니블 RLE 코드에 의한 비트 맵 데이타의 압축과 복원에 관한 연구

조 경 연†

요 약

본 논문에서는 한글 비트 맵 폰트와 프린터 데이타의 실시간 압축과 복원에 적합한 니블 RLE(Run Length Encoding)코드를 제안한다. 제안한 코드를 명조체와 고딕체 완성형 한글 폰트와 프린터 출력 데이타에 적용하여 압축율이 좋음을 보인다. 그리고 압축과 복원을 분리하여 각각 하나의 ASIC(주문형 반도체)으로 설계하고 CAD상에서 시뮬레이션하여 동작을 확인한다. ASIC은 0.8 미크론 CMOS 게이트 어레이로 설계하였 약 2,400 게이트가 소요되었으며 25MHz 클럭으로 동작하였다. 따라서 제안한 코드는 간단한 하드웨어로 최고 100M bit/sec로 압축 및 복원을 수행하여 실시간 응용에 적합하다.

A Study on Compression and Decompression of Bit Map Data by Nibble RLE Code

Gyung-Yun Cho†

ABSTRACT

In this paper, a nibble RLE(Run Length Encoding) code for real time compresssion and decompression of Hanguel bit map font and printer data is proposed. The nibble RLE code shows good compression ratio in complete form Hangeul Myoungjo and Godik style bit map font and printer output bit map data. And two ASICs seperating compression and decompression are designed and simulated on CAD to verify the proposed code. The 0.8 micron CMOS Sea of Gate is used to implement the ASICs in amount of 2,400 gates, and these are running at 25MHz. Therefore, the proposed code could be implemented with simple hardware and performs 100M bit/sec compression and decompression at maximum, it is good for real time applications.

1. 서 론

개인용 컴퓨터의 보급 확대와 GUI 환경으로의 발전은 높은 질의 출력물에 대한 욕구를 증가시켰다. 이에 따라서 윤곽선 및 구조적 폰트를 사용하는 워드프로세서와 복잡한 도형 및 화상을 포함하는 DTP의 보급이 확대되고 있다. 이러한 경향은 WINDOW의 보급으로 더욱 가속화되고 있다.

윤곽선 및 구조적 폰트는 글자의 확대와 축소가 자유롭고 다양한 형태의 글자 생성이 가능하나[1] 복잡한 연산이 필요하여 시스템 전체의 속도가 저하된다. 따라서 하드웨어에 의한 윤곽선 글자 생성이 연구[2, 3]되고 있으나 그 복잡성으로 아직 실용화되지 못하고 있다. 다른 해결책으로 생성된 비트 맵 폰트를 폰트 캐시에 저장하여 사용하므로써 생성에 따른 성능 저하를 최소화시킬 수 있다[4]. 비디오에서는 폰트 캐시를 비디오 메모리에 하드웨어적으로 설정하는 것이 바람직하나 한글의 경우에는 글자 수가 많아서 큰 용량의 폰트 캐시가 필요하다. 따라서 비트 맵 폰트를 압축하여 폰트 캐시에 저장하고 이를

† 정 회 원:부산수산대학교 공과대학 컴퓨터공학과 조교수
논문접수:1995년 6월17일, 심사완료:1995년11월 2일

실시간적으로 복원하는 연구가 요구된다.

일반 프린터에서는 출력하려는 내용을 페이지 기술 언어로 작성하여 출력하고, 이를 프린터에서 해석하는 방식과 본체에서 비트 맵으로 변환하여 프린터에 출력하는 두가지 방식이 있다. 전자의 방식은 고성능 프로세서가 요구되므로 가격이 높아지는 단점이 있다. 후자는 보급형 프린터에서도 고품질의 출력물을 얻을 수 있는 장점이 있으나 비트 맵 데이타의 양이 방대하므로 전송에 많은 시간이 소요된다. 최근 DMA(Direct Memory Access)에 의한 프린터 출력이 보편화되고 있으나 근본적으로 데이타의 양이 줄지는 않는다. 그러므로 프린터에서도 비트 맵 폰트를 실시간적으로 압축 및 복원하는 연구가 요구된다.

비트 맵 폰트의 압축은 비손실 압축이 요구되며, 엔트로피 압축인 호프만 코드, 산술 코드 등과 유니버설 코드인 LZ 코드등의 여러 방식[5-9]을 고려할 수 있다. 한편 폰트는 개별적인 접근이 가능해야 하는데, 예로써 48 X 48 크기 폰트에서 한 글자가 288 바이트로 데이타 양이 작지만, 완성형 한글은 2,350자로 서체 한 벌의 비트 맵 폰트는 676,800 바이트가 된다. 이러한 한글 폰트의 특수성을 살린 압축 방법은 아직 연구가 미미한 실정이다.

이러한 요구에 부응하여 본 논문에서는 한글 비트 맵 폰트의 데이타 특성을 조사하여 실시간 하드웨어 압축 및 복원에 적합한 니블 RLE (Run Length Encoding) 코드를 제안한다. 그리고 240, 300, 400 DPI 프린터에 제안한 코드를 적용하여 효용성을 증명한다. 또한 제안한 코드를 실현하는 하드웨어를 설계하고 CAD상에서 시뮬레이션하여 동작을 확인한다. 설계한 ASIC은 시뮬레이션 상에서 25MHz 클럭으로 동작하였으며, 본 ASIC은 니블 단위로 압축 및 복원하므로 최고 100M bps의 성능을 발휘할 수 있다. 따라서 본 논문의 연구 결과는 비트 맵 데이타의 실시간적인 압축과 복원에 특히 유용하다.

2. 한글 비트 맵 폰트의 분석

비트 맵 폰트를 압축하는 방식은 폰트를 0과 1의 연속된 데이타로 보아서 비트 단위로 처리하는 방식과 비트를 묶어서 코드 단위로 처리하는 두가지 방법이 있다. 본 논문에서는 압축율이 비교적 낮으나 실시간 적용에 유리한 코드 단위를 선택한다. 또한 여러 가지 서체와 크기를 고려하기 위하여 한글 완성형 KSC-5601 코드 2,350자의 명조체와 고딕체 각각에 대하여 24×24, 32×32, 40×40 및 48×48 크기의 비트 맵 폰트를 대상으로 하였다.

일차적으로 바이트 단위로 호프만 코드를 적용한 결과를 〈표 1〉에 보인다.

〈표 1〉 호프만 코드에 의한 압축율
(Table 1) Compression ratio by Huffman code

	고딕 48×48	명조 48×48
압축율	3.04	3.12
호프만 코드 수	111	162
가장 긴 코드 길이	19bit	19bit

	고딕 24×24	명조 24×24
압축율	2.04	2.17
호프만 코드 수	171	224
가장 긴 코드 길이	18bit	17bit

압축율은 원시 코드 크기를 압축된 코드 크기로 나눈 값이다. 〈표 1〉에서 호프만 코드의 수가 100개가 넘어므로 하드웨어가 상당히 복잡해짐을 알 수 있다. 또 가장 긴 코드 길이가 19비트로 19클럭의 대기시간(latency time)이 요구되어서 실시간 적용에는 적합하지 않다.

이러한 단점을 개선하기 위해서는 코드 수를 줄여야 하므로 한글 비트 맵 폰트를 니블 단위로 분류하여 빈도수를 조사하였다. 그 결과를 〈표 2〉에 보인다.

〈표 2〉에서 고딕 48×48 폰트의 니블 코드 빈도는 코드 '0'이 72.57%로 압도적으로 많으며, 그 다음이 코드 'F'로 10.48%이다. 이들 두 코드가 83%를 점유하며 나머지 14개 코드가 17%의 발생 빈도를 보이고 있다. 명조 48×48 폰트에서도 코드 '0'이 78.42%로 압도적인 빈도를 보이고 있으며 고딕체와는 다르게 코드 'F'의 빈도가 3.99%로 다소 낮으나 상대적으로 높은 빈도를 차지하여 이들 두 코드가 83.4%로 고딕체와 유사한 분포를 이루고 있다. 24×24 서체는 글자 크기가 작으므로 코드 '0'와 코드 'F'의 빈

도가 48×48 서체보다는 다소 떨어지나 70% 이상의 빈도를 차지하는 것을 알 수 있다. 32×32 및 40×40 서체는 24×24와 48×48 서체 사이의 중간적인 분포를 이루어서 코드 '0'이 70% 이상을 점유하며 다음이 코드 'F'로 서체에 따라서 10% 내외를 나타낸다.

이와같이 특정한 코드에 발생 빈도가 극도로 몰려 있으면 높은 압축 효율을 올릴 수 있다. 그러므로 특정한 코드의 빈도수를 높이는 전처리 과정을 도입하여 더욱 효율화를 기할 수 있다. 한글 폰트는 그 구조상 인접한 2개 행이 유사한 비트 열로 구성되어 있다. 따라서 인접한 행을 XORing시키면 코드 '0'의 빈도수를 높일 수 있다. 이를 행간 XORing 전처리[10]라고 한다. 행간XORing 전처리를 수행한 한글 비트 맵 폰트의 니블 단위 빈도 수를 〈표 3〉에 보인다.

〈표 3〉에서 XORing 전처리에 의한 고딕 48×48 폰트는 코드 '0'의 빈도가 13.85% 증가하였으며 반대로 코드 'F'의 빈도는 5.5% 감소하였다. 명조 48×48 폰트도 코드 '0'의 빈도가 5.5% 증가하고 코드 'F'의 빈도는 1.1% 감소하

〈표 2〉 한글 비트 맵 폰트의 니블 단위 빈도
〈Table 2〉 Statistics of Nibble code in Hangeul bit map font

Code	고딕 48×48	명조 48×48
0	72.57%	78.42%
1	2.98%	2.51%
2	0.02%	0.10%
3	2.15%	3.03%
4	0.02%	0.11%
5	0.00%	0.00%
6	0.02%	0.55%
7	3.21%	3.10%
8	2.04%	2.97%
9	0.11%	0.07%
A	0.00%	0.00%
B	0.06%	0.04%
C	3.40%	2.65%
D	0.08%	0.04%
E	2.85%	2.42%
F	10.48%	3.99%

Code	고딕 24×24	명조 24×24
0	60.96%	66.96%
1	4.53%	4.17%
2	0.10%	2.17%
3	6.78%	3.15%
4	0.09%	5.74%
5	0.01%	0.05%
6	3.36%	1.50%
7	2.29%	1.33%
8	2.85%	4.43%
9	0.45%	0.20%
A	0.00%	0.19%
B	1.66%	0.15%
C	2.92%	3.77%
D	0.38%	0.09%
E	0.58%	1.52%
F	11.04%	4.58%

〈표 3〉 행간XORing 전처리한 한글 비트 맵 폰트의 니블 단위 빈도
〈Table 3〉 Statistics of Nibble code in XORing preprocessed Hangeul bit map font

Code	고딕 48×48	명조 48×48
0	86.42%	83.89%
1	1.74%	2.32%
2	0.81%	1.38%
3	0.74%	1.24%
4	0.02%	0.10%
5	0.22%	0.57%
6	0.69%	0.94%
7	1.69%	0.94%
8	1.65%	2.30%
9	0.05%	0.37%
A	0.02%	0.07%
B	0.02%	0.14%
C	1.03%	1.38%
D	0.02%	0.11%
E	0.74%	0.95%
F	4.98%	2.85%

Code	고딕 24×24	명조 48×48
0	73.75%	3.79%
1	3.79%	3.97%
2	1.10%	1.79%
3	2.45%	2.52%
4	1.00%	1.87%
5	1.18%	0.78%
6	0.89%	0.93%
7	1.66%	1.84%
8	3.28%	3.95%
9	0.61%	0.56%
A	0.20%	0.75%
B	0.29%	0.73%
C	2.35%	3.00%
D	0.18%	0.80%
E	1.78%	1.64%
F	6.49%	5.56%

였다. 또한 24×24 서체에서도 코드 '0'의 빈도가 한층 증가하고 있다. 이러한 현상은 32×32와 40×40 서체에서도 동일하게 나타고 있다.

3. 니블 RLE 코드

한글 비트 맵 폰트는 코드 '0'의 발생 빈도가 60%에서 87%로 압도적인 다수를 차지하고 있다. 즉 코드 '0'과 '1'의 연속은 많으나 이외의 코드는 연속되지 않는다. 그러므로 이들 연속이 많은 코드를 런 렝스로 표기하여서 압축 효과를 얻을 수 있다. 코드를 CD, 연속 발생 회수를 OT로 표기하면 일반적인 런 렝스 코드는 (식 1)로 표현된다.

CD OT 1

니블 RLE 코드에서 CD와 OT는 각기 하나의 니블 즉 4비트 길이이다. 이때 코드 '0'의 발생 빈도가 대단히 높으므로 코드 '0'에 대하여만 (식 1)로 표기한다. 그리고 코드 'F'는 서체에 따라서 빈도가 변화하므로 서체에 따라 (식 1)로 표기하기도 한다. 이외의 다른 코드는 발생 빈도가 낮으므로 발생 회수를 나타내지 않고 코드 자체로만 표기한다.

(식 1)에서는 OT가 니블 단위이므로 발생 회수는 16 이하로 한정된다. 이러한 단점을 해소하기 위하여 제한값 LN을 도입하여 3가지 형식의 OT를 정의한다. 제한값 LN은 압축된 코드 길이를 최적화하기 위한 값으로 본 논문에서는 16진수 8부터 F사이의 값을 가진다.

첫째로 제한값 LN보다 OT가 작은 경우로 (식 2)의 2 니블로 표현한다.

CD C1 ..... (2)

단, C1=OT−1
C1<LN

예를 들어 16진수로 000000600000H인 데이타는 '0'의 반복 회수가 각기 6과 5이므로 이를 니블 RLE 코드로 표현하면 05604이 된다. 이때 처음 '0'은 CD로 반복되는 니블 코드로 다음의 '5'에서 6개 반복되는 것을 나타낸다. 다음의 '6'은 압축되지 않는 니블로 6자체를 나타낸다.

둘째로 OT가 제한값 LN보다 크며 (식 3)의 3 니블로 표현 가능한 경우이다.

CD C2 C1 3

단, C2=INT((OT−1−LN)/16)+LN
INT(×)=the largest integer<=×
C1=(OT−1−LN) MOD 16
C2<15

셋째로 (식 3)에서 C2가 15보다 크거나 같은 경우로 (식 4)의 4 니블로 표현한다.

CD 'F' C2 C1 4
단, 'F'=16 진수 F
C2=INT((OT−1−(15−LN)*16−LN)/16)
C1=(OT−1−LN) MOD 16
C2<16

(식 4)로도 표현할 수 없는 큰 OT는 여러개의 코드로 분할하여 표현한다. 실시간 압축에서는 대기 시간이 일정한 값이하를 가지도록 하는 것이 바람직하다. 따라서 본 논문에서는 압축된 코드 길이를 최대 4니블로 제한하였다.

4. 비트 맵 폰트의 압축율 분석

제안한 니블 RLE코드를 한글 비트 맵 폰트에 적용한 압축율을 〈표 4〉에 보인다.

〈표 4〉로부터 고딕 48×48 폰트에서 코드 'F'를 포함한 경우에 압축율이 다소 높아지는 데 이

〈표 4〉 변형 RLE 코드에 의한 비트 맵 폰트의 압축율
〈Table 4〉 Compression ratio of bit map font by Nibble RLE

고딕 48×48				
LN	case 0	case 1	case 2	case 3
8	2.17	2.03	3.91	3.71
9	2.18	2.04	3.93	3.74
A	2.21	2.07	3.95	3.78
B	2.23	2.08	3.94	3.75
C	2.21	2.07	3.92	3.73
D	2.21	2.07	3.91	3.72
E	2.22	2.07	3.87	3.69
F	2.21	2.07	3.75	3.57

case 0=코드 'F'를 포함한 경우
case 1=코드 'F'를 포함하지 않은 경우
case 2=case 0에 행간 XORing 전처리
case 3=case 1에 행간 XORing 전처리

명조 48×48				
LN	case 0	case 1	case 2	case 3
8	2.22	2.23	2.92	2.96
9	2.23	2.25	2.97	3.01
A	2.27	2.28	3.02	3.07
B	2.30	2.31	3.05	3.09
C	2.29	2.30	3.04	3.08
D	2.29	2.30	3.03	3.07
E	2.29	2.30	3.01	3.06
F	2.29	2.30	2.99	3.03

고딕 24×24				
LN	case 0	case 1	case 2	case 3
8	1.38	1.39	1.89	1.94
9	1.38	1.40	1.92	1.98
A	1.38	1.40	1.93	1.99
B	1.38	1.40	1.93	1.99
C	1.38	1.40	1.94	2.00
D	1.39	1.40	1.95	2.01
E	1.39	1.39	1.96	2.02
F	1.37	1.38	1.92	1.98

명조 24×24				
LN	case 0	case 1	case 2	case 3
8	1.44	1.46	1.61	1.67
9	1.44	1.46	1.62	1.69
A	1.44	1.46	1.62	1.69
B	1.44	1.46	1.62	1.69
C	1.44	1.46	1.63	1.69
D	1.44	1.47	1.63	1.70
E	1.44	1.47	1.63	1.70
F	1.45	1.47	1.64	1.70

것은 코드 'F'의 빈도가 〈표 2〉에서 10.48%로 나타나고 있는 것과 일치하는 현상이다. 또한 명조 48×48 폰트에서는 코드 'F'가 압축율에 영향을 거의 주지 않고 있다. 이것은 〈표 2〉에서 코드 'F'의 빈도가 3.99%로 작은 것으로부터 예측할 수 있다. 제한값 LN은 대체적으로 16진수 'C'에서 'E'의 값에서 최적 압축율을 보이고 있다.

행간 XORing 전처리는 압축율에 큰 영향을 준다. 고딕 48×48 폰트에서는 전처리하지 않은 경우의 압축율 2.03－2.35가 행간 XORing 전처리를 수행하면 3.57－3.95로 대폭적으로 개선된다. 명조체 48×48 폰트에서도 행간 XORing은 고딕체보다는 작으나 30% 이상의 압축율 개선을 보이고 있다. 이것은 고딕체가 명조체에 비하여 행간에서 데이타 일치성이 높은 것에 기인한 것으로 보인다.

고딕 24×24 서체에서는 압축율이 1.4정도로 그다지 높지 않으나 행간 XORing 전처리를 하면 2.0대로 높아진다. 따라서 글자 크기가 작은 24×24 폰트에서도 충분한 압축 효율을 올릴 수 있다.

전반적으로 명조체가 고딕체에 비하여 압축율이 좋은데 이것은 코드 '0'과 코드 'F'의 빈도가 고딕체에 비하여 명조체가 높게 나타나기 때문이다. 또한 행간 XORing 전처리후에는 빈도수가 반전되기 때문이다.

고딕 40×40 폰트의 압축율은 1.96－2.08을 보이고 있으며 행간 XORing 전처리를 수행하면 3.23－3.45가 된다. 명조 40×40 폰트의 압축율도 2.08－2.13이며 행간 XORing 전처리에 의하여 2.70－2.86으로 압축율이 개선된다. 고딕 32×32 폰트 압축율은 1.5로 비교적 낮으나 행간 XORing 전처리로 2.5까지 개선되어 충분히 실용적이다. 명조 32×32 폰트에서도 1.7의 압축율을 보이며 행간 XORing 전처리후에는 2.0으로 개선된다.

5. 프린터 데이타의 압축율 분석

프린터에 전송되는 비트 맵 데이타의 압축율을 구하기 위하여 〈표 5〉에 보이는 9가지 종류의 문서를 작성하였다.

〈표 5〉의 문서는 '아래 한글 2.5'에서 여러 가지 서체와 다양한 글자 크기로 작성하였으며 각기 1페이지 분량이다. 프린터는 180 DPI용으로 도트 매트릭스 프린터 앱슨 LQ－1550을 사용하였으며, 300 DPI용으로 삼보의 레이저 프린터

〈표 5〉 프린터 비트 맵 데이타 분석을 위한 문서 파일
〈Table 5〉 Word processor file for bit map data analysis of printer

	내 용
File 1	학습 보고서
File 2	회사 업무 보고서
File 3	C 소스 프로그램
File 4	합격자 명단－표 작성
File 5	정부 공문서
File 6	모임 초청장
File 7	한글 자판 안내문
File 8	그림을 포함한 안내문
File 9	한문이 많은 문서

TG-600을, 400 DPI용으로 삼보의 레이저 프린터 TG-850을 각기 사용하였다. 프린터마다 제어 코드와 비트 맵 표현 형식이 다르나 제어 코드에 비하여 비트 맵 코드의 양이 압도적으로 많으므로 제어 코드에 따른 압축율의 차이는 무시해도 좋다. 〈표 6〉에 제한값을 12로 고정시키고 코드 'F'를 압축시키지 않은 프린터 비트 맵 데이타의 압축율을 보인다.

〈표 6〉 프린터 비트 맵 데이타의 압축율
(Table 6) Compression ratio of printer bit map data

	180 DPI Printer		
	원시 데이타 (byte)	압축 데이타 (byte)	압축율
File 1	182,920	60,279	3.03
File 2	195,916	59,231	3.31
File 3	126,496	39,165	3.23
File 4	139,667	54,517	2.56
File 5	153,175	49,171	3.12
File 6	222,740	43,656	5.10
File 7	216,296	80,654	2.68
File 8	150,540	89,161	1.67
File 9	185,053	77,911	2.38

	300 DPI Printer		
	원시 데이타 (byte)	압축 데이타 (byte)	압축율
File 1	593,236	116,025	5.11
File 2	583,015	113,607	5.13
File 3	480,015	82,537	5.82
File 4	404,292	78,840	5.13
File 5	517,177	90,344	5.72
File 6	627,304	87,740	7.15
File 7	647,416	148,231	4.37
File 8	484,190	181,046	2.67
File 9	563,364	154,419	3.65

	400 DPI Printer		
	원시 데이타 (byte)	압축 데이타 (byte)	압축율
File 1	926,159	179,542	5.16
File 2	953,519	174,099	5.48
File 3	683,066	126,796	5.39
File 4	687,619	115,898	5.93
File 5	855,568	142,945	5.98
File 6	1,082,162	134,536	8.05
File 7	1,106,312	231,767	4.77
File 8	755,544	262,166	2.88
File 9	958,161	243,462	3.94

〈표 6〉으로부터 24핀 도트 매트릭스 프린터로 가장 많이 사용되는 저해상도 180 DPI 프린터에서 압축율이 1.67-5.10에 이르는 것을 알 수 있다. 파일-6의 압축율이 5.10으로 특히 좋은 것은 모일 초청장으로 줄간 간격이 넓은 것에 기인한다. 또한 그림이 포함된 안내문인 파일-8은 화상 데이타의 압축율이 좋지 않기 때문에 전체적인 압축율이 다소 나쁘게 나타났다. 또한 한자는 한글보다 복잡하므로 한자가 대부분인 파일-9의 압축율이 2.38에 머무르고 있다. 그러나 일반적으로 많이 사용되는 문서의 경우는 2.5-3.3으로 양호한 것으로 평가된다. 또한 보급형 레이저 프린터인 300 DPI 프린터에서는 일반 문서가 5.0 이상의 좋은 압축율을 보이고 있으며, 고해상도 레이저 프린터인 400 DPI 프린터에서는 비슷한 수준이나 그림이 포함된 경우나 한자가 많은 경우에서 압축율이 약간 개선되었다. 잉크 제트 프린터에서 많이 사용되는 360 DPI 해상도에서도 유사한 압축율을 보인다.

400 DPI 프린터에서는 A4 한 페이지의 비트 맵 데이타양이 1메가 바이트로 대단히 크다. 따라서 이러한 많은 데이타를 프린터에 전송하는 데는 많은 시간이 소요된다. 그러나 본 논문의 코드를 적용하면 20% 이하로 데이타 양을 줄일 수 있다.

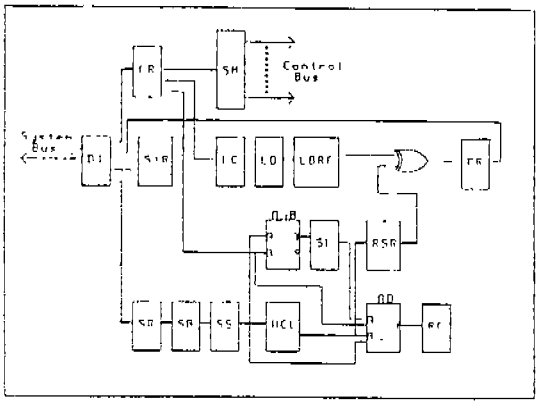
6. 하드웨어 설계 및 구현

제안한 니블 RLE 코드의 압축과 복원을 분리하여 각각 하나의 ASIC(주문형 반도체)로 설계하였으며, 설계된 회로는 SUN 워크스테이션에서 Compass로 시뮬레이션하여 동작을 확인하였다.

6.1 복원 ASIC

복원 ASIC은 외부와 8비트 데이타 폭으로 접속되며, 칩 셀랙터, 리드, 라이트, 어드레스 버스, 클럭, 리세트 및 전원을 포함하여 16핀으로 구성하였다. 내부 레지스타는 복원 시작, 행간 XORing 전처리를 위한 라인 크기, 제한값, 코드 'F'의 엔코드 여부를 결정하는 명령 레지스타와 압축된 원시 코드 버퍼와 복원된 결과 코드 버퍼

의 상태를 나타내는 상태 레지스타, 그리고 원시 코드와 결과 코드를 기억하는 데이타 레지스타로 구성한다. (그림 1)에 복원 ASIC의 블록도를 보인다.



(그림 1) 복원 ASIC의 블록도
(Fig. 1) Block diagram of decompress ASIC

(그림 1)에서 버스 인터페이스 BI는 ASIC 외부와 내부 레지스타를 연결한다. 버스 인터페이스를 통하여 명령이 명령 레지스타 CR에 전달되고 또한 상태 레지스타 SR을 판독하여 ASIC의 동작 상태를 확인할 수 있다. 명령 레지스타에 전달된 명령은 상태 기계 SM에서 번역되어 ASIC 기능이 수행된다.

버스 인터페이스로부터 입력된 원시 데이타는 소스 레지스타 SR에 기억되고, 상태 기계의 제어에 의하여 소스 데이타 버퍼 SB로 전달된다. 이와같이 이중 버퍼를 취하므로 복원중에 새로운 원시 데이타를 입력하여 호스트 컴퓨터의 대기 시간을 줄일 수 있다.

소스 데이타 버퍼에 전달된 원시 데이타는 니블 단위로 나누어진다. 이때 해당 니블의 선택은 소스 셀렉타 SS에 의하여 수행된다. 나누어진 니블은 상태 기계에서 해독되어 압축된 코드인가를 판단한다. 압축되지 않은 코드이면 결과 시프트 레지스타 RSR로 전송된다. 결과 시프트 레지스타는 2개의 결과 니블을 연결하여 바이트가 완성되면 이를 결과 레지스타 RR로 전송한다. 이때 행간 XORing 전처리된 것을 복원하기 위하여 이전 행과 XOR를 취하여 결과 레지스타에 전송한다. 또한 다음 행의 XORing 처리를 위하여 행 버퍼 레지스타 파일 LBRF에도 전송되며, 행내의 위치를 지정하는 행 카운터 LC를 1 증가시킨다. 행 카운터의 출력은 행 디코더 LD에서 해석되어 정확한 행 버퍼 레지스타의 위치를 지정한다.

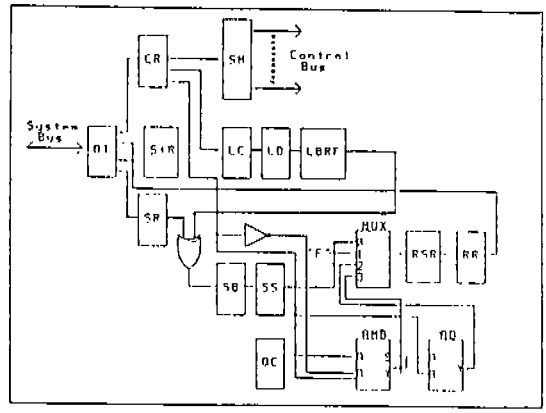
결과 레지스타에 새로운 데이타가 전송되면 상태 레지스타 SR에 그 상태를 기억시켜서 호스트 컴퓨터가 버스 인터페이스를 통하여 읽어 갈 수 있게 한다. 결과 레지스타와 결과 시프트 레지스타가 분리되는 이중 버퍼 구조를 취하여 호스트 컴퓨터의 대기 시간을 최소화하였다.

한편 압축된 코드이면 그 형태를 분석하여 발생 회수를 산출한다. 이를 행하는 회로가 뺄셈기 AMB, 뺄셈 결과 레지스타 SL, 상위 니블 발생 레지스타 HCL 및 8비트 덧셈기 AD이다. 이렇게 산출된 발생 회수는 결과 카운터 RC에 기억된다. 그리고 결과 카운터를 감소해가며 복원된 코드를 결과 시프트 레지스타로 전송한다.

이러한 복원 ASIC의 동작을 제어하는 상태 기계는 10가지 상태로 구성하였다. 설계된 복원 ASIC은 0.8 미크론 게이트 어레이로 구성하여, 약 2,400 게이트가 소요되었으며 133.32 X 133.32 mil 크기의 실리콘에 집적하였다. Compass에서 25MHz 클럭으로 시뮬레이션하여 동작을 확인하였다.

6.2 압축 ASIC

압축 ASIC은 복원 ASIC과 동일한 신호선을



(그림 2) 압축 ASIC의 블록도
(Fig. 2) Block diagram of compress ASIC

가진 16핀으로 내부 레지스타 구성도 유사하다. (그림 2)에 압축 ASIC의 블럭도를 보인다.

행간 XORing 전처리는 소스 데이타 버퍼 SB 앞단에서 수행한다. 전처리된 데이타는 소스 데이타 버퍼에 기억되며, 소스 셀렉타 SS에 의하여 니블 단위로 선택되어 상태 기계에서 코드의 종류를 판정한 후 결과 시프트 레지스타 RSR로 보내진다. 압축가능한 코드이면 다음 소스 데이타를 소스 셀렉타로부터 가져 와서 이전 코드와 동일한 것인가를 판단한다. 동일한 코드이면 9비트 발생 카운터 OC를 1 증가시킨다. 동일한 코드가 아니면 발생 카운터 내용을 분석하여 코드 종류를 판별하여 (식 2, 3, 4)의 적합한 코드를 생성한다. 이러한 코드 생성은 9 비트 뺄셈기 AMB, 4 비트 덧셈기 AD에서 수행한다. 압축 ASIC의 상태 기계는 16가지의 상태로 구성하였다. 설계된 압축 ASIC은 0.8 미크론 게이트 어레이로 구성하여, 약 2,400 게이트가 소요되었으며 133.32×133.32 mil 크기의 실리콘에 집적하였다. Compass 상에서 25MHz 클럭으로 시뮬레이션하여 동작을 확인하였다.

7. 결 론

고해상도 출력기기가 폭 넓게 보급되면서 높은 질의 문서 출력에 대한 요구가 증대되고 있다. 이러한 요구에 부응하기 위하여 비트 맵 데이타를 하드웨어에 의하여 실시간적으로 압축 및 복원하는 니블 RLE 코드를 제안하였다.

제안한 니블 RLE 코드는 비트 맵 데이타를 니블 단위로 구분하고 코드 '0'와 'F'에 대하여만 런 렝스로 표기한 것으로, 연속 발생 회수의 크기에 따라 3가지 포맷의 구조를 가진다.

니블 RLE 코드의 효용성을 증명하기 위해서 완성형 한글 2,350자 명조체와 고딕체 24×24, 32×32, 40×40, 48×48 크기의 비트 맵 폰트에서 압축율이 1.6—5.0에 이르는 것을 보였다. 또한 여러 서식의 9가지 문서를 '아래 한글 2.5' 워드프로세서에서 작성하고, 이를 180, 300, 400 DPI 프린터로 출력하여 출력되는 비트 맵 데이타에 대하여 제안한 코드를 적용하여 1.6—8.05의 압축율을 얻었다.

또 니블 RLE 코드를 실현하는 하드웨어를 압축과 복원을 분리하여 각각 ASIC으로 설계하고 SUN 워크스테이션에서 Compass로 시뮬레이션하여 25MHz 클럭에서 정상 동작하는 것을 확인하였다. 니블 RLE 코드는 4비트 단위 코드이므로 한 클럭에서 최대 4비트의 압축 및 복원을 수행하므로 최대 압축 및 복원 속도는 100Mbps가 된다. 이것은 종래 소프트웨어에 근거한 방식의 평균 압축 속도가 0.64Mbps[11], 1.57Mbps[12] 및 하드웨어 호프만 코드에 근거한 방식에서의 30Mbps[13]에 비하여 대단히 효율적이다. ASIC은 0.8 미크론 CMOS 게이트 어레이로 구성하였으며, 압축 복원에 각각 약 2,400 게이트가 소요되었고, 132.32×132.32 mil 크기의 실리콘에 집적하였다.

본 논문에서 제안한 니블 RLE 코드는 간단한 ASIC으로 복원과 압축 기능을 실시간적으로 구현할 수 있는 장점을 가진다. 따라서 비트 맵 폰트 캐시, 프린터 인터페이스에서의 비트 맵 데이타의 실시간 압축 및 복원에 유용하게 활용할 수 있다.

참 고 문 헌

[1] D.E.Knuth, The METAFONT Book, Reading, MA:Addison Wesley, 1986

[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] K. Yamaashi, et al, "Fast printing method for high quality Janapese Characters," Trans. IPS japan, vol. 31, no. 1, pp. 144-151, Jan. 1990.

[5] S. Golomb, "Run-length encodings," IEEE Transactions on Information Theory, Vol. IT-12, pp. 399-401, Jul. 1966

[6] G. Langdon, "An introduction to

Arithmetic coding," IBM Journal of Research and Development, Vol. 28, pp. 135-149, Mar. 1984

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

[8] Reza Hashemian, "Design and Hardware Implementation of a Memory Efficient Huffman Decoding," IEEE Transactions on Consumer Electronics, Vol. 40, No. 3, pp. 345-352, Aug. 1994

[9] Seung Bae Choi and Moon Ho Lee, "High Speed Pattern Matching for a Fast Huffman Decoder," IEEE Transactions on Consumer Electronics, Vol. 40, No. 1, pp. 97-103, Feb. 1995

[10] 우정원, 김홍배, 조경연, 이정현, "전처리에 의한 비트 맵 한글 폰트의 압축 방법," 제6회 한글 및 한국어정보처리 학술대회, 한국 정보과학회, pp. 231-234, Dec. 1994

[11] R. Lea, "Text compression with an associative parallel processors," Computer Journal, Vol. 21, pp. 45-56, 1978

[12] P. Hawthron, "Microprocessor assisted tuple access decompression and assembly for statistical data base systems," in Proc. VLDB, pp. 223-233, 1982

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

조 경 연

1990년 인하대학교 공과대학 전자공학과 정보공학전공(공학박사)
1983년~91년 삼보컴퓨터 기술연구소 책임연구원
1991년~95년 부산수산대학교 자연과학대학 전자계산학과 조교수
1993년~95년 부산수산대학교 전자계산소 소장
1995년~현재 부산수산대학교 공과대학 컴퓨터공학과 조교수
1991년~현재 삼보컴퓨터 기술연구소 비상임 고술고문
1993년~현재 아남반도체기술(주)비상임 기술고문
1995년~현재 대흥전자 비상임 기술고문
관심분야: 전자계산기구조,. ASIC 회로 설계, ASIC memory