論文2001-38SD-2-3

LZSS 알고리즘과 엔트로피 부호를 이용한 사전탐색처리장치를 갖는 부호기/복호기 단일-칩의 VLSI 설계 및 구현

(A VLSI Design and Implementation of a Single-Chip Encoder/Decoder with Dictionary Search Processor(DISP) using LZSS Algorithm and Entropy Coding)

金 鍾 燮*, 趙 相 福**

(Jong-Seop Kim and Sang-Bok Cho)

요 약

본 논문은 0.6μm CMOS 기술로 LZSS 알고리즘과 엔트로피 부호를 이용한 부호기/복호기 단일-칩의 VLSI 설계 및 구현에 관하여 기술하였다. 처리 속도 50MHz를 갖는 사전탐색처리장치(DISP)의 메모리는 2K×8bit 크기를 사용하였다. 이것은 매번 33개 클럭 중 한 개의 클럭은 사전의 WINDOW 배열을 갱신으로 사용하고 나머지 클럭은 주기마다 한 개의 데이터 기호를 바이트 단위로 압축을 실행한다. 결과적으로, LZSS 부호어 출력에 엔트로피 부호를 적용하여 46%의 평균 압축률을 보였다. 이것은 LZSS에 보다 7% 정도의 압축 성능이 향상된 것이다.

Abstract

This paper described a design and implementation of a single-chip encoder/decoder using the LZSS algorithm and entropy coding in 0.6μm CMOS technology. Dictionary storage for the dictionary search processor(DISP) used a 2K×8bit on-chip memory with 50MHz clock speed. It performs compression on byte-oriented input data at a data rate of one byte per clock cycle except when one out of every 33 cycles is used to update the string window of dictionary. In result, the average compression ratio is 46% by applied entropy coding of the LZSS codeword output. This is to improved on the compression performance of 7% much more then LZSS.

Ⅰ. 서 론

컴퓨터의 발전은 많은 컴퓨터들의 네트워킹으로 인하여 전보다 많은 용량의 데이터를 처리하게 되었다. 물리적으로 원거리 지역에서의 데이터 통신은 데이터를 통신 채널로 전송하는 데이터 전송 기술과 컴퓨터의 데이터 처리 기술을 의미한다. 데이터 전송 속도는 통신 채널의 전송 대역폭에 한정되고, 데이터 처리 속도는 데이터 처리량에 달려있어 있다. 컴퓨터로 많은 용량의 데이터 처리와 통신 전송 매체의 제한된 대역폭은 정보통신시스템의 병목현상을 가져온다.

본 논문에서는 LZSS 알고리즘[2]과 엔트로피 부호를 이용하여 데이터의 정보 손실 없이 표현된 데이터의 비트 수를 줄일 수 있는 방법을 제안하였다. LZSS의 사전탐색처리장치(DISP)[5]는 사전 내용을 조회하여 일치한 최장 데이터 열을 부호화 하여 압축을 실행한다. 입력 데이터는 사전의 내용과 비교되어 일치 기호를 탐색한다. 만약 부호화 된 일치 기호가 입력 데이터 열

* 正會員, 徐羅伐大學 電氣電子電算學部
(School of Electricity, Electronics & Computer Science)
** 從信會員, 蔚山大學校 電氣電子 및 自動化 工學部
(School of Electrical and Automation Engineering)
※ 본 연구는 2000년 정보통신부 정보우수시범학교 지원사업과 산업자원부 반도체설계교육센터(IDEC)의 지원으로 수행되었음.
接受日字: 2000年 7月24日, 수정완료일 :2001年12月18日

보다 더 적은 비트로 표현되었을 때 데이터 압축이 이루어진다. 사전의 내용이 입력 데이터와 좀더 많은 일치를 예측할 수 있다면, 입력 데이터의 최장 일치 길이는 클 것이다. 따라서 LZSS의 사전탐색처리장치[5]로 압축된 부호어 출력에 대한 엔트로피 값을 구하여 부호화하면, 입력 데이터가 LZSS 부호어에 비해 2단계 압축[4]으로 부호화된 비트 수를 더 많이 줄일 수 있다.

사전은 입력 데이터를 예측할 수 있으면 데이터를 압축하는데 부호기의 압축 성능이 향상된다. 본 논문의 사전은 렘펠과 지브에 의해 제안[1]된 변형된 LZSS 알고리즘 방식으로 형성된다. 이 방법은 파일의 기호가 사전에 저장되어 모두 채워지면, 계속되는 입력 데이터의 가장 새로운 기호로 인하여 이전에 저장된 사전의 기호를 지우고 그 장소에 입력 데이터의 새로운 기호가 갱신된다. 사전탐색처리장치의 이와 같은 동작으로 인하여 입력 데이터의 내용을 쉽게 압축시켜 사전에 저장된다. LZSS 부호어 출력은 다시 엔트로피 부호화에 의해 또다시 압축을 수행하여 압축 성능을 향상시킨다.

컴퓨터 시스템은 프로그램 파일, 실행 파일, 비트맵 파일, 테스트 파일 등 많은 데이터 형태를 사용한다. 사전탐색처리장치는 사용될 데이터가 어떤 형태든 적용할 수 있도록 데이터 형태에 따른 입력 데이터의 변화가 필요하다. 제안된 적응 모델은 입력 데이터의 내용을 데이터 형태로 사용함으로써 컴퓨터 네트워크에서 사용되는 여러 가지 데이터 형태를 폭 넓게 사용할 수 있다. 부호기에 의해서 압축된 사전의 내용을 복호기를 사용하여 정확히 복호하여 메시지를 전달한다. 다시 말해서, 입력 데이터로부터 부호화 된 사전 내용은 엔트로피 부호기에 의해 다시 부호화하고 그 부호어가 복호기에 입력되어 원래의 파일을 복호한다. LZSS 부호기의 부호어 출력은 미일치 부호어와 일치 부호어의 두 경우로 나누어 처리한다.

일치 부호어 출력은 입력 파일과 사전의 내용을 비교하여 일치된 기호가 2개 이상 존재하면 발생한다. 여기서 일치 부호어 출력은 세 개의 영역으로 구성되는데 첫 번째 영역의 1비트는 플래그 비트로써 플래그 비트가 '0'이면 일치 부호어를 나타낸다. 두 번째 영역은 일치 오프셋으로 부호기 사전 목록에서 존재하는 최장 일치 길이의 선두 위치를 가리킨다. 세 번째 영역은 일치 길이를 표시하는 것으로 입력 데이터 열이 사전의 내용과 일치된 기호가 몇 개인지를 나타낸다.

미일치 부호어 출력은 입력 파일에 대한 일치가 사전에 없거나 또는 한 개일 때 발생한다. 미일치 부호어 출력은 두 영역으로 구성되는데, 첫 번째 영역은 일치 부호어와 마찬가지로 플래그 비트를 표시하며 이 비트가 '1'이면 미일치 부호어 출력임을 의미한다. 두 번째 영역은 입력 파일로부터 발생한 일치되지 않은 8bit ASCII 기호를 나타낸다.

II. 구 성

LZSS_Entropy는 LZSS 부호에 근거한 2단계 부호화에 의해 뛰어난 압축률을 달성하고 있다. 사전탐색처리장치(DISP)의 *WINDOW* 크기는 2K 바이트 LZSS 부호에 의해 32바이트 이하의 최장 일치 길이와 최장 일치 길이의 선두 위치를 찾아 부호화하고, 또 만일 일치 길이를 찾지 못하면 기호어로 부호화한다. 2단계의 부호화인 엔트로피 부호기는 내부에 2K의 엔트로피 중간 버퍼를 구성하여 LZSS 부호가 이 버퍼에 가득 찬 단계에서 각 기호의 출현 빈도를 조사하여 엔트로피 부호화를 수행한다.

그림 1은 LZSS_Entropy 부호기에 대한 전체 구성도를 나타내었다. 데이터 입력 파일이 사전탐색처리장치(DISP)로 입력되어 LZSS 알고리즘에 의한 1단계 압축 부호어가 생성된다. LZSS 부호어는 일치 부호어와 미일치 부호어로 형성되며 그 각각의 비트 수는 17비트 및 9비트로 이루어져 있다. 이와 같은 각각의 LZSS 부호어는 엔트로피 부호기에 의하여 2단계 압축을 실행하여 최종 출력 엔트로피 부호어를 출력한다.

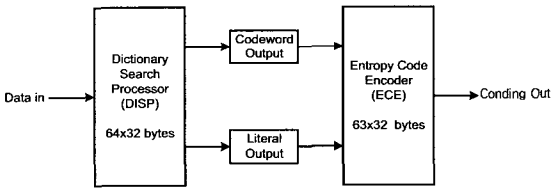

그림 1. LZSS_Entropy 의한 부호화의 구성도
Fig. 1. Block Diagram of Encoding by LZSS_Entropy.

단일-칩 부호기 및 복호기에 대한 사전탐색처리장치(DISP)는 2K 바이트의 메모리[8] 크기를 갖는다. 사용될 사전은 별도의 외부 메모리 없이 내부 메모리만 존재한다. 여기서 사전의 메모리 크기는 압축 성능과 매우 밀접한 관계를 갖는다. 일반적으로 이용할 수 있는

메모리 크기가 크면 클수록 좋은 압축이 얻어진다. 그러나 본 연구에서는 사전의 메모리를 2K 바이트 크기까지는 압축 성능이 증가하고, 그 이상일 때는 압축 성능이 거의 변화가 없이 일정하게 유지하였다. 사전은 *WINDOW*[7]와 *SWR*(Short Window Register)로 구성된 메모리다. *WINDOW*는 2K 메모리로, 8비트로 구성된 기호를 32개 저장할 수 있는 64개의 열을 갖는다. *WINDOW* 번지에서 상위 6비트는 열 번지를 지정하고, 하위 5비트는 행 번지를 지정한다.

EOW 카운터의 값은 열과 행을 갱신할 때 또는 선택할 때 사용한다. 이 것은 11비트 카운터로 구성되며 상위 6비트(REOW)는 열 번지에 할당되고, 하위 5비트(CEOW)는 행 번지에 할당되어 사전의 메모리 번지를 지정한다. 부호화 또는 복호화의 시작은 사전에 초기 신호가 발생해야만 동작한다. 이 때 *EOW* 번지 값을 모두 '0'으로 초기화하고, 사전에 있는 모든 내용을 지운다.

Ⅲ. 부호기

부호기는 소스 기호열(source string)이라 부르는 입력 기호에 대한 일치를 찾는다. 부호기의 사전은 소스 기호열과 사전에 저장된 기호와의 가능한 최장 일치 기호열을 찾는다. 사전탐색처리장치의 모든 *SWR*와 *WINDOW*는 일치 논리 회로와 연결되어, 만약 사전의 내용과 소스 기호열 사이에 일치가 존재한 사전 위치의 출력을 일치 논리 회로의 일치 레지스터로 보내어 일치를 나타낸다.

사전은 소스 기호열의 첫 번째 기호의 일치를 찾는다. 만약 일치를 찾으면, 입력 파일의 두 번째 소스 기호열의 기호와 사전의 있는 기호와 비교하여 연속된 일치 기호를 다시 찾는다. 만약 연속된 일치 기호를 찾으면, 두 번째 기호는 소스 기호열로 추가된다. 그리고 사전은 계속적으로 입력 파일의 다음 기호와 비교하여 더 많은 연속된 일치를 찾아 가능한 최장 일치 기호열에 대한 일치 부호어를 만든다.

부호기가 계속된 기호와 소스 기호열의 연속된 일치를 찾지 못했을 때, 계속된 기호는 다음 소스 기호열의 첫 번째 기호가 되고, 부호기에 의해 발견될 입력 파일의 *EOF*(End-of-File)의 마지막 기호까지 이와 같은 과정을 반복한다. 부호기에서 *EOF* 기호가 나타나면 그 출력은 미일치 부호어가 되고, 입력 파일의 부호화가 완전히 끝난다. LZSS 부호기 부호화에 대한 데이터 경로를 그림 2에 나타내었다.

사전탐색처리장치로 사전에 있는 *SWR*로 이동된 기호나 혹은 *WINDOW*의 내용을 비교하여 일치를 찾는다. 부호기가 일치 기호를 찾을 때 아직 입력되지 않은 기호 보다 앞서 예견하는 능력에 의해서 *SWR*로 이동된 기호의 일치를 찾는다. 사전은 *CAM*(Content Addressable Memory)[8]을 사용하여 각각의 *SWR*와 *WINDOW*를 구성한다. 사전탐색처리장치는 입력 버퍼로부터 전송된 기호와 저장된 기호를 비교한다. 저장

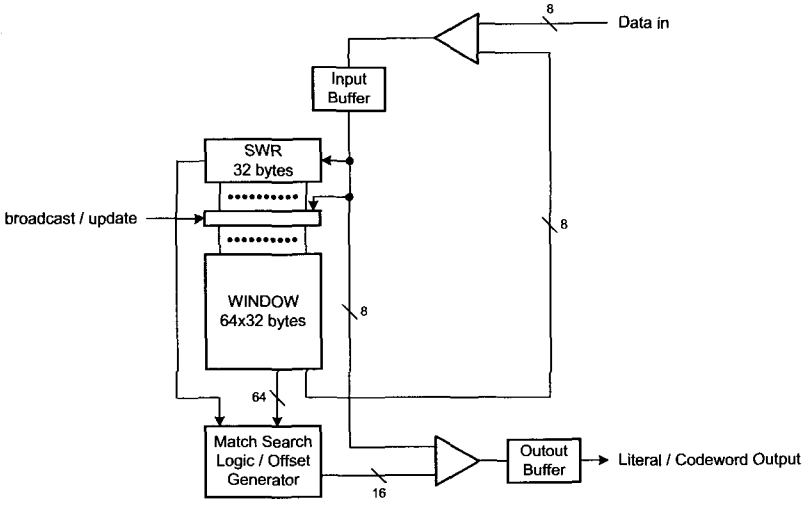

그림 2. LZSS 부호기 데이터 경로
Fig. 2. LZSS Encoder Data Paths.

된 모든 비트는 동적이고, 전송된 기호와 어떤 일치가 발생하지 않았다면 *Hit*라 부르는 비교 회로 출력은 발생하지 않는다. 반대로 저장된 기호가 전송된 기호와 일치하였다면 *Hit* 출력이 발생한다.

Hit 출력은 단지 사전에 저장된 기호가 전송된 기호와 일치인지 아닌지를 나타낸다. 사전에 저장된 기호열이 소스 기호열과 동일하다는 것을 판단하기 위하여 일치 논리 회로가 필요하다. 각각의 일치 논리 회로는 *Hit*, *LastHit* 그리고 *MPropagate*과 같은 세 개의 입력 신호를 갖는다. 또한 일치 논리 회로는 비교된 기호열의 결과 값을 다음과 같은 세 개의 1비트 레지스터에 저장한다. Hit 레지스터(Hit Register)는 CAM의 *Hit* 출력을 저장하고, 만약 CAM에 저장된 기호가 일치 기호열의 가장 새로운 기호이면 일치 레지스터(Match Register)에 '1'을 저장된다. 일치 지연 레지스터(Match Delay Register)는 추가된 클럭 주기 동안 일치 레지스터 출력을 저장한다. 또한, 일치 논리 회로는 *First*와 *Second* 신호와 같은 두 개의 제어 입력을 갖는데, 이와 같은 제어 입력은 초기(first) 상태, 다음(second) 상태, 그리고 지연(propagate) 상태에 대한 세 가지 가능한 상태로 동작한다.

한 개 이상의 기호열 중 첫 번째 전송 기호를 발견하면 초기 상태로 동작한다. 초기 상태는 사전을 초기화시키고, 32개 최장 일치 길이를 갖는다. 초기 상태가 되려면 *First* 신호는 '1', 그리고 *Second* 신호는 '0'을 가져야 한다. 이 경우에 일치 레지스터의 입력은 단지 두 개의 입력 중 하나를 선택하는 2×1 멀티플렉서에 의해 선택된 *Hit* 출력 신호이다.

현재 전송 기호가 일치 기호열이 계속 중이거나 끝났을 때 지연 상태로 동작된다. 다음에 전송된 기호는 일치 기호열일 수도 있고, 일치가 아닌 첫 번째 기호일 수도 있다. 만약 그 기호가 일치 기호가 아니면 다음 기호열의 첫 번째 기호가 된다. 지연 상태에서는 *First*와 *Second* 제어 신호가 모두 '0'이 된다.

전송된 기호가 이전의 일치 기호열의 기호와 비교하여 첫 번째 일치하지 않는 기호를 발견하면 다음 상태가 된다. 이 동작 상태는 어떤 전송된 기호가 한 개 이하의 일치하지 않는 기호가 발생할 때 부호화하는 것을 방지한다. 기호가 전송될 때, 일치 논리 회로는 지연 상태에 있다. 제어 장치는 전송 기호가 현재의 일치 기호열이 아니고, 다음 기호열의 첫 번째 기호인지를 분석하여 결정한다. *Second* 입력이 '1'이고, *First* 입력이 '0'일 때 다음 상태로 동작한다. 만약 양쪽 입력이 '1'이면, 그때 일치된 기호열의 두 번째 기호를 일치 레지스터에 저장한다. 일치 논리 회로는 다음 입력 기호가 전송할 때 지연 상태가 된다. 만약 모든 일치 레지스터의 입력이 '1'을 갖지 않으면, 미일치 부호어 출력이 발생한다. 일치 논리 회로는 지연 상태가 될 때까지 다음 기호 모두에 대하여 다음 상태를 유지한다.

*WINDOW*의 일치 오프셋은 *EOW*부터 탐색된 일치의 위치까지에 대한 간격이다. 일치 오프셋은 세 개의 항으로 구성되어 있는데 첫 번째 항은 *EOW*부터 *EOW* 열의 시작점까지에 대한 간격이다. 이 간격은 CEOW 값(*EOW* 하위 5비트)과 같다. 두 번째 항은 탐색된 일치가 있는 열과 *EOW* 사이의 열 간격이다. 우선 순위 열 번지는 탐색된 일치가 있는 열 번지와 *EOW* 열 번지의 사이의 번지와 같다. 따라서 이 항의 일치 오프셋의 구성은 각각의 열(32) 위치에 우선 순위 열 번지를 곱한 것이다. 세 번째 항은 탐색된 일치를 갖는 열의 마지막부터 일치까지의 간격인 우선 순위 행 번지와 1을 더한 값이다. 따라서 *WINDOW*에 대한 일치 오프셋은 식 (1)과 같이 계산된다.

$$WINDOW\ Match\ Offset = CEOW+(32\times RBUS) + (CBUS+1) \quad (1)$$

또한, 일치가 *SWR*에 있다면, 일치 오프셋은 탐색된 일치 위치로부터 *SWR* 열의 마지막까지의 간격이다. 따라서 *SWR* 일치 오프셋 계산은 총 행수 32에서 탐색된 일치 행 번지 수를 뺀 것이다. 우선 순위 행 번지는 31에서 탐색된 일치 행 번지를 빼야하므로 *SWR*에 대한 일치 오프셋은 다음 식 (2)와 같이 계산된다.

$$SWR\ Match\ Offset = CBUS + 1 \quad (2)$$

그림 3은 부호기 실행을 위한 제어 장치의 동작 흐름을 나타낸 것이다.

부호기 제어 장치는 기호 일치, 열 번지 부호화, 행 번지 부호화, 그리고 오프셋 발생을 위해 네 개의 경로(pipelines)[6]를 사용한다. 병렬 처리 동작은 일정한 입력 비율로 부호기가 동작할 때, 일치부호어(codeword)와 미일치부호어(literal)의 출력이 발생된다. 부호기는 처음 32개 클럭 주기 각각에 대해 하나의 기호가 입력되고, 그리고 33번째 클럭은 윈도우의 열 갱신에 사용되어진다.

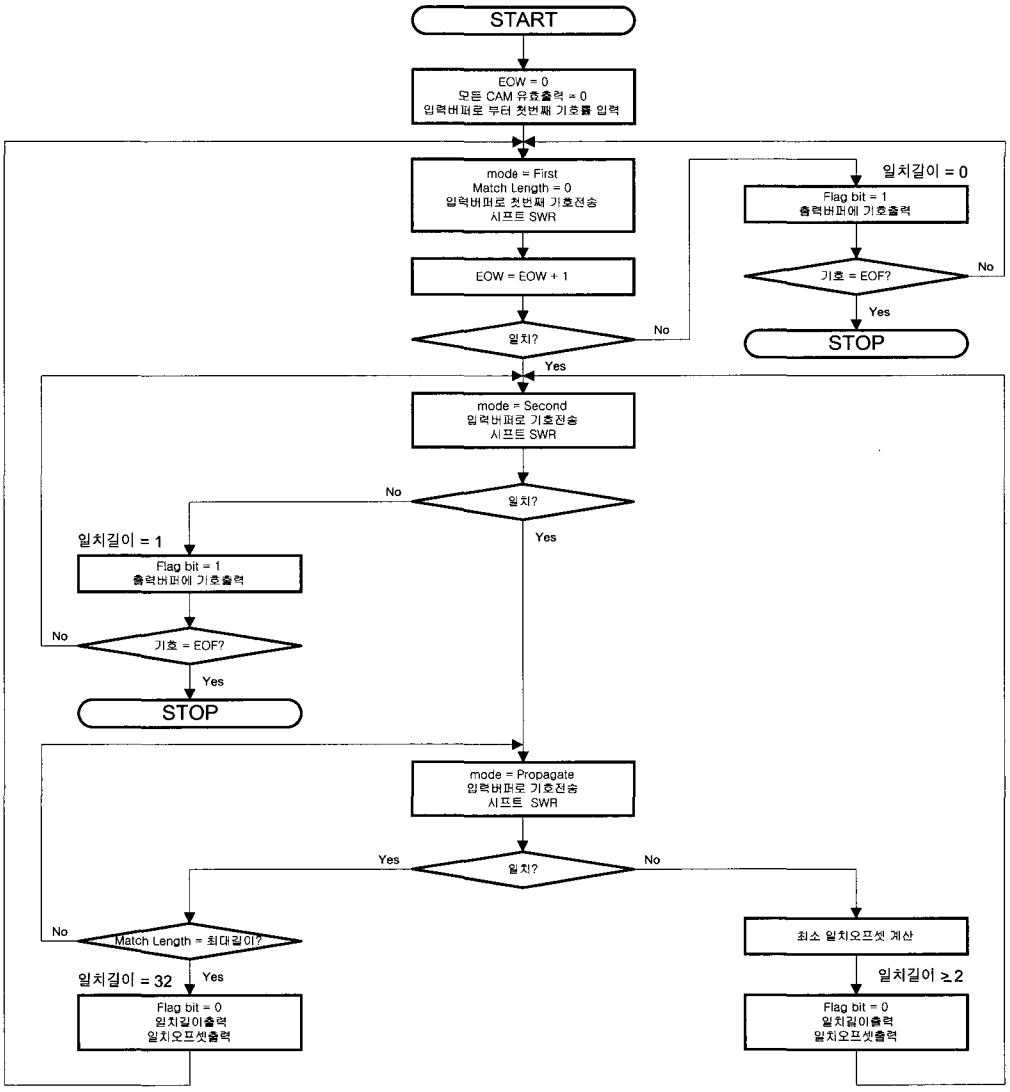

그림 3. 부호기 제어 장치 흐름도
Fig. 3. Encoder Controller Flow Diagram.

Ⅳ. 복호기

복호기는 부호기와 같은 방식으로 복호기의 사전을 구성한다. 이것은 부호화된 기호가 계속적인 이동과 탐색을 위해 32개 기호 시프트 레지스터인 *SWR*와 2K 바이트의 SRAM 방식으로 *WINDOW* 기억 장치를 사용한다. 플래그 비트는 미일치 부호어 또는 일치 부호어 인지를 판별한 후 첫 번째 플래그 비트를 없애고 복호기 입력으로 보낸다.

만약 플래그 비트가 '1'일때, 부호어 입력은 미일치 부호어(9비트)로서, 플래그 비트를 제외한 나머지 8비트는 ASCII 기호를 나타낸다. ASCII 기호는 출력 버퍼로 입력되고 복호기의 사전에 새로운 기호로 멀티플렉서에 의해 선택되어 *SWR*의 31행 번지로 시프트된다. 이것은 부호기와 동일한 방법으로 *SWR*에 각각의 기호를 시프트함으로써 사전을 갱신한다. 그림 4는 복호기에 의한 *SWR*의 갱신 데이터 경로를 나타낸다.

만약 플래그 비트가 '0'이면, 부호기의 입력은 일치 오프셋과 일치 길이를 갖는 일치 부호어(17비트)이다. 여기서 일치 오프셋 비트는 일치 오프셋 레지스터에 보내어 일치 오프셋 값이 *SWR*에서 일치를 찾을 수 있는 최대 오프셋 값 32가 되는지를 비교한다. 만약 일

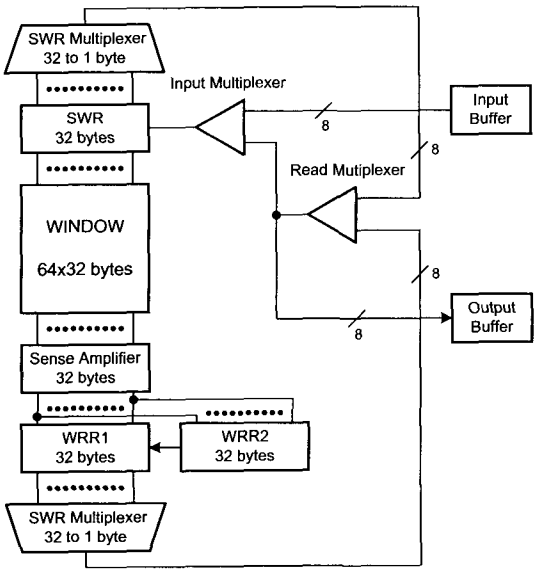

그림 4. 복호기 갱신 데이터 경로
Fig. 4. Decoder Update Data Paths.

치 오프셋이 32 보다 작거나 같으면, 사전으로부터 읽은 기호는 *SWR*에서 입력된 것이다. 다시 말해서, 처음 복호화된 기호열의 기호는 *WINDOW* 기억 장치로 보내진다. 이것의 논리적인 번지는 일치 오프셋 -1-CEOW이다. 대응하는 물리적 번지는 아래와 같다.

$$physical\ row\ address = (63 - logical\ row\ address + \mathrm{REOW}) \bmod 64 \quad (3)$$

$$physical\ column\ address = the\ complement\ of\ logical\ column\ address \quad (4)$$

첫 번째 기호열이 기억 장치에 보내지면, 일치 길이의 값에 의해서 결정되어진 일치 기호들의 수만큼 *SWR*로 이동되어 사전에 보내진다. 만약 일치 오프셋이 32 보다 크면 사전으로부터 읽은 기호는 *WINDOW*에서 입력된 것이다.

SWR 기호 판독 및 복호는, 만약 일치 오프셋이 32 보다 작거나 같으면, 그 기호는 *SWR*에서 찾은 사전으로부터 읽은 것이다. *SWR* 멀티플렉서에 의해서 *SWR*에 32개 이하의 기호를 입력한다. *SWR* 멀티플렉서는 각각 8비트를 갖는 32×1 멀티플렉서로 구성된다. 일치 오프셋의 값은 *SWR* 멀티플렉서에 의해 다음과 같은 (5)식으로 선택할 기호의 행을 결정한다.

$$\text{Selected } SWR \text{ Column} = 32 - \text{Match offset} \quad (5)$$

SWR 멀티플렉서의 출력은 판독 멀티플렉서(Read Multiplexer)로 입력된다. 판독 멀티플렉서는 각각 8비트를 갖는 2×1 멀티플렉서로 구성된다. 판독 멀티플렉서는 *SWR*로 이동할 다음 기호가 *WINDOW*로부터 또는 *SWR*로부터 읽을 것인지를 선택하는데 사용된다. 판독 멀티플렉서는 일치 오프셋이 32 보다 작거나 같으면 *SWR* 멀티플렉서로부터 기호를 선택한다.

판독 멀티플렉서의 출력 기호는 복호기의 출력 버퍼로 보내진다. 또한 이 기호는 복호기 사전에 가장 새로운 기호가 되어 *SWR*의 31번째 행으로 시프트된다. 일치 기호가 *SWR*로 시프트될 때 *SWR*의 모든 기호가 한 비트씩 시프트된다. 그러므로, 첫 번째 일치 기호를 갖고있는 이전의 *SWR* 위치는 다음 일치 기호가 입력될 것이다. 다음 기호를 *SWR*로 시프트할 때는 *SWR* 멀티플렉서에 선택된 *SWR* 행은 변화지 않는다.

*WINDOW*의 기호 판독 및 복호는, 일치 기호를 *WINDOW*로부터 읽어 들여, *EOW* 값에서 일치 오프셋 값을 뺀 결과가 *WINDOW*에 탐색된 일치의 첫 번째 기호에 대한 번지가 된다. 만약 일치 오프셋이 *EOW* 보다 크면, 탐색된 일치의 첫 번째 기호 번지를 나타내기 위해서는 *EOW* 값에 일치 오프셋을 뺀 결과에 2048을 더한다. 이 기호가 위치한 모든 *WINDOW* 열은 데이터 버스로 전송된다. 데이터 버스로 전송된 기호는 32개의 감지 증폭기(Sense Amplifier)로 보내진다. 각각의 감지 증폭기는 8비트 폭을 갖는다. 다음 (6)식은 선택할 *WINDOW* 번지의 계산식을 나타낸 것이다.

$$\text{Selected } WINDOW \text{ Address} = \begin{cases} EOW - \text{Match offset} & (EOW \geq \text{Match offset}) \\ EOW - \text{Match offset} + 2048 & (EOW < \text{Match offset}) \end{cases} \quad (6)$$

최대 일치 길이는 32개고, 그리고 각각의 열의 행 또한 32개다. 그러므로 모든 일치가 *WINDOW*의 한 열 안에 존재할 수도 있고, 또는 *WINDOW*의 다음 열까지 연장되어 존재할 수도 있다.

그림 5는 복호기 실행에 따른 제어 장치(controller)의 동작 흐름을 나타낸 것이다.

복호기 제어 장치는 부호기 제어 장치와 달리, 플래그 비트가 '0' 또는 '1'이냐에 따라 동작 상태가 다르다. 플래그 비트가 '0'이면, 미일치 부호어의 8비트 ASCII 기호를 출력 버퍼로 보내고, 만약 플래그 비트가 '1'이

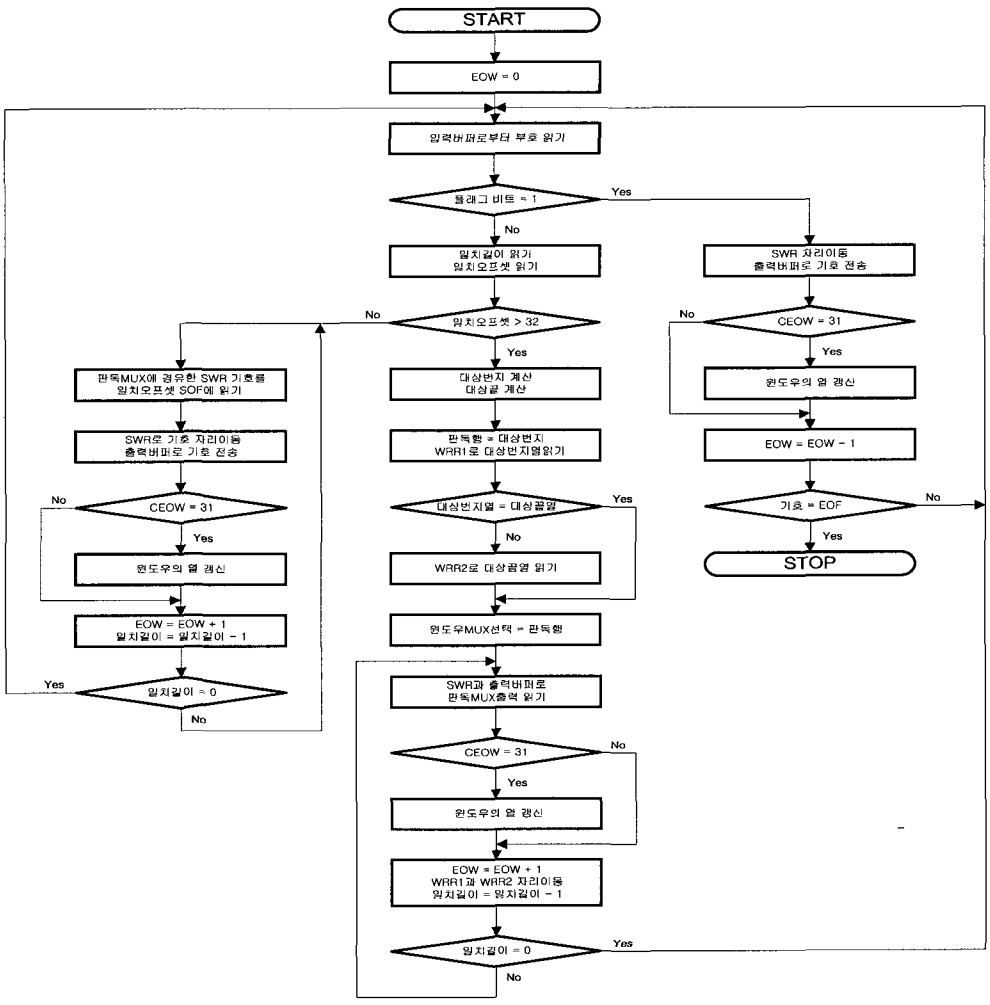

그림 5. 복호기 흐름도
Fig. 5. Decoder Flow Diagram.

면, 일치 부호어를 복호하여 출력 버퍼로 보낸다.

V. 엔트로피 부호화

LZSS 부호어 출력에 향상된 엔트로피 부호화를 적용하면 부호기의 압축률을 증진시킬 수 있다. LZSS 부호기 출력으로부터 얻어진 부호어의 일치 길이와 일치 오프셋 구성 요소에 대한 확률 분포를 계산한다. 그래서 LZSS 알고리즘에 의하여 압축된 비트의 일치 길이와 일치 오프셋을 엔트로피를 적용하여 부호화하면 메시지에 필요한 비트의 수를 더 줄일 수 있다. 여기서 엔트로피는 메시지를 포함하는 평균 정보량을 측정한다. 이것은 메시지의 발생 확률로 정해진다. 메시지는 적은 연관성을 갖는 정보가 높은 발생 확률을 갖는다. 예를 들어, 만약 특정한 메시지에 대해 일정한 절대치가 발생한다면, 정보의 발생 확률에 대한 이득이 없다. 단지 일정하지 않은 메시지가 어떤 비트에 대해 상당한 발생 확률을 갖는다. 따라서 기호에 대한 확률을 $P(i)$로 놓으면 기호 i를 갖는 정보량이 $\log_2 P(i)$로 주어지고, 정보원으로부터 출력되는 한 개의 기호당 정보량의 기대치, 즉

$$E = -\sum_i P(i)\log_2 P(i) \tag{7}$$

로 정의하고, 이것을 평균 정보량 또는 엔트로피라 한다. 기호 i에 대응하는 부호어의 길이에는 i가 가진 정보량을 운반하는 데 충분한 비트 수가 필요하므로 엔트로피는 한 개의 기호를 부호화 하는데 필요한 최소 평균 부호 길이를 부여하고 있다. 평균 부호 길이를

L이라 하면 $E \leq L$이 성립된다. 이 관계식은 어떤 이상적인 부호화를 수행하여도 평균 부호 길이를 엔트로피보다 적게 하는 것은 불가능하다는 것을 의미한다. 만약 파일에 대한 일치 길이와 일치 오프셋 값의 확률이 계산되기 전에 더해지면, 큰 파일은 일치 길이와 일치 오프셋 부호어의 가중치가 더 커질 것이다.

테스트 파일의 일치 길이와 일치 오프셋의 출현 빈도를 사용하여 확률 분포를 구한다. 그리고 테스트 파일에 대한 각각의 일치 길이와 일치 오프셋의 확률로 정보량을 계산한다. 일치 길이와 일치 오프셋은 독립적으로 각각 다르게 부호화된 엔트로피가 존재한다. LZSS의 일치 길이와 일치 오프셋 부호에서도 기호열의 길이가 길어짐에 따라 평균 부호 길이가 엔트로피에 수렴하는 이상적인 부호화가 수행된다.

엔트로피 공식 (7)을 이용하여 일치 길이의 발생 확률에 따른 일치 길이의 평균 엔트로피를 계산하면 2.86이 구해진다. 이것은 각각의 일치 길이를 평균하여 부호화하지 않은 일치 길이의 5비트에 대한 평균 일치 길이의 엔트로피 수를 나타낸 것이다. 그러나, 여기서 각각의 일치 길이는 비트의 정수에 의해 만들어지기 때문에 제시된 최적의 일치 길이는 실현될 수 없다. 예를 들어, 테스트 파일의 압축 결과로부터 일치 길이가 2인 발생 확률은 0.4005이다. 이것에 대한 엔트로피 값은 1.32가 되어 1비트 또는 2비트 어느 것이든 표현할 수 있다. 그러나 1비트를 사용하면 2비트 보다 다른 일치 길이가 엔트로피 보다 더 많은 비트 수를 갖을 수 있다. 그러므로 일치 길이의 평균 산출은 계산된 2.8의 엔트로피 값보다 더 길어야만 한다.

엔트로피 일치 길이 부호어는 각각의 LZSS 일치 길이 부호에 적용된 엔트로피 평균 정보량의 비트를 결정하여 할당한다. 이것은 평균적인 일치 길이에 나타나는 출현 빈도를 이용하는 것으로 실제로 부호화를 행하는 부호열에 의존하지 않고, 몇몇의 기호에 대해서는 짧은 비트 길이를 할당하였다. 결과적으로, 이 부호열 중에서 가장 출현 빈도 횟수가 많은 부호부터 순서대로 짧은 비트 길이를 할당하여 엔트로피 일치 길이 부호어에 대한 결과를 표 1에 나타내었다.

일치 길이의 엔트로피 부호화에 사용된 같은 방법으로 일치 오프셋의 엔트로피 부호화를 구한다. 일치 길이의 가능한 최대 값은 31이었고, 일치 오프셋의 가능한 최대 값은 2047이다. 일치 오프셋의 가능한 값이 커지면 일치 오프셋의 부호화가 더 복잡해진다. 따라서 부호화를 간단히 하기 위해, 일치 오프셋의 값을 네 개의 그룹으로 나누었다. 그룹_0은 0~15까지이고, 그룹_1은 16~143까지의 일치 오프셋을 갖는다. 또한 그룹_2는 144~1167까지이고, 마지막 그룹_3은 1168~2047까지의 일치 오프셋으로 나누었다.

표 1. 일치 길이 엔트로피 부호화
Table 1. Match Length Entropy Coding.

Match length value	group	Number of code bits	Coded length
2~3	0	2	1X
4~7	1	4	01XX
8~15	2	6	001XXX
16~23	3	7	0001XXX
24~27	4	7	00001XX
28~31	5	8	000000XX
32	6	6	000001

각각의 그룹에 발생할 수 있는 엔트로피 값을 계산하여 평균을 취한다. 그리고 계산된 그룹의 평균값을 다른 그룹과 비교하여 근사한 값들의 그룹을 다시 일치 오프셋 그룹으로 재편성한다. 재편성한 일치 오프셋 그룹은 표 2와 같이 네 개의 그룹으로 구성된다. 이 일치 오프셋 그룹에 포함된 그룹들의 평균 최적 비트 값의 확률에 대한 엔트로피를 구하여 일치 오프셋 그룹에 대한 부호어 비트 수를 결정한다. 그리고 부호어 비트의 수만큼 대응되는 일치 오프셋 부호어를 설정한다. 표 2는 LZSS 부호어에 엔트로피를 적용하여 그룹별 일치 오프셋 값에 대한 부호어 비트 수와 엔트로피 일치 오프셋 부호어의 최종 결과 값을 나타내었다.

표 2. 일치 오프셋 엔트로피 부호화
Table 2. Match Offset Entropy Coding.

Match offset value	group	Number of code bits	Coded Offset
0~15	0	7	000XXXX
16~143	1	9	01XXXXXXX
144~1167	2	11	1XXXXXXXXXX
1168~2047	3	13	001XXXXXXXXXX

VI. 결과 및 평가

본 논문은 LZSS 알고리즘과 엔트로피 부호화를 이용하여 사전탐색처리장치(DISP)로 부호 및 복호를 실행할 수 있는 VLSI 단일 칩을 설계하였다. 메모리는 사전 내부에 저장 용량은 2048바이트 크기를 갖고, 추

가된 외부 기억 장치는 없다. 메모리 크기 변화[9]에 따른 LZSS 부호기와 LZSS_Entropy의 압축률이 어떻게 변화하지를 그림 6의 그래프에 나타내었다. 이 그림으로 알 수 있듯이 양쪽 모두가 메모리 크기가 커짐에 따라 좋은 압축률을 갖는다. 그러나, 메모리 크기가 2K 이상이 되면 압축 성능이 현저히 떨어지는 것을 알 수 있다. 따라서, 엔트로피 부호기로 결합되어진 2K 사전 메모리 크기는 압축률과 칩 면적 사이에 알맞은 균형을 이루었다. 또한 엔트로피가 적용된 부호와 LZSS 부호를 비교하면 엔트로피가 적용된 부호가 약 절반 이하의 메모리로 LZSS 부호와 동등한 압축률을 보이고 있다.

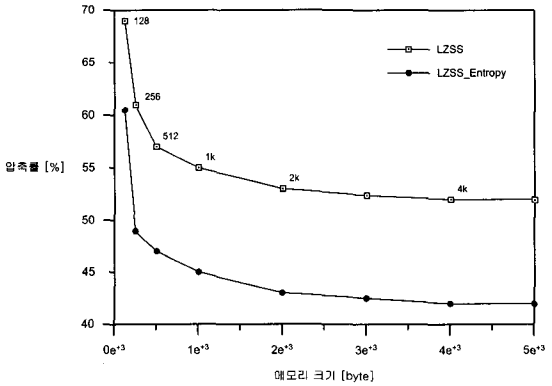

그림 6. 메모리 크기와 압축률의 관계
Fig. 6. Relation of the Compression Ratio and Memory Size.

0.6μm CMOS로 구현된 LZSS_Entropy는 처리 속도 50MHz를 갖는다. 클럭 주기마다 32개의 기호를 입력한다. 첫 번째부터 32번째까지의 클럭은 32개 기호에 대한 각각의 클럭이고, 33번째 클럭 주기는 *WINDOW*의 열 설정으로 사용된다. 따라서 처리 속도[character/cycle]는 50MHz 주기마다 한 개의 기호를 처리한다. 처리 시간과 압축률의 관계에 있어서 일반적으로 압축률이 좋아짐에 따라 처리 시간이 길어지는 관계가 있다. 그래서 엔트로피 부호기를 LZSS에 적용할 경우 부호 및 복호 속도가 엔트로피 부호화 속도만큼 떨어지는 경향이 있다. 이와 같은 문제를 해결하기 위해 사전탐색처리장치에 트리(tree) 구조를 이용하여 최장 일치 기호열의 탐색을 고속화하고, 역변환표를 준비하여 부호 트리 대응 절점을 구함으로써 복호화를 고속화한다. 테스트 파일(printers.txt)를 부호화 및 복호화 하는데 걸린 시간을 표 3에 나타내었다.

표 3. 부호화 및 복호화 처리 시간
Table 3. Processing Time for Encoding and Decoding.

Code	Encoding time[sec]	Decoding time[sec]
LZSS	80.6	5.6
LZSS_Entropy	84.4	8.9

표 4는 입력 테스트 파일 크기에 대한 미일치 부호어와 일치 부호어의 출력 비트 수를 구하여 압축률을 나타내었다. 이 것은 LZSS 부호에 의해 부호화된 출력 부호어로 엔트로피 부호화가 적용되지 않은 압축률이다. 표 4와 같이 LZSS 설계에 대한 압축 결과는 총 289K 바이트 크기의 실험 파일에 대하여 42%부터 79%까지의 압축 범위를 갖는다.

표 4. LZSS 압축 결과
Table 4. LZSS Compression Results.

Test file	Test file size[bit]	literal output[bit]	Codeword output[bit]	Compression ratio[%]
A	224496	32490	84490	52
B	896944	29736	368798	44
C	167256	13923	60401	44
D	141112	53604	58667	79
E	102776	13896	49062	61
F	235056	49248	110551	51
G	185672	45774	76211	65
H	193688	14337	67269	42
I	164464	10512	66300	46

그리고 향상된 압축 성능을 얻기 위하여 LZSS 부호어 출력에 휴프먼(Huffman) 부호[3]와 같은 엔트로피 부호를 적용하여 약 7% 정도 압축률을 증진시킬 수 있었다. 엔트로피 부호는 LZSS 부호 출력 값을 복잡한 여분의 블록을 사용하여 엔트로피 부호화에 대한 압축률을 시뮬레이션하였다. 표 5는 LZSS 압축률과,

표 5. 엔트로피 부호기에 의한 LZSS 압축률 개선
Table 5. Improvement LZSS Compression Ratio by Entropy Coder.

Test file	LZSS compression ratio [%]	Entropy compression ratio [%]
A	52	46
B	44	39
C	44	39
D	79	71
E	61	49
F	51	41
G	65	57
H	42	36
I	46	41
AVE	53	46

그리고 LZSS 부호어 출력에 엔트로피 부호를 적용한 압축률을 비교한 것이다. 이 표에서 엔트로피 부호를 적용한 LZSS가 개선된 압축률을 나타냄을 알 수 있다.

그림 7은 엔트로피를 적용한 부호와 LZSS 부호에 대해 압축 성능을 비교하여 그 결과를 그래프로 나타내고 있다. 그래프 결과로 알 수 있듯이, 레이저 프린트 활자 파일(.sfp)의 압축률이 매우 좋음을 알 수 있고, 실행 파일(.com)의 압축률이 낮음을 알 수 있다. 모든 테스트 파일에 대해서 엔트로피를 적용한 부호가 적용하지 않은 LZSS 부호보다 좋은 압축 성능을 보이고 있다. 평균적으로는 양쪽 모두가 50% 정도의 압축률을 보이고 있다.

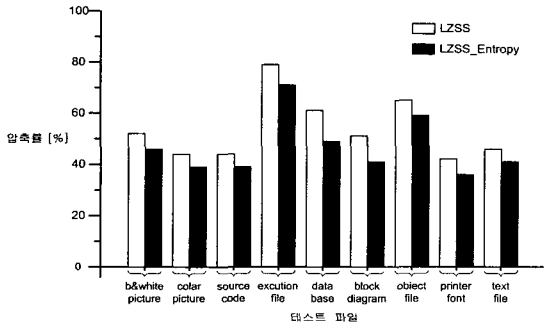

그림 7. 테스트 파일의 압축률 비교
Fig. 7. Comparison to The Compression Ratio of Test Files.

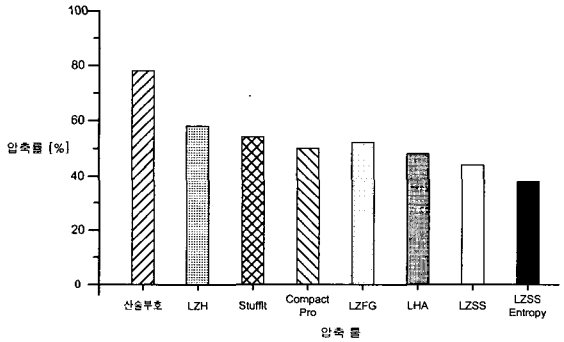

그림 8. LZSS_Entropy와 다른 압축 툴에 대한 압축률 비교
Fig. 8. Comparison of Compression Ratio between LZSS_Entropy and Another Compression Tools.

그림 8은 압축 툴의 압축률을 비교하여 그래프로 나타낸 것이다. 여기서 동일한 테스트 파일을 사용하지 않아 비교하는데 다소 무리가 있지만, 가능한 비교적 같은 테스트 파일의 크기와 종류를 사용하여 압축률을 제시하였다. 본 논문의 LZSS와 LZSS_Entropy에 대한 테스트 파일은 grdemo.c 프로그램 데이터의 20,907바이트를 사용하였으며, 다른 비교 압축 툴에 대한 테스트 파일[10]은 목적 코드 프로그램 데이터의 21,504바이트에 대한 압축률 수치를 나타낸 것이다. 그림 8에도 알 수 있듯이, 다른 압축 툴에 비해 상당히 좋은 성능을 보이고 있다.

전체 칩 평면도를 그림 9에 나타내었다. 여기서 다른 모듈 구성들에 대한 면적이 2K의 CAM 보다 작아 전체 칩 크기는 *WINDOW* 메모리 크기에 달려있다. 왜냐하면 CAM 셀의 트랜지스터 개수가 많기 때문이다. 전체 칩에 대한 트랜지스터를 살펴보면, 사전의 트랜지스터 개수가 약 16만개에 이르며, 이것은 전체 트랜지스터 개수 약 50만개의 삼분의 일을 차지한다.

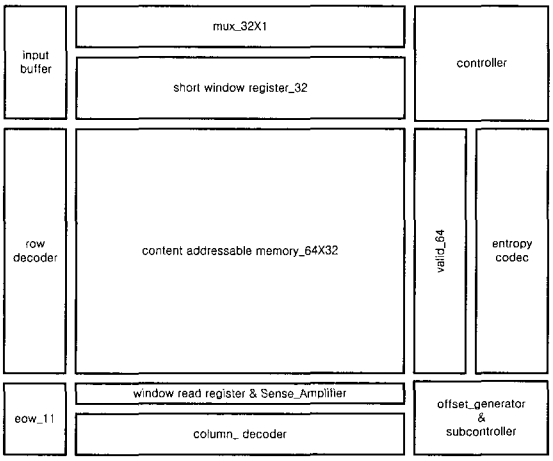

그림 9. LZSS_Entropy 칩 평면도
Fig. 9. LZSS_Entropy Chip Floorplan.

VI. 결 론

0.6μm CMOS 기술로 구현된 LZSS_Entropy 부호기/복호기는 처리 속도 20ns를 갖는다. 구현된 단일-칩의 성능은 약 289KB의 테스트 파일에 대하여 36%부터 71%의 범위의 압축률을 나타내었다. 이 압축률은 테스트 파일 종류와 크기에 따라서 차이가 많이 있다. 그리고 LZSS 부호기에 비하여 엔트로피 부호를 적용한 LZSS_Entropy 부호기는 테스트 파일에 대해 최소 5%부터 최대 12%까지 압축 성능을 개선할 수 있었다. 전체적으로는 평균 46%의 압축률과 알맞은 사전의

WINDOW 크기와 결합된 LZSS_Entropy 부호기는 전체 칩 크기와 균형을 이루었다.

참 고 문 헌

[1] J. Ziv, and A Lempel, "A universal algorithm for sequential data compression", *IEEE Trans. on Information Theory*, IT-23(3), 337-343. May 1977.

[2] J. A. Storer, and T. G. Szymanski, "Data compression via textual substitution", *J. Association for Computing Machinery*, 29(4), 928-951. October 1982.

[3] H. Yokoo, "An Improvement of Dynamic Huffman Coding with a Simple Repetition Finger", *IEEE Trans. Communication Theory*, vol. 39, no. 1, pp. 8-10, January 1991.

[4] H. Yokoo, "Improved Variation Relating the Ziv-Lempel and Welch-Type Algorithms for Sequential Data Compression", *IEEE Trans. on Information Theory*, vol. 38, no. 1, pp. 73-81, January 1992.

[5] M. Motomura et al., "A 1,2-Million Transistor, 33 MHz, 20-bit Dictionary Search Processor (DISP) ULSI with a 160-kb CAM", *IEEE J. Solid State Circuits*, vol. 25, no. 5, pp. 1158-1165, October 1900.

[6] Lawrence T. Clark and Robert O. Grondin, "A Pipelined Associtive Memory Implemented in VLSI", *IEEE J. Solid State Circuits*, vol. 24, no. 1, pp. 18-28, February 1989.

[7] S. Bunton and G. Borriello, "Practical Dictionary Management for Hardware Data Compression", University of Washington, 1991.

[8] H. Kadota et al, "An 8-bit Content Addressable and Reentrant Memory", *IEEE J. Solid State Circuits*, vol. SC-20, no. 5, October 1985.

[9] H. Morita and K. Kobayashi, "On Asymptotic Optimality of a Sliding Window Variation of Lempel-Ziv Codes", *IEEE Trans. on Information Theory*, vol IT-39, no. 6, pp 1840-1846, November 1993.

[10] Timothy C. Bell, John G. Cleary and H. Witten, "Text Compression", Prentice Hall, New Jersey, 1990.

저 자 소 개

金 鍾 燮(正會員)

1961년 6월 28일생. 1990년 2월 숭실대학교 전자공학과 졸업(공학사). 1992년 1월 San Jose State Univ. 대학원 전자공학과 졸업(공학석사). 1997년 2월 울산대학교 대학원 전자공학과 졸업(공학박사). 1993년 3월~현재 서라벌대학 전기전자전산학부 전임강사. 주관심분야는 VLSI & CAD 설계, 컴퓨터 및 데이터 통신

趙 相 福(從信會員)

1955년 6월 10일생. 1979년 2월 한양대학교 전자공학과 졸업(공학사). 1981년 2월 동 대학원 전자공학과 졸업(공학석사). 1985년 2월 동 대학원 전자공학과 졸업(공학박사). 1994년 8월~1995년 8월 Univ. of Texas, Austin 초빙학자. 1986년 3월~현재 울산대학교 전기전자 및 자동화 공학부 교수. 자동차전자 연구센터 소장. 주관심분야는 ASIC 설계, 자동차 전자회로 설계, 비젼 시스템 개발, 테스트 및 테스트 용이한 설계 등임