이고 있다. 또한 바이트 단위로 허프만 코드를 적용하면 모든 서체에서 동일하게 코드 수는 100개 이상 200여개까지 나타나며 압축율은 가변 길이 블럭 코드에 비하여 5% 정도 떨어진다. 따라서 하드웨어도 복잡하고 압축율도 나빠서 바이트단위 허프만 코드는 적합하지 않다.

Ⅳ. ASIC 설계

제안한 가변 길이 블럭 코드의 압축과 복원을 수행하는 ASIC(Variable Length Block CODEC : VLB-CODEC)을 설계하여 워크스테이션에서 Compass로 시뮬레이션하였다. VLB-CODEC은 그림 1의 18 핀 패키지로 구성하였다.

VLB-CODEC은 CLK에 동기되어서 동작하며, 최대 한 클럭에 1 비트씩을 압축 또는 복원한다. 내부 레지스타는 8 바이트로 이들을 3 비트의 어드레스 버스로 지정한다. 표 8에 내부 레지스타 구성을 보인다.

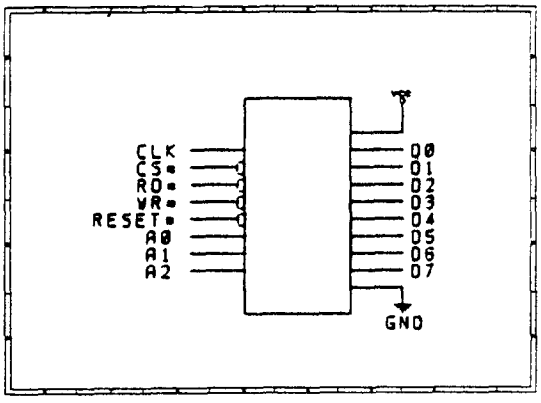


그림 1. VLB-CODEC 페키지 구성도
Fig. 1. VLB-CODEC package.

표 8. VLB-CODEC 내부 레지스타
Table 8. VLB-CODEC internal register.

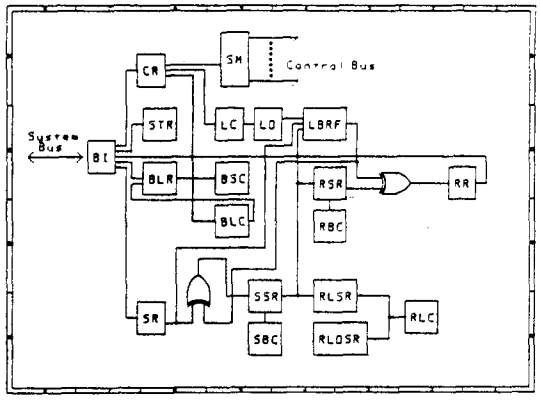
Address	Definition
0 CR (WO)	b7 : enable b6 : compress/decompress# b5 b3 : line size b2-b0 : reserved
0 STR (RO)	b7 : result register(RR) full b6 : idle state b5-b1 : reserved b0 : source register (SR) full
1 SR (WO) RR (RO)	SR : source register RR : result register
2,3	reseved
4 7 BLR (WO)	Block length register 4 5 : white run (8 block) 6 7 : black run (8 block) 2 bit / block

명령 레지스타 CR은 쓰기 전용(WO)으로 VLB CODEC의 동작을 지정한다. 비트 7이 '1'이면 압축 또는 복원을 수행하며, '0'이면 동작하지 않는다. 비트 6이 '1'이면 압축을, '0'이면 복원 기능을 수행한다. 행간 XORing 전처리는 비트 5 - 비트 3으로 지정한다. 이들 비트가 '000'이면 행간 XORing 전처리를 하지 않으며, 다른 수이면 폰트 행의 바이트 수에서 1을 뺀 수를 나타낸다. 즉, 48 비트 행의 폰트는 6 바이트 행이므로 여기에서 1을 빼면 '101'이 된다. 따라서 16 비트부터 64 비트 행까지를 8 비트 단위로 지정한다.

상태 레지스타 STR은 읽기 전용(RO)으로 VLB-CODEC의 상태를 나타낸다. 압축 또는 복원 동작으로 결과 레지스타(RR)에 새로운 바이트가 만들어지면 비트 7이 '1'로 된다. 결과 레지스타를 읽어 가면 비트 7은 자동적으로 '0'이 된다. 또한 비트 0은 소스 레지스타(SR)에 처리할 데이타가 있으면 '1'이 된다. 비트 6는 모든 동작이 완료되면 '1'이 된다.

3장의 연구 결과로부터 VLB-CODEC에서 최대 런 렝스는 32 비트, 블럭 길이는 최대 4비트로 정하였으며, 8개 블럭까지를 레지스타로 지정하였다. 9번째 블럭부터는 압축율에 별다른 영향을 주지 않으므로 블럭 길이를 4비트로 고정하였다. 블럭 길이는 블럭 길이 레지스타(BLR)에 프로그램한다.

그림 2에 VLB-CODEC의 하드웨어 블럭도를 보인다.



BLC : Block line counter, 3 bit up.
RLC : Run length counter, 32 bit up/down.
BSC : Block length counter, 2 bit down.
SBC : Source byte counter, 3 bit up.
RBC : Result byte counter, 3 bit up.
RLSR : Run length shift register,
32 bit SI-PO shift left.
RLOSR : Run legth offset shift register,
32 bit SI-PO shift left/right.
SSR : Source shift register,
8 bit PI-SO shift left.
RSR : Result shift register,
8 bit SI-PO shift left.

그림 2. VLB-CODEC의 블럭도
Fig. 2. VLB-CODEC block diagram.

그림 2에서 버스 인터페이스 BI는 VLB-CODEC과 내부 레지스타를 연결한다. 버스 인터페이스를 통하여 명령이 명령 레지스타 CR에 전달되고 또한 상태 레지스타 SR을 판독하여 VLB-CODEC의 동작 상태를 확인할 수 있다. 명령 레지스타에 전달된 명령은 상태

기계 SM에서 번역되고, 상태 기계에서 번역된 결과를 내부 제어 신호선에 출력하여 압축 및 복원 기능이 수행된다.

압축 동작에서 버스 인터페이스로부터 입력된 원시 데이타는 소스 레지스타 SR에 기억되고, 상태 기계의 제어에 의하여 소스 시프트 레지스타 SSR로 전달된다. 이때 행간 XORing 전처리 기능이 정의되어 있으면 전행의 데이타와 XOR를 취하여 소스 시프트 레지스타로 전송한다. 또 다음 행의 전처리를 위하여 소스 레지스타의 데이타는 라인 버퍼 레지스타 파일 LBRF에 저장한다. 소스 레지스타의 데이타가 전송되면 스테터스 레지스타에 소스 레지스타가 비어 있음을 기록하여 새로운 데이타를 받아들일 준비를 한다. 이와같이 이중 버퍼를 취하므로 동작중에 새로운 원시 데이타를 입력하여 호스트 컴퓨터의 대기 시간을 줄일 수 있다.

소스 시프트 레지스타에 전달된 데이타는 비트 단위로 시프트되면서 압축이 진행된다. 최초의 비트는 상태 기계에 기억되며, 동시에 블럭 라인 카운터 BLC가 0으로 크리어되어 첫번째 블럭을 지정하고, 최초 비트의 타입에 따라서 '0' 또는 '1' 런의 블럭 길이를 블럭 라인 레지스타 BLR에서 블럭 길이 레지스타 BSC로 전송한다. 또한 런 렝스 오프셋 레지스타 RLOSR과 런 렝스 카운터 RLC를 0으로 초기화시킨다. 이어서 블럭 길이 레지스타 값만큼 런 렝스 오프셋 레지스타를 직렬 입력 '1'을 가하면서 시프트시킨다. 최초 비트의 처리가 끝나면, 소스 시프트 레지스타로부터 다음 비트를 시프트시킨다. 다음 비트가 최초 비트와 동일하면 런 렝스 카운터를 '1' 증가시킨다. 증가된 값이 런 렝스 오프셋 레지스타의 값보다 커지면 런 렝스 카운터를 다시 '0'으로 크리어시키고, 블럭 라인 카운터를 '1'증가하여 다음 블럭 길이를 블럭 길이 레지스타로 전송하여, 이 값만큼 런 렝스 오프셋 레지스타를 시프트시킨다. 이러한 동작을 최초 비트와 다른 비트가 나타날 때까지 반복한다. 다른 비트가 나타나면 하나의 런이 끝났으므로 런 렝스 카운터에 저장되어 있는 값을 런 렝스 오프셋 레지스타와 블럭 라인 카운터를 참조하여 가변 길이 블럭 코드 형식으로 변환하여 한 비트씩 결과 시프트 레지스타 RSR로 전송한다. 결과 시프트 레지스타에 8비트 바이트가 모아지면 결과 레지스타에 전송하고, 이를 상태 레지스타에 기록한다.

복원은 압축 과정과 반대로 진행되어, 런 렝스가 런 렝스 시프트 레지스타 RLSR에 시프트되어 기억되고, 런 렝스 오프셋 레지스타는 블럭 길이를 기억한다. 새로운 블럭이 출현하면 런 렝스 오프셋 레지스타 값이 런 렝스 카운터로 전송되고, 런 렝스 카운터를 '1'씩 감소시키면서 복원된 비트가 결과 시프트 레지스타로 전송된다. 결과 시프트 레지스타에 8 비트가 모아지면 결과 레지스타로 전송된다. 이때 행간 XORing 전처리된 것을 복원하기 위하여 전행과 XOR를 취하여 결과 레지스타에 전송한다. 또 다음 행의 XORing 처리를 위하여 행 버퍼 레지스타 파일 LBRF에도 전송된다. 런 렝스 카운터가 0으로 감소되면, 런 렝스 오프셋 레지스타는 다음 블럭 길이만큼 시프트된다. 하나의 런 코드가 끝나면 런 렝스 시프트 레지스타 값이 런 렝스 카운터로 전송되어, 런 렝스 만큼의 복원 비트가 결과 레지스타로 전송된다.

VLB-CODEC의 모든 동작을 제어하는 상태 기계는 20가지 상태를 가지며, 설계된 VLB-CODEC은 0.8 미크론 게이트 어레이로 구현하여, 약 5,200 게이트가 소요되었으며 133.32 X 133.32 mil 크기의 실리콘에 집적하였다. Compass에서 33MHz 클럭으로 시뮬레이션하여 동작을 확인하였다.

V. 결 론

사용자 환경이 그라픽 중심으로 변화되면서 가정용 멀티미디어 기기 및 산업용 제어기기 등에서 다양한 서체를 실시간적으로 출력하려는 요구가 증대되고 있다. 이러한 요구에 부응하기 위하여 본 논문에서는 비트 맵 폰트를 하드웨어에 의하여 실시간적으로 압축 및 복원하는 가변 길이 블럭 코드를 제안하였다. 가변 길이 블럭 코드는 런 렝스 코드로 원시 데이타의 성질에 따라서 블럭 길이가 달라지는 블럭 코드이다. 따라서 압축율이 우수하면서 상대적으로 간단한 하드웨어에 의하여 실시간적으로 비트 맵 폰트를 압축 또는 복원할 수 있다.

가변 길이 블럭 코드를 KSC 5601 완성형 한글 2,350자 명조체 및 고딕체와 한자 4,888자 바탕체 및 돋움체 24 X 24, 32 X 32, 40 X 40, 48 X 48 크기의 비트 맵 폰트에 적용하여 압축율을 구하였다. 행간 XORing 전처리를 병행하여 고딕체 한글에서 압축율이 2.56 ~ 5.25이 되며, 돋움체 한자에서는 1.54 ~ 3.02의 압축율을 보였다. 변형 허프만 코드와 비교하여 비슷한 수준의 압축율을 가지며, 복원시에 대기 시간이

짧고, 코드 테이블이 필요없으므로 간단한 하드웨어로 구현할 수 있는 장점을 가진다.

또한 가변 길이 블럭 코드에 의한 압축과 복원을 수행하는 하드웨어를 ASIC으로 설계하고, SUN 워크스테이션에서 Compass로 시뮬레이션하여 동작을 확인하였다. 설계한 ASIC은 0.8 미크론 CMOS 게이트 어레이로 구현하여, 132.32 X 132.32 mil 크기의 실리콘에 집적하여 약 5,200 게이트가 소요되었고, 33MHz 클럭으로 시뮬레이션하였다.

본 논문에서 제안한 가변 길이 블럭 코드는 간단한 ASIC으로 복원 및 압축을 실시간적으로 수행할 수 있으므로, 가정용 멀티미디어 기기나 산업용 기기등에 폭넓게 활용할 수 있다.

참 고 문 헌

[1] 이성희, "멀티미디어 단말기-Video CD Player를 중심으로," 전자공학회지, Vol. 22, No. pp. 88-94, Apr. 1995

[2] N. Kai, et al, "A high speed outline font rastering LSI," in Proc. CICC, pp. 24. 6. 1 - 24. 6. 4, May 1989.

[3] Kawata, et al, "An Outline Rendering Processor with an Embedded RISC CPU for High Speed Hint Processing," *IEEE Journal of Solid State Circuits*, Vol. 29, No. 3, pp. 280-289, Mar. 1994.

[4] C.E. Shannon, "A Mathematical Theory of Communication," *Bell System Technical Journal*, Vol. 27, pp. 379-423, 1948.

[5] D.A. Huffman, "A Method for the Construction of Minimum Redundancy Codes," Proc., IRE. Vol. 40, pp. 1098-1101, 1952.

[6] G.G. Langdom, "An Introduction to Arithmetic Coding," *IBM Journal of Research and Development*, Vol. 28-2, pp. 135-149, Mar. 1979.

[7] Keshab K. Parhi, "High-Speed VLSI Architecture for Huffman and Viterbi Decoders," IEEE Transactions on Circuits and Systems-II, Vol. 39, No. 6, pp. 385-391, Jun. 1992.

[8] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," IEEE Transactions on Information Theory, Vol. IT-23, pp. 337-343, 1977.

[9] Amar Mukherjee et al, "Efficient VLSI Designs for Data Transformation of Tree-Bases Codes," IEEE Transactions on Circuits and Systems, Vol. 38, No. 3, pp. 306-314, Mar. 1991.

[10] Heonchul Park, "Area Efficient VLSI Architectures for Huffman Coding," IEEE Transactions on Circuits and Systems-II, Vol. 40, No. 9, pp. 568-575, Sep. 1993.

[11] Amar Mukherjee et al, "MARVLE: A VLSI Chip for Data Compressing Using Tree-Based Codes," IEEE Transactions on VLSI Systems, Vol. 1-2, pp. 203-214, Jun. 1993.

[12] 우 정원, 김 홍배, 조 경연, 이 정현, "전처리에 의한 비트 맵 한글 폰트의 압축 방법," 제6회 한글 및 한국어정보처리 학술대회, 한국 정보 과학회, pp. 231-234, Dec. 1994

[13] 김 태균, 최 형진 편저, 화상 처리 기초, 정익사, pp. 120-128, Aug. 1990

저 자 소 개

趙 璟 衍(正會員)

1990년 2월 인하대학교 공과대학 전자공학과 정보공학전공 (공학박사). 1983년 3월 ~ 1991년 2월 삼보컴퓨터 기술연구소 책임연구원. 1991년 3월 ~ 1995년 3월 부산수산대학교 자연과학대학 전자계산학과 조교수. 1995년 4월 ~ 현재 부산수산대학교 공과대학 컴퓨터공학과 조교수. 1991년 3월 ~ 현재 삼보컴퓨터 기술연구소 비상임 고문. 1993년 6월 ~ 현재 아남반도체기술(주) 비상임 고문. 관심분야 : 전자계산기구조, ASIC 회로 설계, ASIC memory