JPEG2000 영상 압축을 위한 EBCOT 설계

조태준* 이재홍**

요 약 고품질의 영상 압축기인 JPEG2000의 기본 압축 코덱인 EBCOT(Embedded Block Coding With Optimized Truncation)를 설계하였다.

영상 압축기에서 Context 추출 구현을 위하여 코드블록(Code block)으로 분할하고, 비트플랜(Bit-Plane)코딩을 했으며, 3가지 패스 그룹으로 분리한 후 ZC, RLC, MR, SC를 하였다. 산술부호화는 덧셈 연산과 쉬프트 연산만을 사용하는 MQ-coder를 사용하였으며, Context들의 누적 확률을 추정하여 테이블을 작성하였고, 압축데이터를 산출하였다.

영상 압축을 위한 엔트로피 코더의 하드웨어 구현은 VHDL를 이용하여 설계를 하고, Synopsys사의 논리 회로 합성 도구를 사용하여 합성을 하였으며, Altera사의 FLEX 10K250 Device를 이용하여 FPGA로 구현하였다.

1. 서 론

멀티미디어의 사용이 증가함에 따라 문자와 영상, 음성을 저장하거나 유·무선 통신을 통해 전송하려는 요구가 많아지고 있으며, 이러한 영상의 사용 증가에 따른 압축 방법도 다양해지고, 고품질의 압축을 요구하였다

기존 JPEG은 고주파와 중간주파수에서 훌륭한 압축률을 수행하지만 저주파(0.25bpp 이하)에서는 수행이 불가능하므로, JPEG2000은 저주파 비트율을 개선하여 네트워크 상에서 왜곡율과 특정부분의 이미지 품질을 한차원 높일 뿐만 아니라 비트레이트(Bit rate)에 따라 스케빌리티(Scalability)를 제공하는 우수성을 보이고 있다.

본 논문에서는 무손실 압축 방법으로서 JPEG2000에서 선택된 EBCOT (Embedded Block Coding With Optimized Truncation)의 구조에서 실질적인 압축을 수행하는 T1 블록의 구조를 제안하고 설계에 관한 연구 내용을 기술한다.

Context 추출 부분의 구성은 영상을 처리하기 적절한 블록 크기로 분할하고 비트플랜(Bit-Plane) 코딩을 하여 여러개의 비트면으로 분리하는 부분과 각 비트플랜별로 주변의 샘플과 이전의 비트플랜의 관계를 가지고 3가지 패스인 Significance Propagation pass, Magnitude Refinement pass, Clean up pass 그룹으로 나누게 되며 4 가지의 ZC(Zero coding), SC(Sign Coding), RLC(Run-length Coding), MR (Magnitude Refinement) 코딩 방법으로 19개의 Context 중 현 샘플이 해당되는 Context(CX)와 Decision(D)을 추출하게 된다.

산술부호화(Arithmetic coding)는 곱셈연산과 덧셈연산을 사용하는 Q-coder, QM-coder 대신 덧셈 연산과 쉬프트 연산을 사용되는 MQ-coder를 사용하여 추출된 Context와 Decision을 가지고 실질적인 압축을 산술부호화에서 수행하게 된다. MQ-coder는 추출된

* , ** 한발대학교 컴퓨터공학과

Context와 Decision으로 확률 추정 테이블과 47개 확률값을 이용하여 압축데이터를 산출하며, 다음에 입력될 각각의 Context의 확률을 추측하여 확률 추정 테이블에 누적시키게 된다.

EBCOT의 T1 블록의 구현은 Visual C++로 프로그램 하여 알고리즘을 증명하였으며 하드웨어의 구현은 VHDL로 설계하여 Synopsys로 논리 합성하였고, 합성한 로직을 Altera FLEX 10K250 Device를 이용하여 FPGA로 구현하였다.

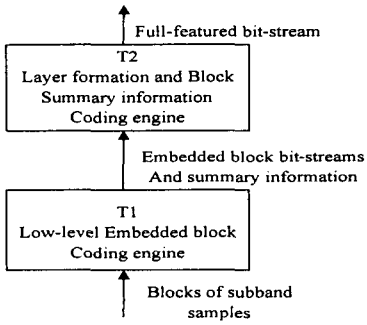
2. EBCOT

2.1 EBCOT의 구성

EBCOT(Embedded Block Coding With Optimized Truncation)는 94년 비트 플랜 코딩과 산술부호화를 사용하는 LZC(Layerd Zero Coding)방식과 블록 코딩 방식을 혼합하여 Taubman 과 Zakhor에 의해 발표되었다.

처음의 비트플랜 코딩은 한번의 스캔 과정을 거치어 코딩하였으나 SPIHT의 방법을 도입하여 하나의 비트플랜을 3번의 스캔 과정을 거치어 코딩을 하고 있다.

[그림-1]은 EBCOT 구성으로서 T1, T2 블록으로 나눌 수 있다. T1 블록은 웨이블릿 변환과 양자화를 거친 서브밴드를 EBCOT에서 처리하기 적당한 크기의 코드블록(Code block)으로 나누며, 여러 층의 비트플랜(Bit-plane)으로 분할하여 3 가지의 패스(Pass)로 구분하여 각각의 샘플로 19개중 한 개의 Context와 Decision이 추출되며, 추출된 Context와 Decision을 이용하여 무손실 압축 방법인 MQ-coder를 사용하여 확률을 추측 및 확률 추정 테이블을 만들고 누적시키며 추측된 값을 이용하여 압축된 데이터를 얻을 수 있다. T2 블록은 T1에서 만들어진 압축된 비트스트림(Bit-stream)을 주어진 비트레이트(Bit rate)에 맞게 데이터를 삭제하거나 주어진 비트레이트를 맞추어 점진적인 SNR 스케빌리티(Scalability)와 Resolution 스케빌리티(Scalability)가 가능하게 압축된 데이터를 재배열시키는 블록이다.

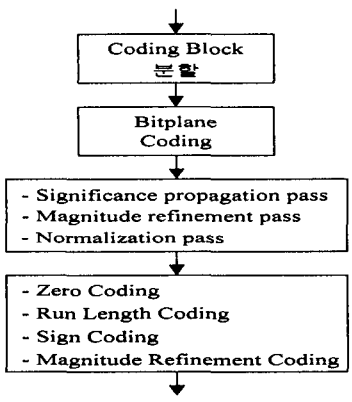


[그림-1] EBCOT 구성

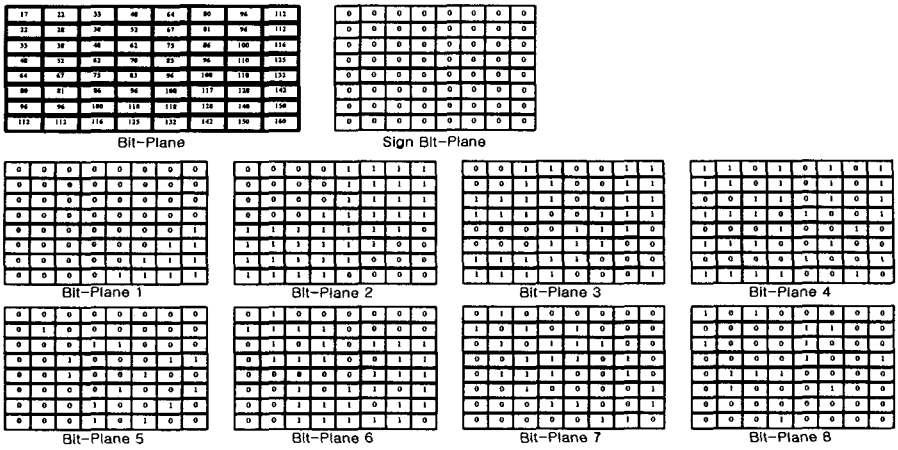
3. Context 추출

3.1 Context의 구성

Context를 추출하는 과정은 [그림-2]와 같이 제안하였다. T1으로 입력된 영상의 서브밴드를 처리하기 쉬운 크기의 여러개의 블록으로 나누어지게 되며, 비트플랜 코딩은 하나의 블록을 여러개의 층으로 분할한다. 분할된 각 층을 Significance Propagation pass(P1), Magnitude Refinement pass(P2), Clean up pass(P3)의 그룹으로 분리하고, 데이터의 특성에 따른 4가지의 코딩 ZC(Zero coding), SC(Sign Coding), RLC(Run-length Coding), MR (Magnitude Refinement)을 수행하므로서 Context와 Decision을 추출할 수 있다.



[그림-2] Context 추출 블록 다이어그램

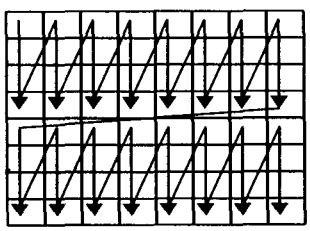


[그림-3] 비트플랜 코딩

코딩블록으로 양자화된 데이터가 입력되면 처리하기 가장 적절한 데이터 크기로 나누어야 하며 최고 64×64까지 나눌 수 있지만 본 논문에서 메모리 사용을 최소로 하기 위하여 8×8 크기를 사용하였다.

비트플랜 코딩은 4가지의 코딩을 하기 위한 샘플을 여래개의 층으로 분리하는 것을 말한다. [그림-3]은 비트플랜 코딩의 예를 보여주고 있는 것으로 원 데이터를 부호 비트를 먼저 추출하여 비트플랜을 만든 후 나머지 양수 값으로 여러개의 비트플랜으로 분리하게 된다.

비트플랜을 한 후 3가지의 Pass 그룹으로 분리하고 데이터의 특성에 따라 4가지 코딩을 하기 위해서 데이터를 스캔하게 된다. 스캔하는 패턴은 [그림-4]과 같이 4개의 샘플을 세로로 스캔하게 된다.



[그림-4] 코드 블록 스캔패턴

3.2 Three Pass Coding

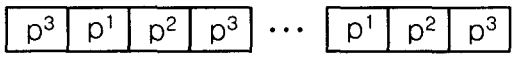
본 논문에서는 제한하는 3 패스 코딩은 각 샘플이 3 패스 중 어디에 속해야 하는지를 먼저 결정을 하고 P1, P2, P3 순으로 코딩을 진행하며, 각 패스에 주어진 코딩방법으로 Context와 Decision을 추출한다.

비트플랜의 크기를 η(m, n)이라고 하고, 비트플랜의 상태값을 가지는 변수를 σ(m, n)이라고 했을 경우, 3가지 Pass 와 4가지 코딩은 상태값을 가지고 있는 변수 σ(m, n)를 참조하여 코딩을 한다.

3가지의 Pass를 하는 방법은 다음과 같다.

- Pass 1 (Significance Propagation pass)
  - σ(m, n) = '0' 일 경우
  - σ(m, n) 주변의 8개의 샘플중 '1'이 있음
  - Zero coding, Sign Coding
- Pass 2 (Magnifude Refinement Pass)
  - σ(m, n) = '1' 일 경우
  - Magnifude Refinement Coding(MR)
- Pass 3 (Cleanup Pass)
  - σ(m, n) = '0' 일 경우
  - σ(m, n) 주변의 8개의 샘플이 모두 '0' 일 경우
  - Run Length Coding(RLC), Zero Coding(ZC), Sign Codimg(SC)

Context와 Decision을 추출하게 되면 데이터를 3가지의 Pass 형태로 추출된다. 3가지 패스에 의해서 나온 데이터들은 P1, P2, P3의 순서대로 비트스트림을 전송하게 되며, 코드블록의 맨 상위 플랜은 σ(m, n)이 모두 '0'이 되므로 P3 코딩만 수행하게 되며 그 순서를 나열해 보면 [그림-5]와 같이 나타난다.



[그림-5] 비트스트림(Bit-streams)

3.3 RLC(Run Length Coding)

RLC은 2가지의 Context로 17과 18값을 가지고 있으며, 세로의 연속적인 4개의 샘플값이 σ(m, n) = '0' 이며, σ(m, n) 주변의 모든 샘플이 '0'값을 가지는 경우 Context 18을 사용한다.

현재의 4개의 모든 샘플이 '0' 또는 '1'인가에 따라 Decision 값이 달라진다. Decision 값이 1인 경우 4개의 샘플 중에 '1' 값이 1개 이상 가지고 있다는 의미로 첫 번째 '1'값의 위치를 찾기 위해 17이라는 Context를 2번 사용하게 되며, '1'값의 위치에 따라 Decision 값이 결정된다.

[표-1] RLC으로 추출되는 Context

Context		Decision
18	4개의 샘플 모두 = 0	0
	4개의 샘플 모두 ≠ 0	1
17	첫번째 샘플 = 1	00
	두번째 샘플 = 1	01
	세번째 샘플 = 1	10
	네번째 샘플 = 1	11

3.4 ZC(Zero Coding)

ZC은 9개의 Context를 사용하고 있으며, LH 또는 LL 서브밴드, HL 서브밴드, HH 서브밴드에 따라 Context값이 달라지며, 현 샘플 주위의 8개의 샘플값을 참조를 하여 부호화를 하며, [표-2]의 9개중 하나의 값을 가지게 된다. ZC은 현 샘플의 σ(m, n) = '0'이면서, σ(m, n) 주변의 샘플들이 모두 '0' 이였을 경우 ZC을 하게 되며, h_i는 수평방향, v_i는 수직 방향, d_i는 대각선 방향을 뜻하는 것으로 그들의 합에 의해서 선택되어진다.

Decision 값으로는 현재 샘플의 값을 가진다.

[표-2] ZC로 추출되는 Context 값

* X : Don't care

LH Subband (also used for LL) (vertically high-pass)			HL Subband (horizontally high-pass)			HH Subband (diagonally high-pass)		context
Σh_i	Σv_i	Σd_i	Σh_i	ΣV_i	Σd_i	Σd_{ii}	$\Sigma h_i + \Sigma v_i$	
2	X	X	X	2	X	≥3	X	8
1	≥1	X	≥1	1	X	2	≥1	7
1	0	≥1	0	1	≥1	2	0	6
1	0	0	0	1	0	1	≥2	5
0	2	X	2	0	X	1	1	4
0	1	X	1	0	X	1	0	3
0	0	≥2	0	0	≥2	0	≥2	2
0	0	1	0	0	1	0	1	1
0	0	0	0	0	0	0	0	0

3.5 SC(Sign Coding)

ZC, RLC을 한 후 코드블록에서 처음 '1'이 나왔을 때 한번만 수행되는 것으로 SC은 현재 샘플의 주변 부호비트의 연관성 여부에 따라 Context를 결정하고, 5개의 Context를 사용하고 있으며, σ(m, n) = '0' 이고, 현재 비트플랜의 샘플이 '1'일 경우 SC을 하게 된다.

[표-3]은 수평방향과 수직방향의 샘플과 부호에 따른 v값과 h값을 나타낸 것이다.

- Significant : σ(m, n) = '1'
- Insignificant : σ(m, n) = '0'
- Positive : 현 재의 샘플의 Sign 비트가 양수(0) 이다.
- Negative : 현 재의 샘플의 Sign 비트가 음수(1) 이다.

[표-5]는 결정되어진 h, v값에 따라 Context가 결정되어지며, Decision 값은 현재의 샘플의 부호비트와 [표-5]의 XOR 비트와

Exclusive-OR 하여 결정되어 진다.

[표-3] 수평방향과 수직방향의 샘플과 부호에 대한 값

v0(h0)	v1(h1)	v(or h)
Significant, positive	Significant, positive	1
Significant, negative	Significant, positive	0
insignificant	Significant, positive	1
Significant, positive	Significant, negative	0
Significant, negative	Significant, negative	-1
insignificant	Significant, negative	-1
Significant, positive	insignificant	1
Significant, negative	insignificant	-1
insignificant	insignificant	0

[표-5] SC로 추출되는 Context 값

h	v	XOR	Context
1	1	0	13
1	0	0	12
1	-1	0	11
0	1	0	10
0	0	0	9
0	-1	1	10
-1	1	1	11
-1	0	1	12
-1	-1	1	13

3.6 MR(Magnitude Refinement Coding)

MR 코딩은 현재 샘플과 이전의 샘플과의 연관성을 가지고 Context를 추출하는 것으로 σ (m, n) = '1'인 경우 수행하는 것으로 3가지의 Context를 가지고 있다.

[표-6] MR로 추출되는 Context 값

$\Sigma h_i + \Sigma v_i + \Sigma d_i$	$\sigma(i, j)$	Context
x	1	16
≥ 1	0	15
0	0	14

4. 산술부호화

4.1 산술부호화의 개요

산술부호화는 P.Elias에 의해 제안되어 Rissanen, Pasco, Pubin등에 의해 발전하게 되었으며, 순차적인 샘플의 길이에 비례하는 정도의 계산량으로 부호화 할 수 있다.

데이터를 압축하는 방법으로 손실 압축과 무손실 압축으로 나뉘어진다. 여기서 사용되는 산술부호화기는 무손실 압축 방법이다. 기존의 산술부호화 방법은 구간을 [0, 1) 사이의 수를 가지고 유일한 산술 코드를 만든다. 단위 구간 [0, 1)사이의 실수는 무한하기 때문에 이렇게 유일한 산술 코드를 만드는 것이 가능하다.

그러므로, 메시지가 길면 길수록 그 구간은 좁아지고, 그 구간을 표시하기 위한 비트 수는 늘어난다. 이 때, 코딩에 사용되는 모델에서 주어지는 샘플의 확률에 기반하여 좁아지는 비율이 결정된다. 많이 발생하는 샘플은 적게 줄어들고, 적게 발생하는 샘플은 많이 줄어든다. 따라서, 적게 발생하는 샘플을 코딩할 때, 더욱 더 많은 비트가 필요하게 된다.

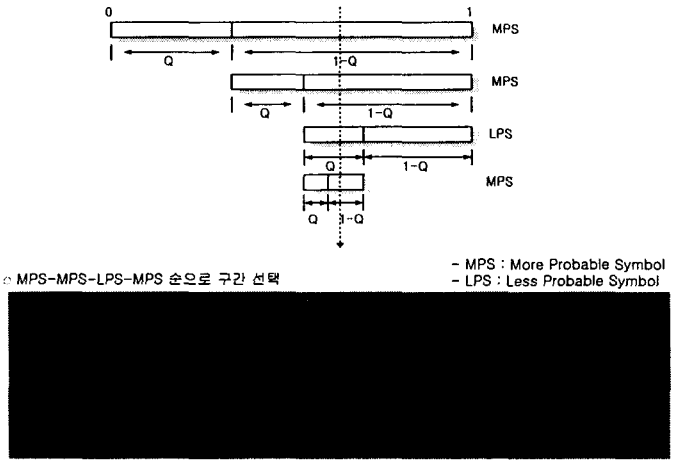
이진 산술부호화는 현재의 구간을 2개의 구간으로, 1이 나올 확률 구간을 MPS, 0이 나올 확률 구간을 LPS의 구간으로 나누며, LPS가 나올 확률은 Q이다.

[그림-6]은 산술부호화의 기본 과정으로 MPS, MPS, LPS, MPS 순의 구간이 선택 되었을때의 과정이다.

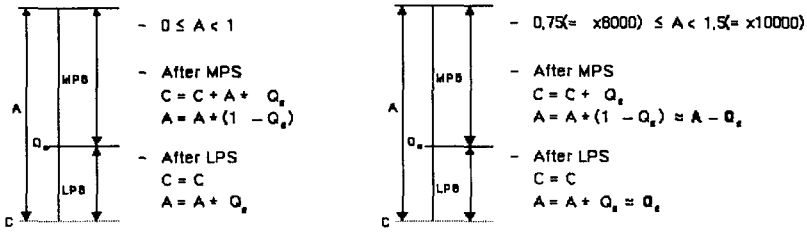
위와 같은 방법으로 많은 데이터를 산술부호화 하면 구간을 표현해야 하는 비트들은 무한정 늘어나게 될 것이다. 그리하여 본 논문에서는 이 문제점을 해결하기 위하여 MQ-Coder라는 산술부호화를 사용할 것이다.

기존의 방식은 MPS, LPS의 구간 값을 구하기 위해서 곱셈 연산과 덧셈 연산을 이용하여 값을 구해야 하는 문제점을 가지고 있다. 그러나 본 논문에서 사용되는 MQ-Coder는 이러한 문제점을 덧셈 연산과 쉬프트 연산을 이용하여 해결하였고, 범위는 [0.75, 1.5)를 사용하였다.

[그림-7]은 기존의 산술부호화와 근사화된



[그림-6] 산술부호화 기본과정



[그림-7] 기존의 산술부호화 및 근사화된 MQ-coder

MQ-Coder 이다.

4.2 확률 추정

산술부호화는 확률을 추정하여 구간을 계산을 하며, 추정된 확률은 현재와 같은 샘플이 다음에 들어왔을 때, 사용하기 위해 누적을 시킨다.

[표-7]은 확률의 추정된 값이 누적되는 테이블로 19개의 Context들의 초기 Index값과 MPS 값을 보여주고 있다. Index는 다음에 Context가 들어오면 참조 해야할 47개의 LPS의 확률 테이블의 Index를 가지고 있다. 이 LPS의 테이블을 이용함으로써 무한히 늘어날 수 있는 LPS의 확률을 47개의 값으로 고정시킬 수 있다.

[표-7] 확률 추정 테이블

Context(CX)	Initial Index	Initial MPS
Zero coding (0)	4	0
Zero coding (1 ~ 8)	0	0
Sign Coding (9 ~ 13)	0	0
Magnitude Refinement (14 ~ 16)	0	0
Uniform (17)	46	0
Run Length coding (18)	3	0

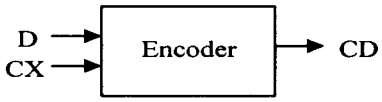
4.3 산술부호화기

[그림-8]은 산술부호화의 엔코더에 대한 입·출력 블록 다이어그램을 보여주고 있다.

[표-8] 엔코더의 레지스터 구조

	MSB			LSB
C-register	0000 cbbb	bbbb bsss	xxxx xxxx	xxxx xxxx
A-register	0000 0000	0000 0000	aaaa aaaa	aaaa aaaa

- a = interval 값을 나타내는 fractional bits
- x = C-register의 fractional bits
- s = carry-over를 막기 위한 spacer bits
- b = C-register에서 출력된 bit의 위치(실질적인 compressed data output 부분)
- c = carry bit



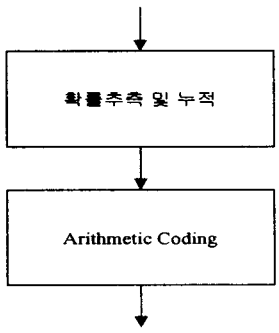
[그림-8] 산술보호화기 엔코더 입·출력

엔코더는 Context 추출 과정에서 얻어진 D(Decision)과 CX(Context)를 입력받아 CD(Code stream)를 출력하게된다.

[표-8]은 엔코더의 레지스터 구조를 나타내주고 있다.

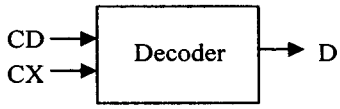
C 레지스터는 32비트를 사용하는 반면에, A 레지스터는 16비트를 사용하고 있다. 산술부호화에 의해 C 레지스터가 덧셈과 쉬프트 연산에 의해서 'b' 의 위치에 모두 채워지면 압축된 데이터로 보고 출력시키는 구조를 가지고 있다.

[그림-9]은 본 논문에서 제안하는 산술부호화의 블록 다이어그램이다. 입력되는 Context와 Decision값을 가지고 확률을 추측하고 누적시키며, 추측된 값은 산술부호화 코딩 블록에 의해 압축된 데이터를 산출할 수 있다.



[그림-9] 산술부호화 블록 다이어그램

[그림-10]은 산술부호화의 디코더에 대한 입·출력 블록 다이어그램을 보여주고 있다.



[그림-10] 산술부호화 디코더 입·출력

디코더는 CD(Code stream)과 CX(Context)를 입력받아 D(Decision) 값을 얻을 수 있다.

[표-9]는 디코더의 레지스터 구조를 나타내주고 있다.

[표-9] 디코더의 레지스터 구조

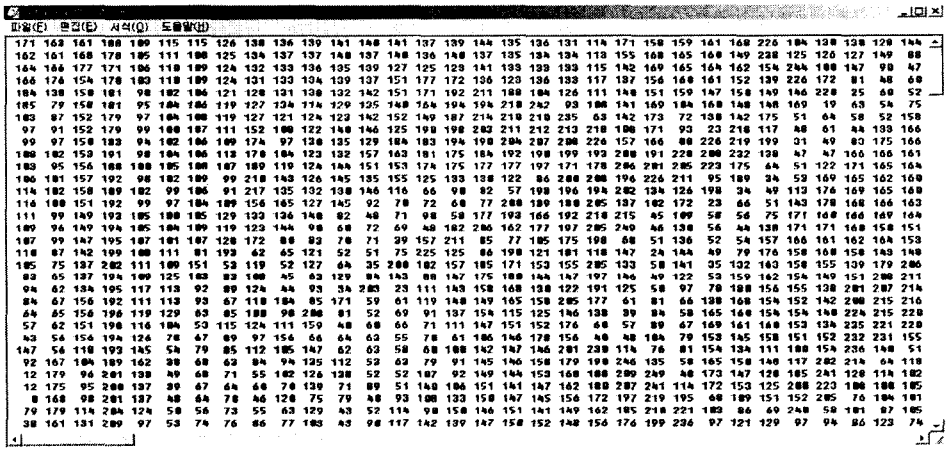
	MSB	LSB
Chigh register	xxxx xxxx	xxxx xxxx
Clow register	bbbb bbbb	bbbb bbbb
A-register	aaaa aaaa	aaaa aaaa

C 레지스터의 32비트를 상위 16비트를 Chigh 레지스터로, 하위 16비트를 Clow로 구성하고, A 레지스터는 16비트를 사용하고 있다.

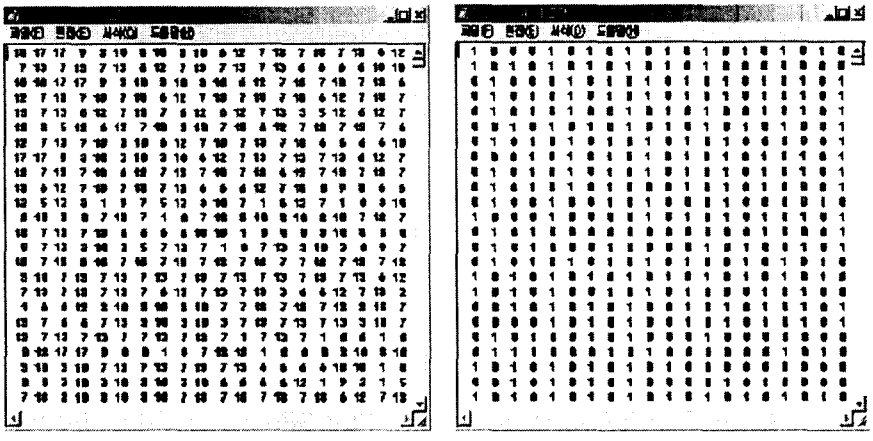
5. 실험 결과

본 논문에서는 영상 압축을 위한 엔트로피 코더를 설계하기 위해 샘플 주변의 관계성을

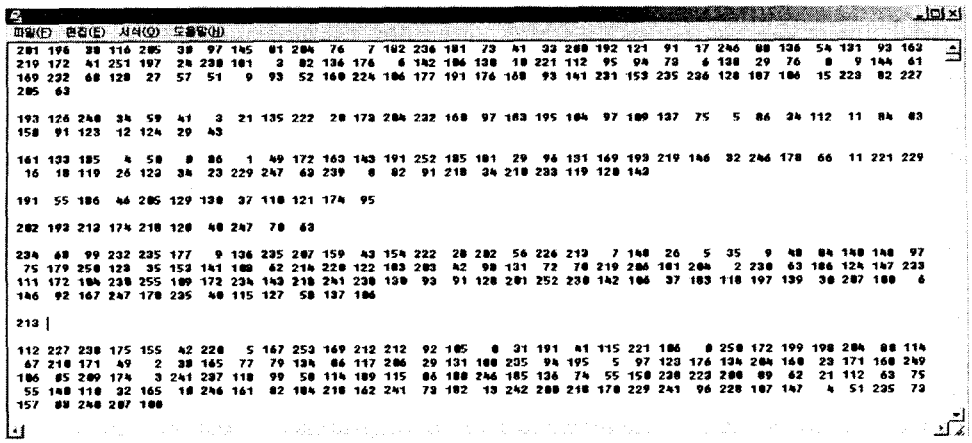
[그림-11] 원 영상과 웨이블릿 변환 및 양자화된 데이터



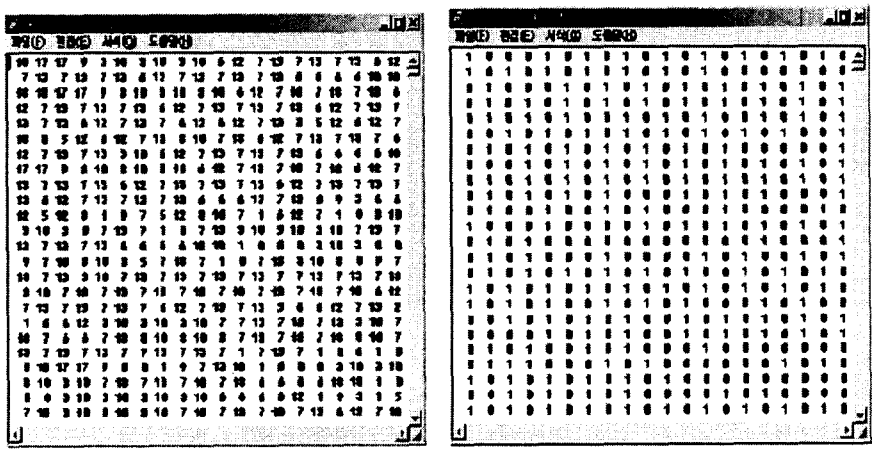
[그림-12] Lena 영상을 정수형으로 변환한 값



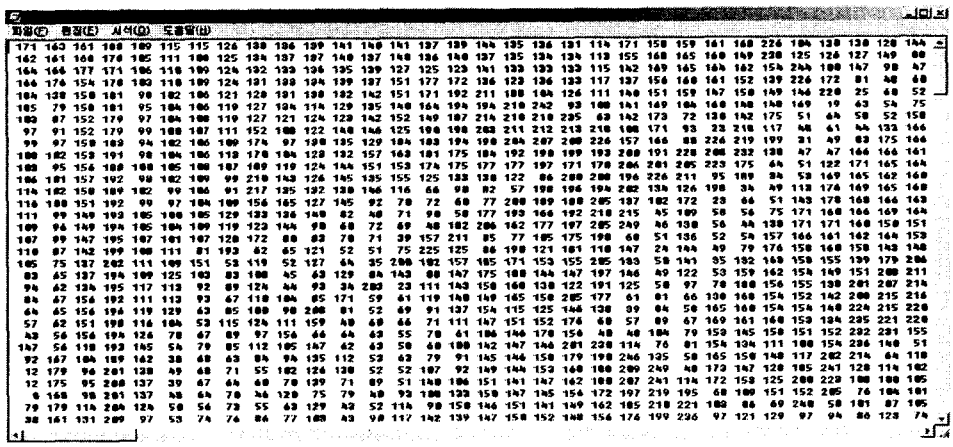
[그림-13] Context 추출 Encoder 결과값



[그림-14] 산술부호화 Encoder 값



[그림-15] 산술부호화 디코드 결과값



[그림-16] Context 추출 디코더 결과값

[그림-17] 역양자화 하기전 데이터 및 역웨이블릿 변환된 영상

나타내는 Context의 추출과 산술부호화중 무손실 압축방법을 사용하는 EBCOT(Embedded Block Coding With Optimized Truncation)의 블록 T1, T2중 T1을 구현하였다. 알고리즘의 분석 및 성능을 분석하기 위해서 Visual C++ 프로그램으로 엔트로피 코더를 구현 및 테스트'를 하였다.

영상의 압축의 압축률을 테스트하기 위해 256×256 크기의 영상을 사용하였으며, 영상의 명도의 차이에 의해 압축률이 차이가 많을 수 있으므로 인물영상과 배경영상등 여러 가지를 영상을 가지고 실험을 하였다.

다음 [그림-11]는 EBCOT의 입력 데이터로 사용될 원 영상과 웨이블릿 변환 및 양자화된 Lena 영상이다.

[그림-12]은 Raw 파일의 Lena 영상을 웨이블릿 변환 및 양자화된 데이터중 LL 서브밴드를 정수형으로 변환하여 출력한 값이다.

[그림-13]는 [그림-12]의 데이터를 Encode하여 Context와 Decision이 추출된 결과이다.

[그림-14]은 추출된 Context와 Decision를 산술부호화 Encode된 코드스트림 데이터 결과이다.

[그림-15]은 코드스트림을 Decode하여 Context와 Decision이 복구된 결과를 보여주고 있다.

[그림-16]은 산술부호화 디코드된 Context와 Decision을 Context 추출 디코더에 의해 복구된 데이터이다.

[그림-17]는 [그림-16]의 데이터 결과값을 웨이블릿 변환 형식으로 재배열 한 데이터와 역양자화 하여 역웨이블릿 변환을 실행했을때의 Lena 영상이다.

Context 추출 엔코더 및 디코더, 산술부호화 엔코더 및 디코더를 설계 및 구현하여 [그림-11]의 영상 데이터를 압축하고 복원하여 [그림-17]의 영상을 얻을 수 있으며, 이 두 영상이 동일함을 알 수 있다.

영상 압축을 위한 엔트로피 코더의 하드웨어 구현은 VHDL를 이용하여 설계를 하고 Synopsys사의 논리 회로 합성 도구를 사용하여 합성을 하였으며 Altera사의 FLEX 10K250 Device를 이용하여 FPGA로 구현하였다.

6. 결 론

본 논문은 무손실 압축 방법으로서 JPEG2000에서 선택된 EBCOT (Embedded Block Coding With Optimized Truncation)의 구조에서 실질적인 압축을 수행하는 T1 블록의 구조에 대하여 제안하고 설계를 하였다.

T1 단계는 실질적인 압축을 수행하는 단계로 코드블록으로 분할 및 비트플랜 코딩, Context 추출, 산술부호화 부분으로 구성되어 있다.

코드블록으로 분할 및 비트플랜을 함으로써 현재의 샘플이 변경되거나 잃어버려도 다른 코드블록에 영향을 주지 않을 수 있었으며, 처리하기 위한 메모리 사용을 최소한으로 줄일 수 있었으며, 비트플랜을 함으로써 비트레이트가 주어질 경우 비트레이트에 맞게 데이터를 삭제할 수 있거나, 점진적인 SNR 스케빌리티와 Resolution 스케빌리티에 맞게 압축된 데이터를 재배열 할 수 있는 기반을 제공할 수 있었다.

향후 연구과제로는 유·무선 통신의 전송 환경에 따른 실시간 영상 전송 서비스가 가능하도록 영상을 재배열하거나 스케빌리티를 지원할 수 있는 기능의 연구가 이루어져야 하겠고, 구현된 EBCOT의 칩 면적을 최소로 줄여, 칩으로 제작하여 다른 영상에도 적용할 수 있는 연구가 필요할 것이다.

참 고 문 헌

[1] 최정희, 김송주, 김영민, "JPEG2000 정지화상 압축 및 복원기술 개발에 관한 연구", 전자통신 기술 논문지, 제 2권 제 1호, 1999년 12월

[2] "웨이블릿 기반 신호처리 시스템 설계 (응용분야)" 반도체 설계교육센터 2000. 3.

[3] "JPEG2000 영상 압축프로세서 설계 및 구현", 반도체 설계교육센터, 2002. 5.

[4] Charilaos Christopoulos, "JPEG2000 Verification Model 7.0"
ISO/IEC JTC 1/SC 29/WG1 WG1N1684

[5] Charilaos Chritopoulos "The JPEG2000 Still Image Coding System : An Overview", IEEE Transaction on Consumer Electronic, Vol.46, No.4, PP.1103-1127, Nov 2000.

[6] Chung-Jr Lian, "JPEG2000 Embedded Block Coding with Optimized Truncation", Sep 2000.

[7] David Taubman, Erik Ordentlich, Marcelo Weinberger, "Embedded Block Coding in JPEG2000" 2001

[8] Jin Li, Shawmin Lei, "An Embedded Still Image Coder With Rate-Distortion Optimization", IEEE Transaction on Image Processing, Vol. 8, No.7, July 1999.

[9] Martin Boliek, Charilaos Christopoulos, Eric Majani, "JPEG2000 Part I Final Committee Draft Version 1.0" ISO/IEC JTC 1/SC 29/WG1 N1646R

[10] Majid Rabbani, Rajan Joshi, " An Overview of the JPEG 2000 still image compression standard", Signal Processing; Image Communication 17 (2002) 3-48.

[11] Michael D.Adams, "The JPEG2000 Still Image Compression Standard" ISO/IEC JTC 1/SC 29/WG1 N2412