JPEG2000 CODEC을 위한 Entropy 코딩 알고리즘의 VLSI 설계

준회원 이 경 민*, 오 경 호*, 정 일 환*, 정회원 김 영 민*

A VLSI Design of Entropy Coding Algorithm for JPEG2000 CODEC

Kyoung-min Lee*, Kyoung-Ho Oh*, Il-Hwan Jung* ***Associate Members***
Young-Min Kim* ***Regular Members***

요 약

본 논문은 차세대 정지영상 압축방식인 JPEG2000 코덱의 엔트로피 코딩 알고리즘의 하드웨어적 구조를 제안하고, 설계하였다. 구현된 엔트로피 코더는 컨텍스트 기반의 산술부호화기로서 컨텍스트 추출부(CE)와 산술부호화기(AC)로 구성된다. CE는 각 코딩패스에서 코딩에 참여하지 않는 샘플은 skipping 함으로써 동작속도를 향상시켰으며, AC는 MQ coder에 기반을 둔 산술부호화기로서, 곱셈과 나눗셈 연산대신 단순 가감산과 shift 연산만을 이용하여 구조를 단순화하고 연산량을 줄임으로써 동작속도를 향상시켰다. 설계된 엔트로피 코더는 VHDL 모델링후 Xilinx FPGA technology를 이용하여 합성한 후 동작을 검증하였으며, 30MHz의 동작속도를 보인다.

ABSTRACT

In this paper, we design an efficient VLSI architecture of entropy coding algorithm in JPEG2000. Entropy coder is a context-based binary arithmetic encoder, and composed of a Context Extractor(CE) and an Arithmetic Coder(AC). We speed-up CE by skipping no-operation bits in coding passes, and AC is to be performed based on MQ coder. Because of using Qe value associated with each allowed context and probability estimation, MQ coder is a multiplication free coder that reduces computation loads and makes simple the structure of arithmetic coder. We have developed and synthesized the VHDL models of CE and AC pairs using Xilinx FPGA technology. The proposed architecture operates up to 30MHz.

Keywords : JPEG2000, EBCOT, Context formation, Entropy coder, Arithmetic coder

I. 서 론

이전까지의 정지영상 압축방법의 표준인 JPEG은 저비트에서의 화질열화, 고압축에서의 블록화 현상 등의 단점에의해 새로운 정지영상 압축방법 표준이 요구되었고, 이에 차세대 정지영상 압축부호화 방식으로서 JPEG2000의 표준화[1]가 완료되었다.

JPEG2000은 기존의 정지영상 압축 방식인 JPEG에 비해 뛰어난 압축성능을 보이며 JPEG에서 이용하지 못했던 여러가지 새로운 특징들을 제공한다.[2] JPEG과 JPEG2000의 가장 큰 차이점은 DCT(Discrete Cosine Transform)와 Huff- man코딩 대신에 DWT(Discrete Wavelet Trans- form)와 컨텍스트기반 산술부호화를 사용한다는 점이다.

JPEG2000은 전체 영상이미지를 tile화하여 각

* 전남대학교 전자공학과(wilbgood@chollian.net)
논문번호 : 030268-0620, 접수일자 : 2003년 6월 20일

각의 tile component를 웨이블렛 변환 및 양자화하고, 웨이블렛 계수로 구성된 양자화된 부밴드(subband)를 코드블록으로 나눈후, 각각의 코드블록을 독립적으로 엔트로피 코딩을 수행한다.

그림 1은 컨텍스트 추출부(Context Extractor, CE)와 산술부호화기(Arithmetic Coder, AC)로 구성되는 엔트로피 코더의 블록도이다.

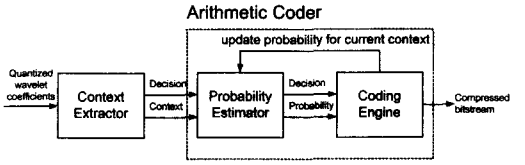


그림 1. JPEG2000의 Entropy Coder 블록도

CE는 3가지의 코딩패스와 4가지의 코딩 operation에 의해 컨텍스트와 디시젼을 출력하고, AC는 입력된 컨텍스트에 대한 확률값의 구간갱신과 코드값 연산을 통해 압축된 코드인 비트스트림을 출력한다.

3 components(RGB)에 대한 JPEG2000 압축 알고리즘의 run time profile을 살펴보면 전체 연산량중 웨이블렛 변환과 엔트로피 코딩부가 차지하는 비율은 lossless 코딩의 경우 웨이블렛 변환은 11.9%, 엔트로피 코딩부는 69.29%이며, lossy 코딩의 경우 웨이블렛 변환은 23.97%, 엔트로피 코딩부는 43.83%이다.[3] 따라서, 일정한 화질로 영상을 빠른 시간내에 큰 압축률로 부호화하기 위해서는 엔트로피 코딩 알고리즘의 하드웨어 구현은 필수적이다. JPEG2000의 conventional 엔트로피 코딩 알고리즘은 3개의 코딩패스가 요구되어지며, 패스 코딩과정이 대부분의 연산을 차지하기 때문에 no operation 픽셀을 skipping 함으로써 연산량을 줄이는 PS(Pixel Skipping), 3개의 패스 코딩을 하나의 패스 코딩으로 통합하여 처리시간을 단축하는 Pass-Parallel Context modeling 구조등이 현재 연구되어지고 있다.[3][4]

본 논문에서는 JPEG2000의 서브모듈로써 이용가능하도록 CE와 AC로 구성되는 엔트로피 코더를 설계하였다. CE는 각 코딩패스에서 코딩에 참여하지 않는(need not to be coded) 샘플은 skipping함으로써 동작속도를 향상시켰으며, AC는 컨텍스트에 따른 적응 확률예측 테이블을 사용하고, 또한 32-bit 부동소수점 곱셈과 나눗셈 연산대신 단순 가·감산과 shift 연산만을 이용하여 동작속도 향상 및 구조를 단순화하였다.

설계된 각 블록은 VHDL 모델링 후, Xilinx FPGA technology를 이용하여 합성을 수행하고 동작을 검증하였다. 본 논문의 구성은 Ⅱ장과 Ⅲ장에서 각각 양자화된 웨이블렛 계수로부터 컨텍스트를 추출하는 컨텍스트 추출부와 MQ coder에 기반을 둔 단순화된 산술부호화기의 구조에 대해 설명하고, Ⅳ장에서는 설계된 엔트로피 코더의 동작 검증 및 성능을 평가한 후, Ⅴ장에서 결론을 맺는다.

Ⅱ. 컨텍스트 추출부(CE)

CE는 웨이블렛 변환과 양자화된 영상이미지의 각 부대역별 코드블록내의 데이터를 bit-plane으로 나눈 후, 부호화하고자하는 샘플과 주위 샘플들의 상태정보와 값을 참고하여 3가지 코딩패스(Significance Propagation Pass, Magnitude Refinement Pass, Cleanup Pass)와 4가지 코딩 operation(Zero Coding, Sign Coding, Magnitude Refinement Coding, Run-Length Coding)으로 컨텍스트를 추출하고, 그룹화함으로써 나중에 수행될 산술부호화 과정의 데이터량을 최대한 줄이기 위해 사용된다. 코드블록내의 모든 bit-plane은 일정한 순서에 의해 3가지 패스를 통과하며, bit-plane 내의 각각의 샘플들은 3가지 패스를 통과하는 동안 단 한번 코딩된다.

본 논문에서는 웨이블렛 변환과 양자화된 영상이미지를 8×8 크기의 코드블록 단위로 tile화하여 컨텍스트를 추출하였다. 코드블록내의 하나의 픽셀은 1비트의 부호비트(sign bit)와 7비트의 크기비트(7비트의 샘플)로 구성되며, 부호 bit-plane을 제외한 나머지 bit-plane에서 코딩이 이루어진다. 코드블록에서 샘플의 스캔은 4비트씩 column단위(stripe)로 왼쪽에서 오른쪽으로 수행되며, 그림 2는 샘플의 스캔순서를 보여준다.

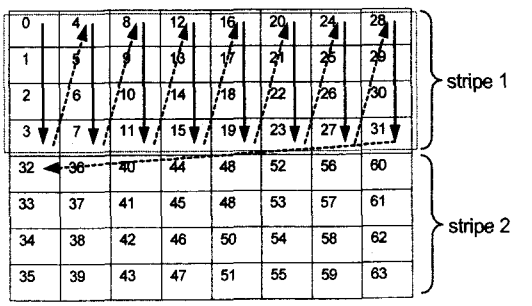


그림 2. 코드블록의 샘플 스캔 순서

1. 코딩 패스(Coding Pass)

CE에서 현재의 샘플이 어떤 코딩패스에서 코딩될 것인지를 결정하기 위해서는 현재 샘플과 주위 샘플들이 갖는 4개의 상태정보 비트들이 참조되며, 각각의 상태정보 비트에 대한 설명은 다음과 같다.

- *Significance bit*(σ) : 새로운 코드블록이 시작될 때 '0'으로 초기화되며, 픽셀의 샘플값 이 처음으로 '1'이 발견되는 bit-plane에서 '1'로 갱신된다.
- *Visited once bit*(ν) : 새로운 bit-plane이 시작될 때 '0'으로 초기화되며, 현재의 샘플이 3가지 코딩패스 중에서 코딩 operation이 수행되면 '1'로 갱신되어 다음 코딩패스에서는 코딩패스 조건과 코딩 operation을 skipping함으로써 연산량을 줄이는데 사용된다.
- *Magnitude refinement coded bit*(μ) : 새로운 코드블록이 시작될 때 '0'으로 초기화되며, 픽셀의 샘플값이 처음으로 Magnitude refinement pass에서 코딩 operation이 수행되면 '1'로 갱신된다.
- *Sign bit*(ζ) : 픽셀의 부호비트로서 '0'일 경우 양의 값을 '1'의 경우는 음의값을 나타낸다.

(1) Significance Propagation Pass(SPP)

각 bit-plane에 대한 첫번째 코딩패스로 현재 bit-plane을 처리하는 과정에서 샘플의 부호정보와 중요도를 전달하여 non zero 컨텍스트를 그룹화한다. 그림 3은 Significance Propagation Pass의 순서도로서 현재 샘플의 상태정보가 무효화(insigni- ficant, σ=0)하고, 주위 8개의 샘플들 중 적어도 하나이상 유효화(significant, σ= 1)하다는 패스 조건을 만족하면, 현재 부호화하고자하는 샘플값이 '0'일 경우 Zero Coding만 수행하고 '1'일 경우 Zero Coding과 Sign Coding을 차례로 수행하여 컨텍스트(CX)와 디시젼(D)을 출력한다. 만약 패스조건을 만족하지 않으면 다음 샘플의 패스조건을 판별한다.

(2) Magnitude Refinement Pass(MRP)

각 bit-plane의 두번째 코딩패스로 현재 bit-plane을 처리하는 과정에서 샘플의 크기에 관

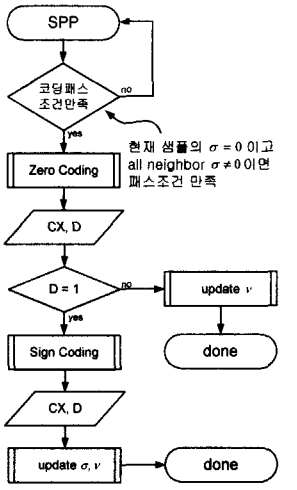


그림 3. Significance Propagation Pass 순서도

한 정보를 갖는 컨텍스트를 그룹화한다. 그림 4는 Magnitude Refinement Pass의 순서도로서 현재 샘플의 σ가 '1'이고 이전 코딩패스(SPP)에서 코딩이 수행되지 않았거나(ν=0), 이전 bit- plane에서 Magnitude Refinement Coding이 발생(μ=1)했다는 패스조건을 만족하면, 현재 샘플값에 관계없이 Magnitude Refinement Coding을 수행하여 컨텍스트와 디시젼을 출력한다. 만약 패스조건을 만족하지 않으면 다음 샘플의 패스조건을 판별한다.

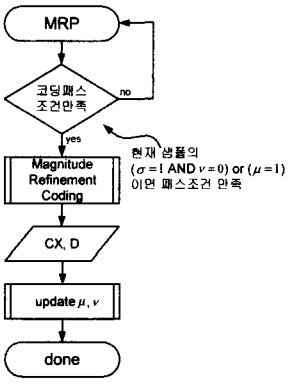


그림 4. Magnitude Refinement Pass 순서도

(3) Cleanup Pass(CP)

각 bit-plane에 대한 마지막 코딩패스로 현재 bit-plane을 처리하는 과정에서 SPP과 MRP에 의해 처리되지 않은 샘플에 대해 실행되어 zero 컨텍스트를 그룹화한다. 그림 5는 Cleanup Pass

의 순서도로서 한 열(row)의 4개의 샘플을 그룹화한 후, 그룹코딩 조건 즉, 그룹내의 샘플들과 그룹주변 샘플들의 σ가 모두 '0'이면 그룹코딩을 수행하고, 적어도 하나이상의 샘플의 σ가 '1'이면 그룹내의 샘플들을 각각 개별코딩을 한다. 그룹코딩은 Run-Length Coding를 의미하며 한 그룹을 구성하는 샘플값이 전부 '0'인 경우와 적어도 하나이상의 '1'이 존재하는 경우로 나누어 수행된다. 개별코딩은 SPP의 경우와 같다.

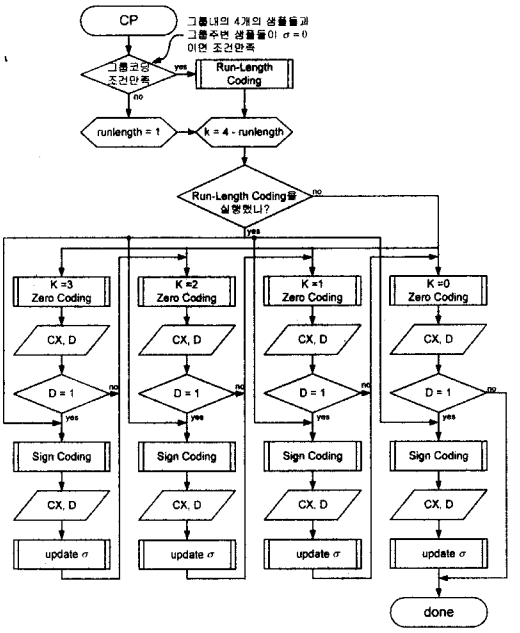


그림 5. Cleanup Pass 순서도

2. 코딩 Operations

부호화하고자 하는 샘플값과 주위샘플들의 상태정보를 참고하여 4가지 컨텍스트 생성규칙을 코딩패스마다 수행한다. Zero Coding에서 9개, Sign Coding에서 5개, Magnitude Refinement Coding에서 3개, 그리고 Run-Length Coding에서 2개로 총 19가지의 컨텍스트를 추출하여 산술부호화기로 전달한다.

(1) Zero Coding(ZC)

현재 샘플의 값이 '0'일 경우 ZC만 수행하며, '1'일 경우 ZC를 수행후 제어를 Sign Coding으로 넘겨 코딩을 수행한다. 또한 ZC는 현재 샘플값을 부호화하기 위해 이웃한 주변 8개 샘플들의 상태정보(수평(h), 수직(v)과 대각선(d)의 각 σ성분)과 현재 샘플이 속해있는 부대역의 특성을 참고하여 표 1에서 보여지는 9개의 컨텍스트를 생성한다. 디시젼은 현재 부호화하고자 하는 샘플의 값이다.

표 1. ZC로 추출되는 컨텍스트와 서브밴드별 h ,v, d조건

LL and LH subband				HL subband				HH subband		
A	B	C	CX	A	B	C	CX	C	D	CX
2	x	x	8	x	2	x	8	≥3	x	8
1	≥1	x	7	≥1	1	x	7	2	≥1	7
1	0	≥1	6	0	1	≥1	6	2	0	6
1	0	0	5	0	1	0	5	1	≥2	5
0	2	x	4	2	0	x	4	1	1	4
0	1	x	3	1	0	x	3	1	0	3
0	0	≥2	2	0	0	≥2	2	0	≥2	2
0	0	1	1	0	0	1	1	0	1	1
0	0	0	0	0	0	0	0	0	0	0

$(A=\sum h_i(\mathbf{0}),\ B=\sum v_i(\mathbf{0}),\ C=\sum d_i(\mathbf{0}),\ D=\sum(h_i(\mathbf{0})+v_i(\mathbf{0}))$

(2) Sign Coding(SC)

SC는 샘플의 부호비트를 코딩하는 방법으로 부호를 나타내는 컨텍스트를 출력시키기 때문에 하나의 코드블록에서 한번만 수행된다. 현재 부호화하고자 하는 샘플의 수평과 수직방향에 인접하는 4개의 주위샘플들의 상태정보(σ)와 부호비트(ζ)를 참고하여 표 2에서 보여지는 수평과 수직방향의 contribution을 구한 후, 각각 contribution에 해당하는 5개의 컨텍스트와 식 (1)로서 표현되는 디시젼을 출력한다. 표 3은 SC에서 추출되는 컨텍스트와 contribution조건을 나타낸다.

$$D = signbit(\zeta) \otimes Xorbit \qquad (1)$$

표 2. 수평과 수직방향의 상태정보에 따른 contribution

v_0(or h_0)의 상태정보	v_1(or h_1)의 상태정보	v(or h) contribution
σ= 1, ζ= 1	σ= 1, ζ= 1	1
σ= 1, ζ= 0	σ= 1, ζ= 1	0
σ= 0, ζ= x	σ= 1, ζ= 1	1
σ= 1, ζ= 1	σ= 1, ζ= 0	0
σ= 1, ζ= 0	σ= 1, ζ= 0	-1
σ= 0, ζ= x	σ= 1, ζ= 0	-1
σ= 1, ζ= 1	σ= 0, ζ= x	1
σ= 1, ζ= 0	σ= 0, ζ= x	-1
σ= 0, ζ= x	σ= 0, ζ= x	0

(3) Magnitude Refinement Coding (MRC)

샘플이 '1'의 값을 갖는 경우, 그 샘플의 중요도는 매우 커지게 된다. 따라서 하위 bit-plane의 샘플값들의 중요도도 그만큼 증가하게 되는데, 이런 샘플을 코딩하는 방법이 바로 MRC이다. 추출되는 컨텍스트는 모두 3가지이며, 현재 샘플의 상태정보(μ)와 주위 샘플들의 상태정보(σ)의 합을 이용하여 컨텍스트를 생성한다. 표 4는 MRC에서 추출되는 컨텍스트와 주변정보 조건을 나타낸다.

표 3. SC에서 추출되는 컨텍스트와 contribution 조건

h contribution	v contribution	Xorbit	CX
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

표 4. MRC에서 추출되는 컨텍스트와 주변정보 조건

$\sum h_i(\sigma)+\sum v_i(\sigma)+\sum d_i(\sigma)$	현재 샘플의 상태정보(μ)	CX
x	1	16
≥1	0	15
0	0	14

(4) Run-Length Coding (RLC)

RLC는 한 열의 4개 샘플을 하나의 그룹으로 묶어서 컨텍스트와 디시젼을 출력시키는 방법으로 그룹내의 모든 샘플의 σ는 '0'이어야한다. 그룹내의 모든 샘플값이 '0'이면 컨텍스트 17과, 디시젼 '0'을 출력하고, 샘플값 중 적어도 하나 이상의 값이 '1'을 갖는다면 컨텍스트 '17'과, 디시젼 '1'을 출력한 후, 그룹내에서 처음으로 발견되는 '1'의 위치를 알리기 위해 UNIFORM 컨텍스트 18과 UNIFORM 디시젼을 출력시킨다. 또한 발견된 샘플값 '1'의 코딩을 하기위해 SC로 제어를 넘기며, 다음 위치의 샘플들은 그 값에 따라 ZC와 SC를 수행한다. 예를들어 그룹을 형성한 샘플들의 값이 "0010"이라면, 먼저 컨텍스트(17), 디시젼(1)을 출력시킨 후, 처음 발견되는 '1'의 위치가 그룹내에서 3번째에 위치하고 있기 때문에 2 클럭 주기동안 UNIFORM 컨텍스트 (18)과 UNIFORM 디시젼(10)을 출력시킨후, 샘플값 '1'에 대한 코딩은 SC에 제어를 넘겨 컨텍스트와 디시젼을 출력시킨다. 그리고 마지막에 있는 '0'은 ZC에 제어를 넘겨 컨텍스트와 디시젼을 출력시킨다.

3. CE의 하드웨어적 설계

그림 6은 CE의 top 블록도로서 8비트 크기의 입력된 영상이미지 데이터를 bit-plane으로 각각 나누어 저장하는 CBS(Code Block Splitter), 각 샘플에 대한 3가지 상태정보 비트(μ σ ν)를 저장하는 SSRs(Sample State Registers), 3개의 코딩 패스, 그리고 4개의 코딩 operation으로 구성된다.

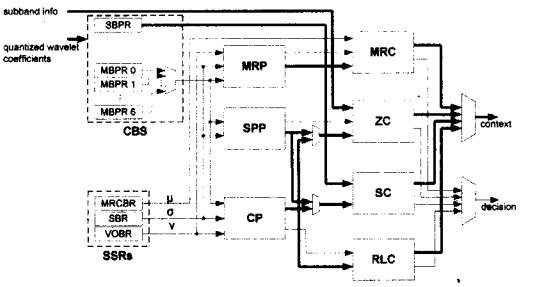


그림 6. 컨텍스트 추출부 Top블록도

CBS는 64개 픽셀의 부호비트를 저장하는 64비트 크기의 SBPR(Sign Bit-Plane Register)과 7비트 샘플값을 저장하기 위한 7개의 MBPR(Magnitude Bit-Plane Register)로 이루어지며, SSRs는 상태정보 비트 μ를 저장하기 위한 MRCBR(MR Coded Bit Register), 상태정보 비트 σ를 저장하기 위한 SBR(Significance Bit Register), 그리고 상태정보 비트 ν를 저장하기 위한 VOBR(Visited Once Bit Register)로 구성된다. 하나의 bit-plane내의 샘플의 수가 64개이므로 각 상태정보 비트 레지스터의 크기는 64비트이다. 각 bit-plane의 샘플은 상태정보 비트들을 이용해 패스 조건을 판단하여 SPP, MRP, CP의 3개 패스블록을 차례로 수행한다. 각 패스블록에서는 조건에 만족하는 샘플만을 적절한 코딩 operation에 의해 코딩한 후, 디시젼과 컨텍스트를 출력시키며, 상태정보 비트들을 갱신한다. 갱신된 ν는 이전 패스에서 샘플의 코딩여부에 대한 정보를 가지고 있기 때문에, 다음 패스(MRP 또는 CP)

에서 패스조건의 판단없이 상태정보 비트만을 검사함으로써 샘플을 skipping하여 연산속도의 향상을 가져온다.

III. 산술부호화기 (AC)

본 논문에서 사용된 산술부호화기는 MQ coder에 기반을 둔 이진 적응 산술부호화기로서, 각 샘플간의 상관관계를 이용해 추출된 컨텍스트에 기반한 통계적인 확률예측 테이블을 이용하여 적응적으로 확률을 추정하는 방법을 사용한다. 부호화는 누적확률구간 [0, 1)내에서 현재의 부호화 구간을 2개의 부구간(sub-interval) 즉, LPS(Less Probable Symbol)에 대한 구간과 MPS (More Probable Symbol)에 대한 구간으로 분할하는 절차를 반복적으로 수행하며, 입력 심볼들에 대해 누적확률 분포를 산술적으로 계산하여 코드워드를 생성한다.[5]

AC는 CE로부터 전달되는 컨텍스트와 연관된 47개의 각기 다른 인덱스(index, I)에 대한 확률예측값 Qe(I(CX))을 제공하는 확률예측기, 구간분할 연산을 담당하는 구간갱신 레지스터, 실제적인 압축코드 생성을 담당하는 코드 레지스터, 그리고 컨트롤러로써 구성된다.

1. 확률예측기(Probability Estimator)

확률예측의 정확도에 따라 AC의 압축률과 동작속도를 결정해주는 부분으로서, 입력된 컨텍스트와 연관된 47개의 각기 다른 인덱스에 대한 확률예측값을 나열해 놓은 표이다.[1][6] 구간 부분할(subdivision)은 항상 이러한 47개의 인덱스에서 이루어진다.

설계된 확률예측기 블록은 2개의 lookup 테이블로 구성된다. 각각의 컨텍스트와 연관된 인덱스값과 MPS값을 출력하는 인덱스 lookup 테이블과 인덱스에 대한 확률예측값 Qe, NMPS(Next MPS), NLPS (Next LPS), 그리고 switch 값을 출력하는 Qe lookup 테이블로 이루어져 있다. NMPS와 NLPS는 컨텍스트와 연관된 인덱스값을 갱신하는데 사용되며, switch값은 입력 심볼에 따라 확률값들을 예측하는 과정에서 MPS의 확률보다 LPS의 확률이 커지는 역전현상이 발생하게 되면 MPS와 LPS의 확률값을 서로 맞바꾸어 확률예측시 발생하게 되는 오차를 개선시키는데 사용된다. 표 5는 컨텍스트에 대한 초기 인덱스와 MPS값을 나타낸다.

표 5. 컨텍스트에 대한 초기 인덱스와 MPS값

context	index	MPS
0	3	0
1	4	0
2	0	0
3	0	0
4	0	0
5	0	0
6	0	0
7	0	0
8	0	0
9	0	0
10	0	0
11	0	0
12	0	0
13	0	0
14	0	0
15	0	0
16	0	0
17	0	0
18	46	0

그림 7은 확률예측 순서도로서 입력된 심볼값인 디시젼과 컨텍스트에 대한 MPS값인 MPS(CX)를 비교하여 MPS 구간내에 있지않으면, 즉 입력된 심볼이 LPS인 경우는 CODELPS를 수행한다.

CODE LPS는 컨텍스트에 대한 인덱스 값을 갱신하기 위하여 Qe lookup 테이블로부터 결정되는 NLPS를 사용한다. CODEMPS는 입력된 심볼이 MPS인 경우이며 NMPS값으로 인덱스를 갱신한다.

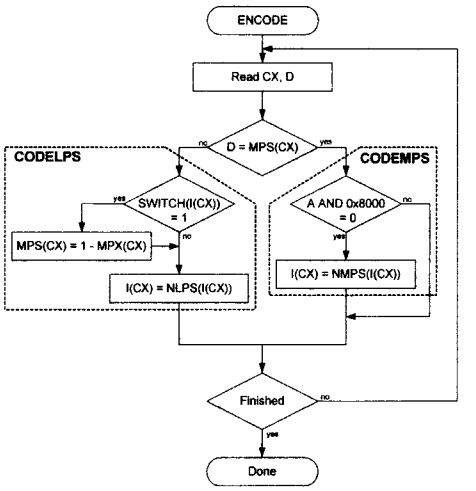


그림 7. 확률예측 순서도

그림 8은 설계된 확률예측기의 블록도이다. 컨텍스트에 대한 인덱스값(6비트)과 MPS를 저장하는 인덱스 lookup 테이블은 19개의 7비트 레지스터로 구성된다. 인덱스는 Qe lookup 테이블로 입력되어

16비트 크기의 Qe, 5비트 크기의 NMPS, NLPS, 그리고 1비트 크기의 switch값을 출력하고 Qe값은 구간 재조정 및 코드값을 연산하는데 사용되어진다.

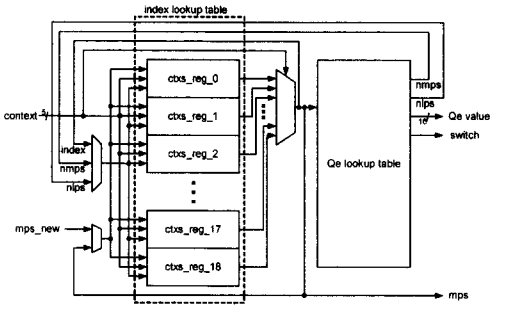


그림 8. 확률예측기 블록도

2. 구간갱신 레지스터 (Interval Renew Register)

반복적인 확률구간 분할과정은 이진 산술부호화의 기본알고리즘 중의 하나이다. 일반적인 알고리즘에서는 구간분할을 위해 곱셈과 나눗셈 연산을 행한다. 하지만 이러한 연산은 많은 처리시간과 하드웨어 면적의 증가를 가져오기 때문에 이를 피하기 위해 본 논문에서는 덧셈기와 barrel shifter만을 이용하여 구간분할 연산과 갱신을 행하도록 설계하였다.

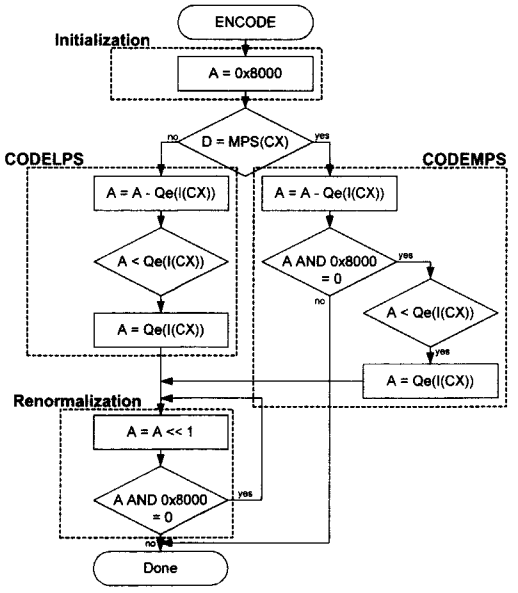


그림 9. 구간갱신 블록의 순서도

그림 9는 구간갱신 블록의 순서도로서, 새로운 코드블록이 시작될 때마다 구간갱신 레지스터(A register)는 0x8000H(decimal 0.75)으로 초기화되며, A의 값은 항상 $0.75 \le A < 1.5$를 유지해야 한다. 따라서, A의 하위구간 계산값이 0×8000H 이하가 되면 비교기와 barrel shifter를 사용하여 좌로 1비트 shift 연산을 통해 구간값이 0.75이상이 되도록 재정규화(Renormalization)를 수행한다. 입력 심볼이 LPS인 경우 구간갱신은 식(2), 식(3)으로 표현되며, 입력 심볼이 MPS인 경우 구간갱신은 식(5), 식(6)으로써 정의된다.

○ 입력 심볼이 LPS인 경우

$$A_{new} = A_{old} - Qe_{old} \quad (\because when\ A < Qe) \qquad (2)$$

$$A_{new} = Qe_{old} \quad (\because when\ A \ge Qe) \qquad (3)$$

$$A_{new} = Lshift(A_{old}) \quad (\because when\ A < 0\times8000) \qquad (4)$$

○ 입력 심볼이 MPS인 경우

$$A_{new} = Qe_{old} \quad (\because when\ A < Qe) \qquad (5)$$

$$A_{new} = A_{old} - Qe_{old} \quad (\because when\ A \ge Qe) \qquad (6)$$

$$A_{new} = Lshift(A_{old}) \quad (\because when\ A < 0\times8000) \qquad (7)$$

여기서 A_{new}는 새로운 구간값을 나타내고 A_{old}는 이전의 구간값, Qe_{old}는 이전의 확률값을 뜻한다. $Lshift(A_{old})$는 이전 구간값을 1비트 좌로 shift하는 것을 의미한다. 그림 10은 구간갱신 레지스터의 블록도로서, 감산기는 Qe값을 2의 보수로 변환후 덧셈연산을 수행하는 16비트 CLA(Carry Lookahead Adder)로써 구현하였다.

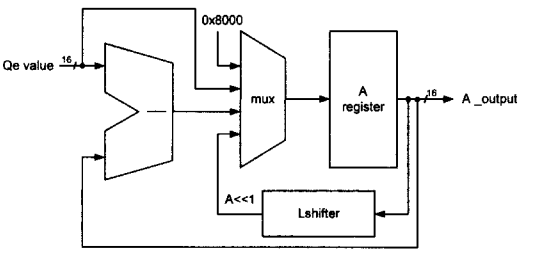


그림 10. 구간갱신 레지스터 블록도

3. 코드 레지스터(Code Register)

실질적인 코드값을 산출하고 입력데이터의 압축을 담당하는 블록으로서, 구간분할 연산과 마찬가지로 곱셈과 나눗셈 연산을 통해 이루어지는데 단순 덧셈과 shift 연산만을 이용하여 동

작속도 향상 및 구조를 단순화 하였다.

그림 11은 코드 연산부의 순서도로서 새로운 코드블록이 시작할때마다 코드 레지스터(C register)를 0으로 초기화하며, 입력 심볼이 LPS 및 MPS인 경우에 따라서 각각 식(8)부터 식(13)에 정의된 연산을 수행한다.

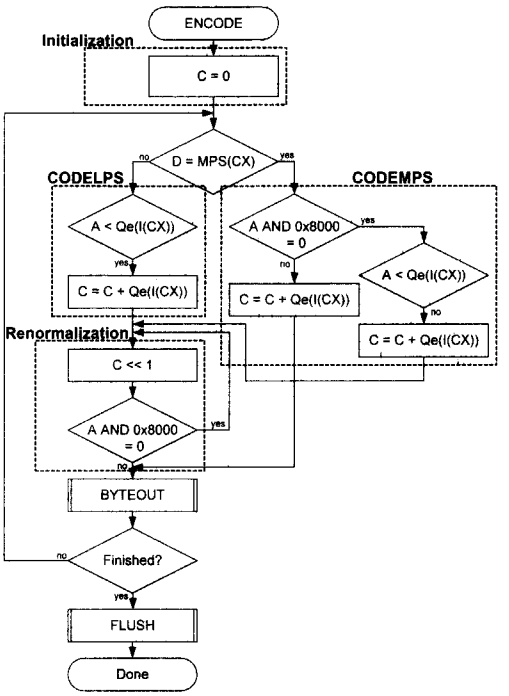


그림 11. 코드 연산부의 순서도

연산 수행중 구간갱신 레지스터인 A의 값이 0x8000H이하가 되면 재정규화 과정을 통하여 C를 좌로 1비트씩 shift시키며, 압축된 코드스트림의 1바이트가 전송 및 저장을 위해 BYTE OUT 연산을 통해 C로부터 제거된다.

BYTEOUT 연산은 32비트 레지스터인 C의 20번째 비트부터 상위 8비트 데이터를 바이트 버퍼 레지스터(B register)에 출력시키며, 이때 출력된 코드값이 0xFFH이면 이 값은 특별한 의미를 갖는 값이므로(디코더에서 코딩의 종료로 인식하는 값) 다음에 출력되는 바이트는 bit stuffing(C의 21번째 비트부터 상위 8비트 데이터를 출력)을 통해 값의 범위가 0x00H부터 0x8FH가 되도록 코드값을 변환하여 출력한다. 이는 JPEG2000에서의 중요한 코드스트림 마커가 0xFF90H부터 0xFFFFH 범위에 존재하기 때문이다. FLUSH는 ENCODE의 모든 연산을 종료하고 터미네이션 마커를 생성시킨다.

○ 입력 심볼이 LPS인 경우

$$C_{new} = C_{old} + Qe_{old} \quad (\because when\ A < Qe) \tag{8}$$
$$C_{new} = C_{old} \quad (\because when\ A \geq Qe) \tag{9}$$
$$C_{new} = Lshift(C_{old}) \quad (\because when\ A < 0\times8000) \tag{10}$$

○ 입력 심볼이 MPS인 경우

$$C_{new} = C_{old} \quad (\because when\ A < Qe) \tag{11}$$
$$C_{new} = C_{old} + Qe_{old} \quad (\because when\ A \geq Qe) \tag{12}$$
$$C_{new} = Lshift(C_{old}) \quad (\because when\ A < 0\times8000) \tag{13}$$

C_{new}는 갱신된 코드값을 나타내고 C_{old}는 이전의 코드값, Qe_{old}는 이전의 확률값을 나타낸다. Lshift(C_{old})는 이전 코드값을 1비트 좌로 shift하는 것을 의미한다.

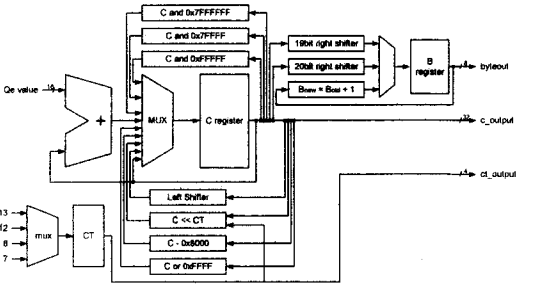


그림 12. 코드 레지스터 블록도

그림 12는 설계된 코드 레지스터의 블록도로서, 코드값 연산 및 압축된 코드스트림을 저장하는 32비트 레지스터인 C, C로부터 1바이트가 출력할 준비가 되었을 때를 지시하는 4비트 다운카운터인 CT, 그리고 8비트 크기의 바이트 버퍼로 구성된다. 새로운 코드블록이 시작될때마다 CT는 C를 구성하는 요소인 3비트 크기의 spacer bits[1]를 제거하기위해 12 또는 13(bit stuffing이 수행될 경우)으로 초기화한다. 부호화 과정에서는 8또는 7(bit stuffing이 수행될 경우)로 지정되어 코드값 연산중 재정규화를 수행할 때마다 1씩 다운카운트하여 CT값이 0이면 C로부터 1바이트의 코드스트림을 출력시킨다.

4. AC의 하드웨어적 설계

그림 13은 확률예측기, 구간갱신 레지스터, 그리고 코드 레지스터로 구성되는 AC의 블록도이다. 그림에서 점선은 컨트롤러로부터 전달되는 제어신호이고 실선은 데이터의 흐름을 나타낸다. CE로부터 컨텍스트와 디시젼이 입력되면 확률예측기로부터 전달되는 컨텍스트와 연관된 확률값이 구간분할 연산 및 코드값 연산에 사용되어지고, 압축된 코드스트림의 1바이트가 코드 레지스터로부터 출력된다.

TempC 레지스터에 저장된 C와 A값의 합은 FLUSH연산에서 터미네이션 마커를 생성시키는데 사용되어진다.

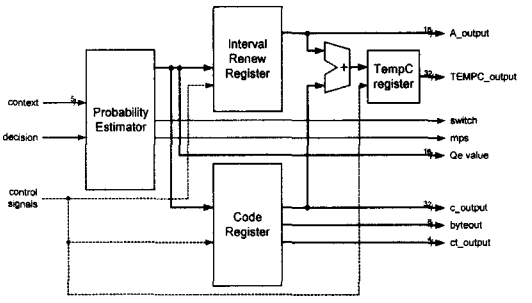
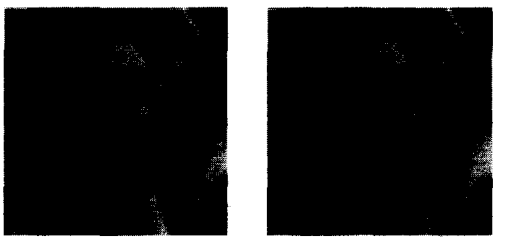


그림 13. AC의 블록도

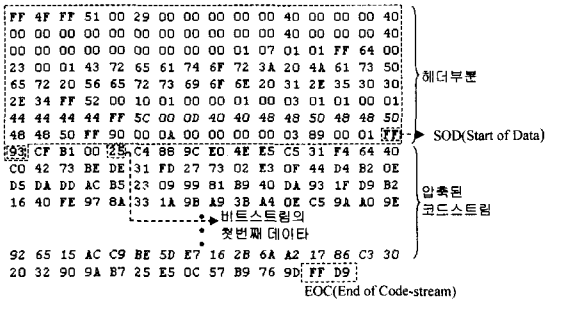
Ⅳ. Simulation 및 동작검증

설계된 CE와 AC는 VHDL 모델링 후, Xilinx FPGA Device(XCV2000E-6BG560C) 라이브러리를 이용하여 Synopsys에서 합성한 후 동작을 검증하였다. CE의 게이트수는 27,525 (2-input NAND 기준) 게이트, AC의 게이트수는 6,006 게이트, 그리고 하나의 FIFO(64-word by 8-bit)가 사용된다. 설계된 회로는 30MHz의 주파수에서 동작을 검증하였으며, 시뮬레이션을 위한 이미지는 Lena 영상(64x64, 4096byte)을 사용하였다.

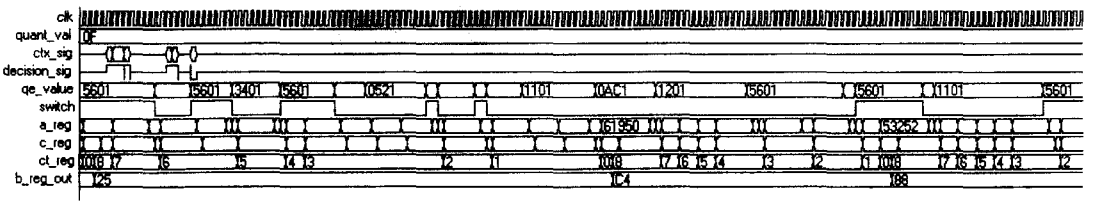
그림 14는 설계된 구조의 동작을 검증하기 위해 영상 데이터를 이용한 시뮬레이션 결과이다. (a)는 64x64크기의 원영상이며, (b)는 원 영상을 software를 이용하여 5배 압축시킨 결과이다. (c)는 (b)의 16진 코드로서, 압축된 비트스트림 데이터의 시작을 지시하는 SOD 마커코드인 0xFF93H가 나타나며 이후에 첫 번째 압축된 1바이트의 데이터 0x25H가 위치한다. (d)는 설계된 하드웨어의 시뮬레이션 결과로서, 입력데이터('*quant-val*')는 원영상으로부터 C program을 이용하여 추출된 양자화된 웨이블렛 계수이며, 다운카운터인 CT레지스터의 출력값('*ct_reg*')이 0이 될 때 1바이트의 압축된 코드가 바이트버퍼 레지스터('*b_reg_out*')로부터 출력되며 software로 압축한 결과와 일치함을 알수 있다.



(a) (b)



(c)



(d)

그림 14. 영상데이타 시뮬레이션 결과

(a) 원영상 (b) 5배 압축된 영상 (c) 5배 압축된 영상 file의 16진 코드 (d) 설계된 하드웨어의 시뮬레이션

V. 결 론

본 논문에서는 JPEG2000 CODEC의 Entropy Coder의 구조를 제안하고 설계하였다. 설계된 구조는 Xilinx FPGA technology를 이용하여 검증하였고, 30MHz의 주파수로 동작된다. 기존의 straight-forward 방식에서는 각 코딩패스마다 모든 샘플의 코딩 여부에 대한 조건판단을 수행하지만, 본 논문에서는 상태정보 비트 v를 검사함으로써 이전 코딩 패스에서 코딩된 샘플들은(v= 1) 패스조건의 판단없이 skipping하여 연산속도의 향상을 가져온다. AC는 컨텍스트에 따른 적응 확률예측 테이블을 사용하여 구간분할 및 코드값 연산에 사용되는 곱셈과 나눗셈 연산 대신 단순 가·감산과 shift연산만을 이용하여 구조를 단순화하고 동작속도를 향상시켰다. 본 논문에서 제안하고 설계한 구조는 JPEG2000 CODEC 알고리즘의 서브모듈로써 응용하여 적용할 수 있으며, 앞으로 진행해야 할 연구과제는 전송 및 저장을 위한 packetization과 trunca- tion을 담당하는 Tier-2와 결합된 구조의 연구가 필요하다.

참 고 문 헌

[1] ISO/IEC 15444-1, "Information technology-JPEG2000 image coding system-Part 1: Core coding system", 2001.

[2] Charilaos Christopoulos, "The JPEG2000 Still Image Coidng System:An Overview", IEEE Transactions on Consumer Electomics, Vol. 46, No. 4, pp. 1103-1127, Nov 2000.

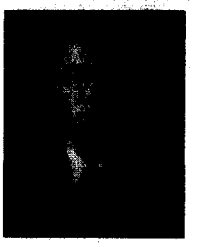
[3] K. Chen, C. Lian, H. Chen, and L. Chen, "Analysis and Architecture Design of EBCOT for JPEG2000", IEEE ISCAS-2001, Vol. 2, pp. 765-768, May 2001.

[4] Jen-Shiun Chiang, Yu-Sen Lin, and Chang-Yo Hsieh, "Efficient Pass-Parallel Architecture for EBCOT in JPEG2000", IEEE ISCAS-2002, Vol. 1, pp. 773-776, May 2002.

[5] Paul G. Howard, Jeffrey S. Vitter, "Arithmetic Coding for Data Compression", Proceedings of IEEE, Vol. 82, No. 6, pp. 857-865, Jun 1994.

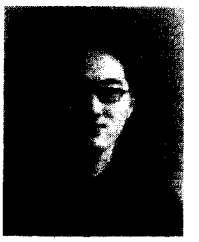
[6] David S. Taubman and Michael W. Marcellin, "JPEG2000-Image compression Fundamentals, Standards and Practice", pp. 56-77, 2001.

李 炅 珉 (準會員)



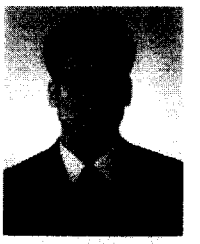
1967년 9월 29일생. 1993년 2월 전남대학교 전자공학과 졸업(공학사). 1998년 2월 전남대학교 전자공학과 졸업(공학석사). 1998년 8월~현재 전남대학교 전자공학과 대학원 박사과정. <주관심분야 : 영상처리, VLSI설계, 신경회로망등임>

吳 炅 鎬 (準會員)



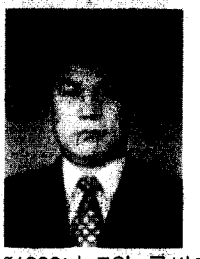
1975년 1월 24일생. 2001년 2월 전남대학교 물리학과 졸업(이학사). 2002년 3월~현재 전남대학교 전자공학과 대학원 석사과정. <주관심분야 : 영상처리, VLSI 설계>

鄭 一 桓 (準會員)



1975년 11월 13일생. 2001년 2월 조선대학교 전자공학과 졸업(공학사). 2002년 3월~현재 전남대학교 전자공학과 대학원 석사과정. <주관심분야 : 영상처리, VLSI 설계>

金 榮 民 (正會員)



1954년 4월 18일생. 1976년 2월 서울대학교 전자공학과 졸업(공학사). 1978년 2월 한국과학원 전기및전자공학과 졸업(공학석사). 1978년 3월~1979년 7월 한국선박해양 연구소(주임연구원). 1979년 8월~1982년 7월 국방과학연구소(연구원). 1986년 오하이오 주립대학교 전기공학과(공학박사). 1988년 6월~1991년 8월 한국전자통신연구원(실장). 1991년 9월~현재 전남대학교 전자컴퓨터정보통신공학부 교수. <주관심분야 영상처리, VLSI 설계, RFID, 신경회로망등임>