論文2000-37TE-3-6

실시간 데이터 압축을 위한 Lempel-Ziv 압축기의 효과적인 구조의 제안

(An efficient Hardware Architecture of Lempel-Ziv Compressor for Real Time Data Compression)

陳 庸 先 *, 鄭 正 和 **

(Yong-Sun Jin and Jong-Wha Chong)

요 약

본 논문에서는 실시간 데이터 압축을 위한 Lempel-Ziv 압축기의 효과적인 하드웨어 구조를 제안한다. 일반적으로 Lempel-Ziv 알고리즘의 구현에서는 matching 바이트 탐색과 dictionary 버퍼의 누적된 shift 동작이 처리 속도에 가장 중요한 문제이다. 제안하는 구조에서는 dictionary 크기를 최적화 하는 방법과 복수개의 바이트를 동시에 비교하는 matching 바이트 처리 방법, 그리고 회전 FIFO 구조를 이용하여 shift 동작 제어 방법을 이용함으로써 효과적인 Lempel-Ziv 알고리즘의 처리 구조를 제안하였다. 제안된 구조는 상용 DSP를 사용하여 하드웨어적으로 정확하게 동작함을 검증하였으며, VHDL로 기술한 후 회로 합성을 수행하여 상용 FPGA 칩에 구현하였다. 제안된 구조는 시스템 클락 33Mhz, 비트율 256Kbps 전용선에서 오류 없이 동작함을 확인하였다.

Abstract

In this paper, an efficient hardware architecture of Lempel-Ziv compressor for real time data compression is proposed. The accumulated shift operations in the Lempel-Ziv algorithm are the major problem, because many shift operations are needed to prepare a dictionary buffer and matching symbols. A new efficient architecture for the fast processing of Lempel-Ziv algorithm is presented in this paper. In this architecture, the optimization technique for dictionary size, a new comparing method of multi symbol and a rotational FIFO structure are used to control shift operations easily. For the functional verification, this architecture was modeled by C programming language, and its operation was verified by running on commercial DSP processor. Also, the design of overall architecture in VHDL was synthesized on commercial FPGA chip. The result of critical path analysis shows that this architecture runs well at the input bit rate of 256Kbps with 33Mhz clock frequency.

I. 서 론

개인용 컴퓨터의 대중화, 디지털 전송 기술의 발전, 고화질 디스플레이 장치 실현, CD 이용의 보편화 및 메모리 장치의 저 가격화 등의 기술 혁신에 따라 정보 전달의 개념은 영상, 오디오, 고용량의 텍스트로 이루어지는 멀티미디어라는 정보 매체를 중심으로 급속히 변화하고 있다. 1980년대 이후 관련기술의 발전으로 영상 및 오디오, 고용량의 텍스트 데이터의 전송에 관련된 서비스는 다양화, 고속화를 요구하게 되었으며 이에 발맞추어 데이터의 효율적 저장 및 전송을 위한 새로운 방식이 필요하게 되었다. 이러한 환경변화 때문에 데이터 압축기술이 데이터 전송과 저장에 중요한 기술로

* 正會員, 京文大學 情報通信科
(Dept. of Information & Communication, Kyung Moon College)
** 正會員, 漢陽大學校 電子工學科
(Dept. of Electronics Eng., Hanyang University)
接受日字:2000年8月7日, 수정완료일:2000年8月24日

등장하였다. 데이터 압축은 손실압축과 무손실 압축으로 구분된다. 실시간 데이터 통신을 위하여는 무손실 압축이 필요하다. 무손실 데이터 압축은 주어진 데이터 셋을 다시 복원할 수 있는 새로운 작은 데이터 셋으로 변환하는 것이다. 즉, 전송될 임의의 데이터의 양을 1/2 크기 정도로 줄여주게 된다. 이것은 결국 통신채널의 사용할 수 있는 통신대역폭을 확대시킴으로써 전체적인 네트워크의 성능을 개선시킬 수 있다. 이러한 이유로 인하여 효과적인 압축방법이 요구되었으며 많은 무손실 압축 알고리즘이 제안되었다. 예를 들면 Huffman coding, Run length coding, Arithmetic coding, Lempel-Ziv 알고리즘[1][2] 등이다. Lempel-Ziv 데이터 압축 시스템[3][4]은 전자메일 파일, 포스트스크립트 파일, 문자화일, 그림화일등을 전송하거나 저장하기 전에 압축시킬 수 있다. Lempel-Ziv 알고리즘의 중요한 장점은 입력 데이터가 가지고 있는 사전 지식이나 통계치 정보가 없이 보다 좋은 압축 율을 제공한다는 것이다. Lempel-Ziv 알고리즘을 위하여 여러 가지 특수목적의 하드웨어 구조가 제안되었다.[5]-[8] 그러나 대부분의 구조가 시스톨릭 어레이 구조로 되어있어 이것을 특수목적의 응용분야에 적용하기 위하여 FPGA 등에서 동작시키고자 할 때에는 구조상 회로소자의 크기가 크기 때문에 직접적용하기가 어렵다. 따라서 본 논문에서는 Lempel-Ziv 알고리즘의 처리방법을 개선하여 하드웨어의 복잡도를 감소시키는 효과적인 Lempel-Ziv 압축기의 구조를 제안한다. 제안하는 압축기 구조는 dictionary 크기를 최적화 하는 방법과, matching byte 탐색의 개선방법, 그리고 dictionary buffer를 갱신하기 위한 효율적인 shift 동작 제어방식을 제안한다.

Ⅱ. 기존 lempel-Ziv 알고리즘

Lempel-Ziv 압축 알고리즘은 커다란 데이터 패켓을 작은 데이터 패켓으로 대체 하는 것이다. 그 동작 흐름은 1차적으로 앞서 발생하였던 독립적인 부분심벌들을 dictionary 에 모으고 2차적으로 현재 들어오고 있는 입력심벌을 앞서 발생하였던 심벌들과 matching을 조사하여 현재입력 심벌을 그대로 데이터 패켓으로 만드는 대신에 앞서 발생한 심벌의 시작위치와 matching 길이를 가지고 encoding된 데이터 패켓을 만든다. 이것을 그림 1에 나타내었다.

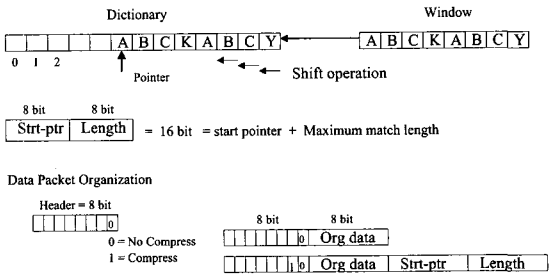


그림 1. Lempel-Ziv 알고리즘의 데이터 처리
Fig. 1. Data process of Lempel-Ziv algorithm.

Lempel-Ziv 알고리즘의 처리는 2단계로 이루어진다. 첫째는 M개의 입력 심벌을 encoding 하는 것이고 둘째는 encoding 단계에서 검색된 최대 matching 길이만큼 dictionary에 있는 심벌들을 왼쪽으로 shift 시키는 것이다. Lempel-Ziv 압축절차에 대한 pseudo-code를 그림 2에 나타내었다.

```
pointer = max_length = 0;
for i from 1 to N {
        length = 0;
        for (j from 1 to M
            if (x[i+j] == y[j]) length ++;
            else                break;
        if (length > max_length) {
            max_length = length;
            pointer = i ;
        }
        if (max_length == M)
            break;
}
```

그림 2. pseudo-code for Lempel-Ziv compression
Fig. 2. pseudo-code for Lempel-Ziv compression.

이와 같이 Lempel-Ziv 알고리즘의 특수성 때문에 기존의 Lempel-Ziv 압축기는 dictionary 데이터를 shift 하기 위한 작업과 심벌 matching 검색 작업에 많은 반복동작이 필요하여 아주 많은 시스템 클럭이 필요하게 된다. 이것은 Lempel-Ziv 압축기의 성능을 저하시키는 원인이 된다. Lempel-Ziv 압축기의 성능을 향상시키기 위하여는 심벌 matching 작업을 빨리 판단할 수 있는 새로운 구조와 dictionary 데이터의 shift 작업을 줄일 수 있는 효과적인 개선 방법이 요구된다.

Ⅲ. 제안하는 Lempel-Ziv 하드웨어 구조

1. 기본 구조

본 논문에서는 Lempel-Ziv 알고리즘을 이용하여 실시간 데이터 압축/복원을 실행하는 하드웨어 아키텍처를 제안한다. Lempel-Ziv 알고리즘 그대로 데이터 압축/복원에 적용할 경우 알고리즘의 특성상 실시간 처리가 어려운 상황이다. Lempel-Ziv 알고리즘은 기본적으로 입력 데이터 버퍼 window와 내부의 데이터를 저장해 놓는 dictionary 메모리 구조를 가지고 있다.

이 알고리즘은 window데이터를 가지고, dictionary 데이터와의 비교를 통하여 matching length가 2이상인 최대 matching이 발생한 곳의 address pointer와 matching length를 전송함으로 해서 실제 data를 압축해서 전송하는 효과를 얻는다.

이러한 알고리즘을 사용함에 따라서 우선 matching을 수행하는 비교 연산과, 비교의 결과에 따른 dictionary와 window 데이터의 shifting 연산이 알고리즘의 대부분의 연산을 차지하게 된다. 이러한 비교와 shifting 연산은 실시간 처리에 있어서 장애 요소가 되고 있고, 이로 인해서 입력 bit-rate가 고정적인 데이터를 실시간으로 압축해서 입력 bit-rate보다 낮은 출력 bit-rate로 전송하는데 어려움이 있다.

이에 본 논문에서는 비교와 shifting연산을 효과적으로 처리하도록 하는 실시간 압축복원 하드웨어 구조를 제안한다.

2. 전체 구조

Lempel-Ziv 알고리즘을 이용하여 압축하는데 있어서 가장 큰 문제점인 비교 횟수와 shifting을 효율적으로 하기 위한 구조를 제안한다.

제안하는 구조는 다음의 특징을 갖는 구조이다.

- Rotational FIFO 구조의 buffer 메모리부
- 병렬 데이터 처리부
- 비교횟수 제어부

우선 비교 횟수를 감소시키는 것은 압축의 효율을 저하시키는 요인이기 때문에 횟수의 단순 감소가 아닌 비교의 효율성을 향상시키기 위해 병렬 비교를 위한 메모리 구조를 갖는다. 그리고, 제안하는 구조는 한번에 각각 3개의 window 와 dictionary 데이터를 읽어서 이것을 한번에 비교하게 된다. 또한 2개의 추가적인 비교기를 통하여 다음의 비교를 해야 할지 아니면 계속 비교를 해야 할 지를 1 clock cycle 내에 결정할 수 있는 구조를 갖는다.

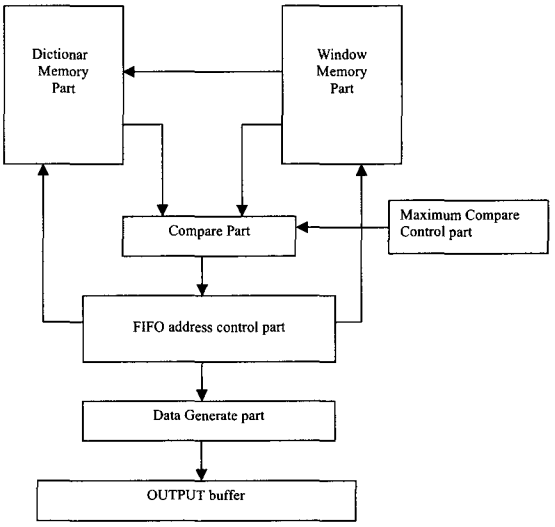


그림 3. 제안하는 알고리즘의 전체 구조
Fig. 3. Structure of proposed algorithm.

그리고, buffer인 window와 dictionary 모두를 rotational FIFO 구조로 구성함으로 해서 실시간으로 들어오는 데이터에 대한 buffering 역할을 할 수 있도록 하였고, dictionary의 가장큰 문제 이었던 shifting을 최대 matching length(Lmax) 만큼의 clock cycle로 처리할 수 있는 구조를 갖는다.

제안하는 전체 구조를 보면 그림 3과 같다.

3. Window / Dictionary 구조

제안하는 압축기의 구조는 3개의 데이터를 병렬로 비교하는 구조를 갖는다. 이러한 비교 구조를 위해서 window와 dictionary는 3개의 bank로 구성되는 메모리 구조를 갖는다. 그리고, 비교후의 데이터 shifting에 들어가는 시간을 최소화하기 위하여 window/dictionary 모두 rotational FIFO 구조를 갖는다.

window/dictionary의 크기를 크게 하면 데이터의 압축 율에는 좋은 영향을 주지만, 반면에 비교 횟수가 증가되어 전체 실행 시간을 증가시키는 요인이 된다.

표 1은 일반적인 텍스트에 대하여 시뮬레이션을 수행했을 때의 window size와 dictionary size와 압축 비의 관계를 보여준다. 압축 비는 window의 size가 크기가 클 때는 window size에 관계없이 dictionary 크기에 크게 의존하고 있다는 것을 보이고 있다. 즉 dictionary

의 크기를 크게 하면 압축 율이 좋아지고 있다. 그런데 dictionary의 크기를 크게 하면 압축 율에는 좋지만 하드웨어의 구현에는 나쁜 영향을 주게 된다. 따라서 본 연구에서는 dictionary의 크기를 1024, window 의 크기를 20으로 설정하였다.

표 1. 압축률과 window size, dictionary size 와의 관계

Table 1. Comparison of compression rate vs window and dictionary size.

Window size	Dictionary size	Results byte	CR
20	512	10439	0.67431
32	512	10433	0.673923
20	1024	9484	0.612622
32	1024	9486	0.612751
20	2048	8584	0.554486
32	2048	8591	0.554938

4. 비교 구조

Lempel-Ziv 압축 알고리즘의 하드웨어 구현에 있어서 가장 큰 문제는 dictionary의 검색에 필요한 clock cycle이다. Lempel-Ziv 알고리즘은 하나의 matching 패턴이 결정되었을 때 dictionary의 start point를 하나씩 증가시키면서 최대 matching 패턴을 검색하는 sliding window방식의 비교를 반복하도록 되어 있다. 이것을 하드웨어를 이용하게 되면 만약 1024byte의 dictionary size의 구조에서 한번 검색을 완료하는데 best case의 경우 1024 cycle이 필요하고, 통상 S_d (dictionary size)+α cycle 만큼의 시간이 필요하게 된다. 이것을 효과적으로 줄이는데 있어서 두 가지의 방법을 제안하다. 한가지는 Sd cycle을 줄이는 것이고, 다른 한가지는 α를 줄이는 것이다.

Sd를 줄이는 방법으로 제안하는 구조는 3개의 데이터를 동시에 비교하는 구조를 갖는다 이것을 위해서는 3개의 비교기를 사용하고, 여기에 2개의 비교기를 추가로 사용하여 이미 읽어 들인 데이터를 이용하여 다음의 matching이 일어 날것인지 아닌지를 미리 판단할 수 있다.

즉, 현재의 start point address의 증가를 1씩 하는 것이 아니고, 2씩 할 수 있는 판단을 현재의 비교와 동시에 할 수 있다는 것이다. 이러한 구조를 이용하는 경우 최대 $S_d/2$ 만큼의 감소를 얻을 수 있는 것이다. 그림 4는 전체 비교 부분의 구조를 나타내는 것으로 비교기 C_4, C_5는 현재의 패턴을 위한 비교가 아니고, 현재 패턴의 비교결과에 따라서 dictionary의 start point address를 하나 증가시킨 다음의 비교 결과를 미리 볼 수 있도록 만들어 놓은 것이다.

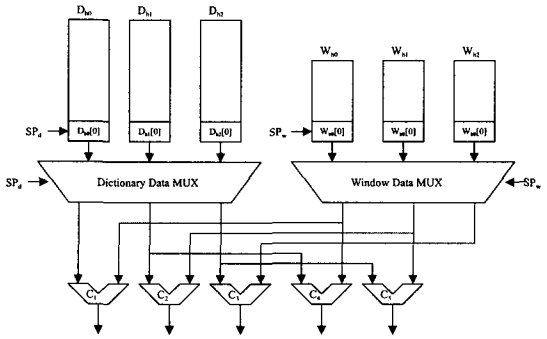


그림 4. 전체 비교 부분의 구조

Fig. 4. Structure of total compare part.

그리고, α를 줄이기 위하여 그림 4에서와 같이 C_1, C_2, C_3 세개의 비교기를 이용하여 현재의 matching이 결과에 따라서 다음 데이터를 비교해야 할 것인지 아닌지를 빠른 시간에 결정할 수 있도록 하였다.

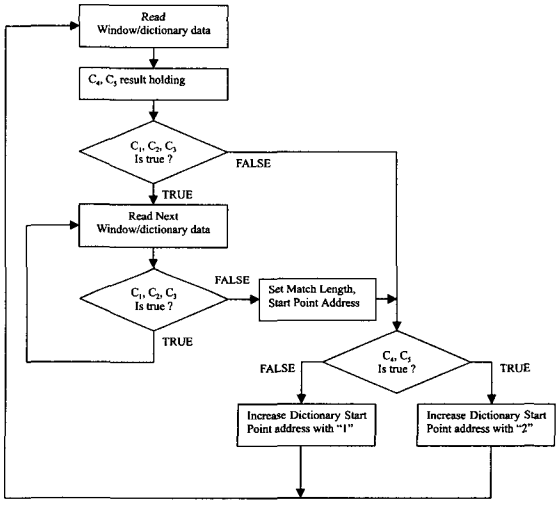


그림 5. 전체 흐름도

Fig. 5. Process Flow.

전체 흐름은 그림 5와 같고, 비교 결과에 따라서 SP_d, SP_w의 값이 결정된다. 그림 5에서 알 수 있듯이 C_1, C_2, C_3의 결과가 모두 참일 경우에는 현재의 matching Lmax가 3이상이 될 가능성이 있으므로 다음의 데이터

에 대한 비교가 행하여지고, 거짓일 경우에는 더 이상 비교를 할 필요가 없으므로, 현재의 matching 길이와 start point address(SP_d)를 저장하게 된다. 그리고, 현재의 matching이 거짓이 되었을 때, C_4, C_5의 결과에 따라서 SP_d를 1을 증가 할 것인지 아니면 2를 증가시킬 것인지를 결정한다. 이렇게 함으로써 기존의 순차 비교 방식에 비하여 현재의 matching을 판단하는데 필요한 클럭을 줄일 수 있고, 이미 읽어 들인 데이터를 이용하여 다음의 비교를 수행해야 할지를 빨리 판단하여 S_d를 줄여 나갈 수 있다.

5. Shifting 구조

비교가 끝이 나게 되면 현재까지의 최대 matching의 결과에 따라서 압축 데이터가 결정되게 되는데, matching 패턴의 최대 길이(Lmax)가 3이상인 경우에는 압축을 하고, 그렇지 않은 경우에는 압축을 하지 않고 현재의 데이터 하나만을 그대로 전송하도록 최종 data packet을 만들게 된다.

이렇게 압축 데이터가 결정되게 되면 데이터는 window를 나와서 dictionary와 압축 데이터 buffer로 들어가게 된다. 압축된 데이터를 dictionary로 전송하는 과정에서 shifting 동작이 일어나게 되는데, 제안하는 구조에서는 이러한 shifting 동작에 들어가는 시간을 최소화하기 위하여 window/dictionary 모두 rotational FIFO 구조를 사용한다. 이렇게 함으로써 shifting 동작에 필요한 cycle은 Lmax(최대 matching 길이) 만큼의 clock으로 shifting 동작을 수행 할 수 있다. 그림 6은 shifting 구조를 보여주고 있다.

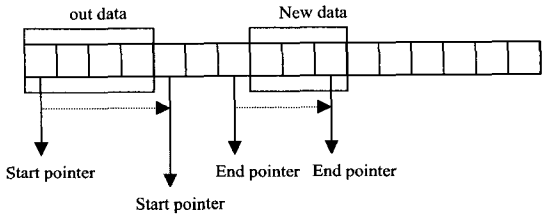


그림 6. shifting 동작
Fig. 6. Operation of shift.

그림 6과 같이 모든 dictionary와 window FIFO 데이터는 start/end pointer를 이용하여 shifting 동작이 일어나게 된다.

- 새로운 데이터의 입력

신규 데이터의 입력 시에는 end pointer만 증가하고, start pointer는 그대로 있게 된다.

- FIFO에서 데이터의 출력

데이터의 출력은 start pointer만을 이동시키는 것으로 동작하게 된다.

이러한 이동은 계속해서 일어나고 만약 end/start pointer가 FIFO memory의 마지막 어드레스에 도달하면 다시 처음으로 돌아가게 된다.

6. 변형 shifting 구조 I

제안된 기본 구조에서 변형된 구조로 이 구조는 window와 dictionary의 메모리를 분리시키지 않고 window와 dictionary가 하나의 메모리 bank를 사용하도록 하는 것이다. 이러한 구조를 사용할 경우 앞에서 언급하였던 shifting 동작에 필요하였던 Lmax만큼의 shifting cycle을 2 cycle로 줄일 수 있다. 그림 7은 변형된 shifting 구조를 보여 주고 있다. 그림 7에서와 같이 변형된 구조는 기존의 메모리 구조에서 D_{b0}와 W_{b0}를 하나의 memory bank로 통합한 구조를 갖게 된다. 이렇게 함으로써 window 데이터가 비교가 끝나고, 압축을 하기 위하여 window 영역에서 dictionary 영역으로 shifting 되어야 할 때 단순히 address pointer의 값만을 변화시킴으로써 데이터의 shifting 동작을 끝내게 된다. 이러한 구조를 이용하는 경우 shifting 동작을 빠르게 처리 할 수 있으나, 비교 시작시의 데이터를 buffering 해야 하는 단점을 가지고 있다, 그러나 하나의 데이터를 shifting 하는 것이 적어도 2개의 clock cycle을 소요한다고 본다면, buffering에 소요되는 overhead는 상쇄될 수 있다.

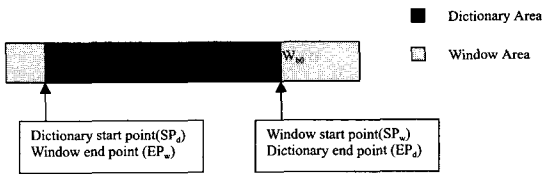


그림 7. 변형 shifting 구조 I의 메모리 구조
Fig. 7. Memory structure of modified shift structure.

7. 변형 shifting구조 II

그림 3의 전체 구조를 보면 maximum compare control part가 있는데 이것은 입력 데이터의 bit-rate가 출력 데이터의 bit-rate보다 높은 경우 입력은 정해진 시간에 스트림의 형태로 계속 들어오게 되어 있고, 비교가 끝나지 않았을 경우 정해진 비교횟수 만큼을 비

교하고, 강제적으로 현재까지의 비교결과를 이용하여 압축 데이터를 만들도록 하는 것이다. 이 값은 사용하고자 하는 입력 bit-rate와 출력 bit-rate, 그리고, system clock에 의해서 결정되게 된다.

통상 system clock은 입력 bit-rate에 비하여 높은 clock 주파수를 사용하게 된다. 이것은 하나의 입력 데이터가 들어오는 동안 system clock을 이용하여 비교를 하고, 그 결과를 출력하도록 하는 것이다. 이런 경우에 높은 system clock을 사용하면 비교에서 발생하는 만큼의 비교 횟수를 충분히 처리 할 수 있지만 그런 경우 하드웨어 cost와 시스템의 불안정성이 증가하는 단점이 있다. 이것을 효과적으로 처리하기 위하여 window 단에 $S_w \times 0.5$의 버퍼를 추가함으로 해서 하나의 데이터를 처리하는데 필요한 시간을 1 F_{in} cycle(F_{in} : 입력 bit-rate에 의한 1 clock 주기)에서 1 + ($S_w \times$ 0.5) F_{in} cycle로 확장해서 비교하는데 할당된 시간을 증가시킬 수 있도록 하는 구조이다.

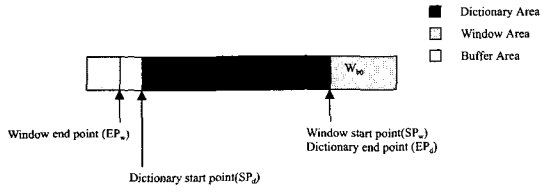


그림 8. 변형 shifting구조 II 의 메모리 구조
Fig. 8. Memory structure of modified shift structure II.

Ⅳ. 실험결과 및 고찰

제안하는 압축기의 알고리즘과 하드웨어 구조의 실행 파라메타를 추출하기 위하여 주요 기능블록을 포함한 전체 Lempel_ziv 압축기를 C 언어를 사용하여 시뮬레이션 하였다. 압축기의 성능에 절대적으로 영향을 주는 window와 dictionary 크기는 데이터 압축 율에 영향을 주며 표 1에 나타난 바와 같이 압축 율은 window 크기에는 영향을 받지 않으며 dictionary가 클수록 압축률이 좋은 것으로 나타났다. 그러나 dictionary의 크기를 크게 하면 압축 율은 좋아지지만 하드웨어 구현에는 나쁜 영향을 주게된다. 따라서 본 연구에서는 여러 가지 파라메터를 종합 시뮬레이션 한 결과 dictionary의 크기는 1024, window 크기는 20으로 하는 것이 최적 화된 값으로 분석되었다. 또한 비교 횟수도 그림 9에 보인 것처럼 일정 횟수를 초과하면 압축 율이 포화 상태로 되는 것을 알 수 있었다.

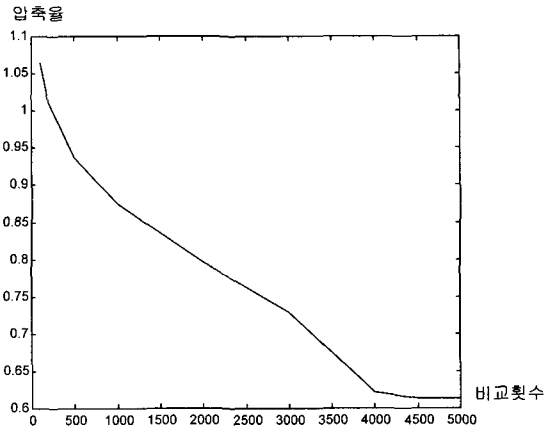
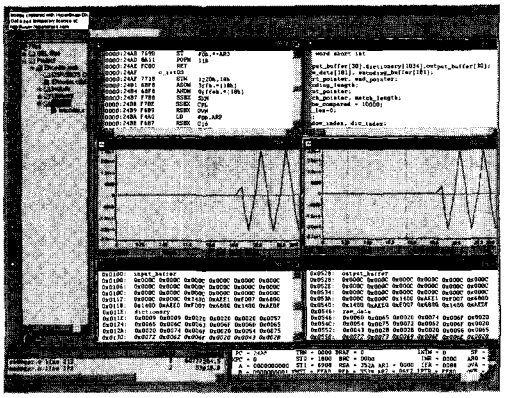
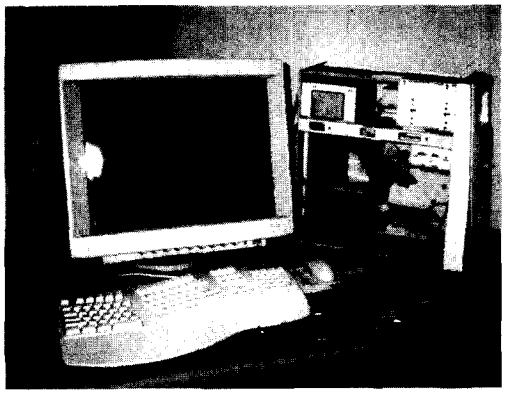


그림 9. 비교횟수에 대한 압축 율 변화
Fig. 9. Compression rate vs compare counts.



(a) DSP Emulation 화면



(b) DSP Board 실험장비

(original data) (복원 데이터)

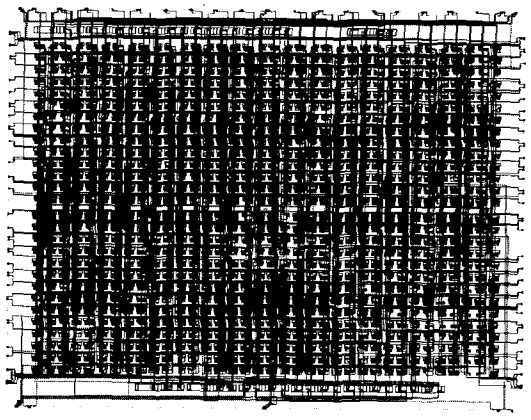
(c) 압축/복원 데이터

그림 10. DSP 에서의 압축기 동작검증
Fig. 10. Verification of compressor in DSP.

C 언어의 시뮬레이션을 통하여 최적화 되어 설정된 본 논문에서 제안하는 압축기 구조를 실제로 하드웨어 상에서 동작여부를 확인하기 위하여 TI사의 TMS320 C5402 DSP 디바이스에 프로그램을 탑재하여 실행한 결과 그림 10에 보인 것처럼 정확하게 동작함을 확인할 수 있었다.

사용 FPGA	QuickLogic QL4058 208pin PQFP
사용 CLB 블록수	Encoder = 688 개 Decorder = 295 개
Clock 주파수	33Mhz
Data Rate	256Kbps

(a) 압축기 설계구현



(b) Place & Route 된 설계도면

그림 11. FPGA를 사용한 압축기 설계
Fig. 11. Design of compressor with FPGA.

알고리즘과 하드웨어 상에서 동작검증이 완료된 본 논문에서 제안하는 Lempel-Ziv 압축기를 사용하여 256Kbps 전용선 응용에 사용하고자 VHDL을 사용하여 시스템을 모델링 하였으며 QuickLogic 사의 FPGA로 합성하여 시스템클럭 33Mhz 비트율 256Kbps를 처리하는 압축기를 설계하였다. 그림 11은 설계된 압축기의 구현결과와 Place & Route된 설계도면을 보여주고 있다.

기존 압축기와 본 논문에서 제안한 압축기의 특성비교를 표 2에 나타내었다.

표 2. 압축기의 비교
Table 2. Comparison of Lempel-Ziv Compressor.

구분	제안하는 구조	Jung[3]
응용분야	입출력 Data Rate가 상이한 시스템에 적용이 쉽다.	Data Rate 제어기능을 위한 추가적인 하드웨어 필요
회로의 융통성	Dictionary 크기를 가변적으로 운용할 수 있다.	Dictionary 크기에 따라 PE(process Element)를 추가하여 재구성 필요
Gate 수	11K gates	15K gates
개발기간	short (ASIC)	long (Full Custom)

V. 결 론

본 논문에서는 실시간 데이터 전송로의 효율적인 사용을 위한 Lempel-Ziv 압축기의 새로운 하드웨어 구조를 제안하였다.

제안하는 구조에서는 dictionary 크기를 최적화 하는 방법과 복수개의 바이트를 동시에 비교하는 matching 바이트 처리 방법, 그리고 회전 FIFO 구조를 이용하여 shift 동작을 제어하는 방법을 제안하여 입력 데이터의 실시간 처리가 가능하도록 하였다.

제안된 압축기를 상용 FPGA 칩 상에서 실험한 결과, 설계된 Lempel-Ziv 압축기는 33Mhz 시스템 클럭주파수에서 256Kbps 전용선에 오류 없이 동작됨을 확인하였다.

통신망에서 인터넷 사용의 확장으로 전용선의 사용이 증가하고 있음을 고려할 때 본 논문에서 개발된 Lempel-Ziv 압축기는 실제 통신 전송로에서 전송 매체의 통신 효율을 향상시키는데 응용될 수 있다.

앞으로의 연구 과제로는 동일한 입력 비트 율에 대하여, 낮은 시스템 클럭 주파수로 동작하는 새로운 하드웨어 구조 설계에 대한 추가적인 연구가 진행되어야

하겠다.

참 고 문 헌

[1] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression", IEEE Trans. Inform. Theory, Vol. IT-23, No. 3, pp.337-343, 1977.

[2] J. Ziv and A. Lempel, "Compression of individual sequence via variable rate coding", IEEE Trans. Inform. Theory, Vol. IT-24, pp.530-536, 1978.

[3] J.A. Storer, "Data Compression: Methods and Theory", 1988,Rockville,MD:Computer Science Press.

[4] N. Ranganatha and S. Henriques, "High speed VLSI designs for Lempel Ziv-based data compression", IEEE Trans. on circuits and system-II: analog and digital signal processing, Feb. 1993, Vol. 40, No. 2, pp.96-106.

[5] N. Ranganatha and S. Henriques, "High speed VLSI designs for Lempel Ziv-based data compression", IEEE VLSI algorithms and architectures: Advanced Concepts, pp.255-265, 1993.

[6] R.Y.Yang and C.Y.Lee, "High throughput data compressor design usign content addressible memory", IEEE ISCAS94. Vol. 4. pp.147-150. 1994.

[7] B.Jung and W.Burleson, "A VLSI systolic array architecture for Lempel-Ziv based data compressoion", IEEE ISCAS94. Vol. 3, pp.65-68. 1994.

[8] B.W.Y. Wei. R. Tarver. J.S.Kim and K.Ng., "A single chip Lempel-Ziv data compressor", IEEE ISCAS93. pp.1953-1955, 1994.

[9] Young surk, Lee, Tae young Lee,Kyu Tae Park, "A Novel PE-Based Architecture for Lossless LZ compression", IEICE Trans. Fundamentals, Vol. E80.A No. 1, pp.233-237, January 1997.

[10] Jung BJ, Wayne P. Burleson, "Efficient VLSI for Lempel-Ziv Compression in Wireless Data Communication Networks", IEEE Transactions on Very Large Scale Integration(VLSI) System, Vol. 6, No. 3, pp.475-483, 1998.

[11] R.Zito-Wolf, "A systolic architecture for sliding window data compression", in Proc. IEEE Workshop VLSI Signal Processing, 1992, pp.339-351.

저 자 소 개

陳 庸 先(正會員)

1955년 2월 29일생. 1978년 한양대학교 전자공학과 졸업(학사). 1991년 한양대학교 대학원 전자공학과 졸업(석사). 1991년~현재 한양대학교 대학원 전자공학과. 박사과정. 1978년~1979년 전자통신연구원 연구원. 1980년~1987년 LG전자연구소 선임연구원. 1988년~1996년 Texas Instrument 연구소 연구원. 1997년~현재 경문대학 정보통신과 교수. <관심분야> 신호처리, VLSI설계, 디지털통신 시스템

鄭 正 和(正會員) 第37卷 SP編 第2號 參照