Encoder/Decoder

세종대학교 ITRI 연구실

정보통신공학과

하재민

목차

1. Encoder

- **단계별 확인 방법 - Prediction**

  - **함수 호출 세팅**

```
    }
    Prediction_Func(Ori_Padding_buf, BLK, Blk_Row, Blk_Col, img);
    //DCT_Func (BLK, Blk_Row, Blk_Col, img);
    //Quantization_Func(BLK/2, Blk_Row, Blk_Col, img);

    //Entropy(BLK / 2, Blk_Row, Blk_Col, img);

    //IQuantization_Func(BLK/2, Blk_Row, Blk_Col, img);
    //IDCT_Func(BLK, Blk_Row, Blk_Col, img);
    Reconstruct_Func(Rec_Padding_buf, BLK, Blk_Row, Blk_Col, img);
```

  - **MSE, PSNR**

```
MSE : 0.000000
PSNR : inf

영상 일치
```

  - **예측, 복원만 수행한 경우, MSE는 0, PSNR 무한대 ➔ Prediction 90%완성**
    - 잘못 생성한 예측 블록을 그대로 다시 잔차 블록에 더해도 MSE는 0이 나옴

1. Encoder

- **단계별 확인 방법 – Prediction, Transform, Quantization**
  - **함수 호출 세팅**

```
Prediction_Func(Ori_Padding_buf, BLK, Blk_Row, Blk_Col, img);
DCT_Func (BLK, Blk_Row, Blk_Col, img);
Quantization_Func(BLK/2, Blk_Row, Blk_Col, img);

//Entropy(BLK / 2, Blk_Row, Blk_Col, img);

IQuantization_Func(BLK/2, Blk_Row, Blk_Col, img);
IDCT_Func(BLK, Blk_Row, Blk_Col, img);
Reconstruct_Func(Rec_Padding_buf, BLK, Blk_Row, Blk_Col, img);
```

  - **MSE, PSNR**

```
MSE : 33.587673
PSNR : 32.592140

영상 불일치
```

  - **MSE, PSNR이 위와 같이 나와야함**
    - **MSE, PSNR이 예시와 다른 경우, Prediction 잘못됨**

1. Encoder

▪ 단계별 확인 방법 - Entropy

• 비트연산자

연산자	설명
&	비트 AND
¦	비트 OR
^	비트 XOR (배타적 OR, Exclusive OR)
~	비트 NOT
<<	비트를 왼쪽으로 시프트
>>	비트를 오른쪽으로 시프트
&=	비트 AND 연산 후 할당
¦=	비트 OR 연산 후 할당
^=	비트 XOR 연산 후 할당
<<=	비트를 왼쪽으로 시프트한 후 할당
>>=	비트를 오른쪽으로 시프트한 후 할당

1. Encoder

▪ 단계별 확인 방법 - Entropy

• 비트연산자

- a & b
- a | b
- a ^ b

bitwise_and_or_xor_operator.c

```c
#include <stdio.h>

int main()
{
    unsigned char num1 = 1;    // 0000 0001
    unsigned char num2 = 3;    // 0000 0011

    printf("%d\n", num1 & num2);    // 0000 0001: 01과 11을 비트 AND하면 01이 됨
    printf("%d\n", num1 | num2);    // 0000 0011: 01과 11을 비트 OR하면 11이 됨
    printf("%d\n", num1 ^ num2);    // 0000 0010: 01과 11을 비트 XOR하면 10이 됨

    return 0;
}
```

값은 십진수로 되어있지만, 생각은 이진수로 해야함

실행 결과

```
1
3
2
```

1. Encoder

- 단계별 확인 방법 - Entropy

- 비트연산자

  - &(And) 연산자 예시

```
0111 1000(120)
0001 1010(26)
_________ &
0001 1000(24)
```

  - |(Or) 연산자 예시

```
0000 0001(1)
0000 0011(3)
_________ |
0000 0011(3)
```

1. Encoder

▪ 단계별 확인 방법 - Entropy

• 비트연산자

- a << b
- a >> b

bitwise_shift_operator.c

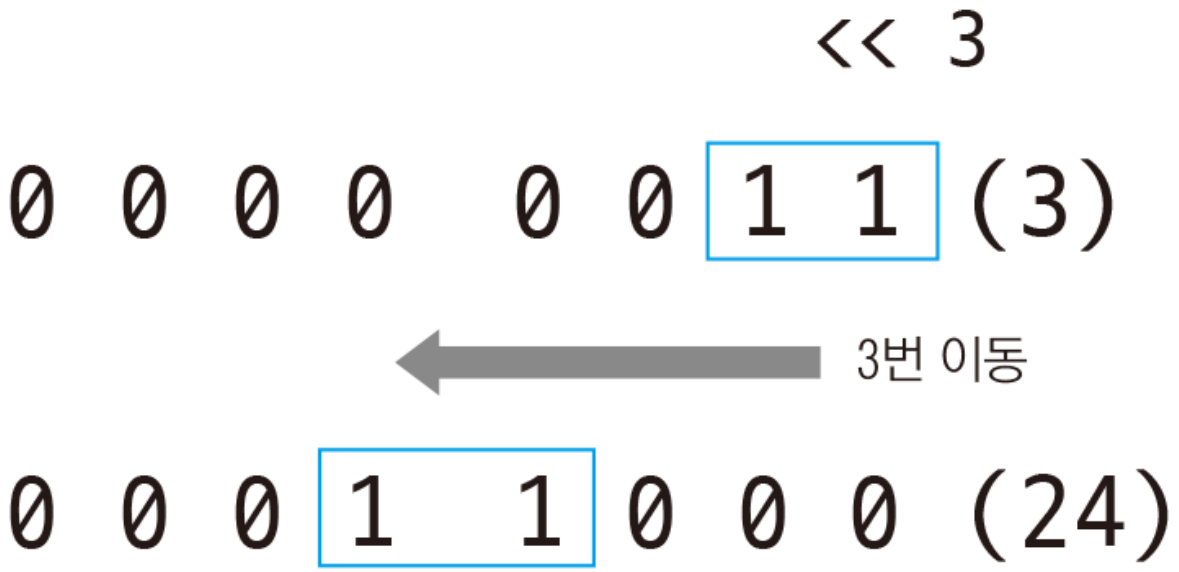
```c
#include <stdio.h>

int main()
{
    unsigned char num1 = 3;     //  3: 0000 0011
    unsigned char num2 = 24;    // 24: 0001 1000

    printf("%u\n", num1 << 3);  // 24: 0001 1000: num1의 비트 값을 왼쪽으로 3번 이동
    printf("%u\n", num2 >> 2);  //  6: 0000 0110: num2의 비트 값을 오른쪽으로 2번 이동

    return 0;
}
```

실행 결과

```
24
6
```

1. Encoder

- 단계별 확인 방법 - Entropy
  - 비트연산자
    - <<, >>(Shift) 연산자 예시



1. Encoder

- **단계별 확인 방법 - Entropy**

- **비트연산자를 이용한 심볼별 비트 할당**

```
if (ALL_Zero_flag) //0000 0000
    bitstream = 0xC0; // 이진수: 1100 0000
switch (img->info.Best_Mode)
{
case 0:
    bitstream |= 0x30; // 이진수: 11 0000
    break;
case 1:
    bitstream |= 0x0C; // 이진수: 00 1100
    break;
case 2:
    bitstream |= 0x03; // 이진수: 00 0011
    break;
case 3:
    bitstream |= 0x00; // 이진수: 00 0000
    break;
}
```

ALL_Zero_flag와 Best_Mode는
비트 할당시 자리가 정해져있음

0x 는 16진수 표현할때 사용
0x30은 0x삼십 ➔ 잘못된 생각
0x30은 0x삼공 ➔ 삼과 공을 각각
4비트로 생각해야함

1. Encoder

- 단계별 확인 방법 - Entropy

- 사실 빠져도 되는 내용……아무도 질문을…

```
if (!ALL_Zero_flag)
{
    for (int i = 0; i < BLK; i++)
    {
        for (Int j = 0; j < BLK; j++)
        {
            Int flag = (img->Quant_blk[i * BLK + j] >= 0) ? 0 : 1;
            bitstream <<= 1;
            bitstream |= flag;
        }
        if (i == 1 || i == 3)
        {
            fwrite(&bitstream, sizeof(UChar), 1, fp);
            bitstream = 0;
        }
    }

    Char bitstream2;

    Huffman(img);
```

1. Encoder

- 단계별 확인 방법 - Entropy

- 심볼별 허프만 알고리즘을 이용하여 코드워드 할당
  - Huff_codes ➔ 심볼별 코드워드 저장
  - Huff_freq ➔ 심볼별 발생 빈도수
  - Huff_ch ➔ 발생한 심볼
  - Huff_length ➔심볼별 코드워드 길이
  - Huff_Size ➔ 발생한 심볼 수

1. Encoder

- **단계별 확인 방법 - Entropy**

- **Decoder에서 동일한 허프만 트리를 만들기 위해 일부 값 전송**

  - **Huff_Size, Huff_ch , Huff_freq 전송**

```
bitstream2 = img->Huff_Size;
fwrite(&bitstream2, sizeof(Char), 1, fp);

for (Int i = 0; i < img->Huff_Size; i++)
{
    bitstream2 = img->Huff_ch[i];
    fwrite(&bitstream2, sizeof(Char), 1, fp);
    bitstream2 = img->Huff_freq[i];
    fwrite(&bitstream2, sizeof(Char), 1, fp);
}
```

1. Encoder

- **단계별 확인 방법 - Entropy**

- **양자화된 변환 블록내 계수값 전송**
  - 양자화된 변환 블록 크기 만큼 반복문 생성(0~15) ➔ 아래 내용은 반복문 내 속함
  - 현재 선택된 심볼(Quant_blk[i])과 동일한 심볼을 Huff_ch[]에서 찾음 ➔ 배열 Idx가 중요!!!!

```
Int Idx = 0;
for (Int i = 0; i < BLK * BLK; i++)
{
    for (Int j = 0; j < img->Huff_Size; j++)
    {
        if (img->Quant_blk[i] == img->Huff_ch[j])
            Idx = j;
    }
```

1. Encoder

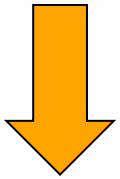
- **단계별 확인 방법 - Entropy**

- **양자화된 변환 블록내 계수값 전송**

  - **Cur_info에 현재 심볼의 코드워드를 저장**

  - **Cur_info에서 유효한 비트만 남기기 위해 MSB로 쉬프트 시킴**

현재 심볼의 코드워드 = 3, 코드워드길이 = 2
➔ 00000011 (실제 코드 워드)



MSB로 비트쉬프트
➔ 11000000 (십진수로 192)

1. Encoder

- 단계별 확인 방법 - Entropy

- 양자화된 변환 블록내 계수값 전송
  - 코드워드 길이만큼 반복문 수행
  - Bitstream 왼쪽으로 1bit 쉬프팅
  - Tmp에 cur_info의 MSB만 저장 (&연산자 이용)
  - cur_info 왼쪽으로 1bit 쉬프팅
  - Tmp 오른쪽으로 7비트 쉬프팅 (즉, LSB로 보냄)
  - Bitstream 에 TMP의 LSB만 저장
  - Bitstream에 8개의 비트가 채워진 경우 fwrite하고 Bitstream 0으로 초기화

1. Encoder

- **단계별 확인 방법 - Entropy**

- **양자화된 변환 블록내 계수값 전송**

  - **16개의 심볼을 전부 코드워드로 할당하여 비트스트림에 저장했음에도 비트가 남은 경우**

    ➔ 남은 LSB 비트들은 0으로 할당 후 저장

```
if (bit_cnt)
{
    bitstream <<= (8 - bit_cnt);
    fwrite(&bitstream, sizeof(UChar), 1, fp);
}
```

1. Encoder

- **단계별 확인 방법 - Entropy**
  - **비트스트림 크기**