이차원 QR Code에서 데이터 코드워드의 디코딩 알고리즘

박광욱[1*], 이종연[1*]
[1]충북대학교 컴퓨터교육과, [2]충북대학교 디지털정보융합학과

Algorithm of Decoding the Data Codeword in Two-Dimensional QR Code

Kwang Wook Park[1*] and Jong Yun Lee[1*]
Dept. of Computer Education, Chungbuk National University[1]
Dept. of Digital Informatics and Convergence, Chungbuk National University[2]

요 약 2차원 QR Code는 1차원 바코드의 용량, 크기, 방향에 대한 한계를 개선하였으며, 방대한량의 데이터를 압축하여 저장할 수 있는 능력을 갖추고 많은 응용분야에서 사용되고 있다. 최근에는 QR Code를 인식할 수 있는 스마트 폰의 도입과 사용의 증대로 QR Code의 도입이 급속도로 확산되었다. 하지만 다양한 정보를 저장할 수 있는 QR Code의 디코딩 기술에 대한 구체적인 문서화가 없는 상태이다. 따라서 본 논문에서는 QR code의 인코딩에 따른 디코딩 과정에 대해 구체적인 처리 절차와 알고리즘을 제시하고 예를 통해 디코딩 과정을 상세히 이해시키는데 연구목적이 있다.

• **주제어** : 이차원 바코드, 큐알코드, 바코드검증기, 부호화, 부호워드

Abstract Two-dimensional QR Code has improved greatly the limits on the capacity, size, direction of the one-dimensional bar code. And it has the capacity of compressing and storing the massive amount of data and is widely used in many applications. Recently, the two-dimensional QR Code has been spread rapidly because of introducing the smart phones and increasing the amount of using them. However, there is little documentations about decoding the QR Code in which can store the large amount of information. In this paper, therefore, we present specific processing procedures and algorithms on decoding the two-dimensional QR Code and then make us understand their decoding process by explaining some examples.

• **Key Words** : Two-dimensional Barcode, QR Code, Barcode Verifiers, Decoding, Data Codeword

1. 서론

1.1 연구동기

과학기술의 발달로 인하여 수많은 정보를 이용하고 공유하는 것이 자연스러운 사회에서 정보에 대한 빠른 인식이 요구되는 것은 매우 당연하다. 사람들은 1차원 바코드를 이용하여 이를 해결하려 하였으나 빠른 정보사회의 흐름을 따라가기엔 1차원 바코드는 그 용량과 다양한 수용에 있어 많이 부족한 모습을 보였다. 1994년 9월에 덴소 웨이브 사에서 2차원 바코드인 QR Code가 개발되

* 교신저자: 이종연(jongyun@chungbuk.ac.kr)
* 본 논문은 미래창조과학부 및 정보통신산업진흥원의 산학협력 특성화 지원사업의 연구결과로 수행되었음(NIPA-2013010918).
접수일 2013년 11월 5일 수정일 2013년 11월 29일 게재확정일 2013년 12월 2일

었다. 기존의 1차원 바코드와는 달리 2차원 바코드는 그 용량과 정보 수용성이 매우 뛰어나 여러 분야에서 받아들여 지고 이용 되게 되었다. 2000년도에 ISO/IEC의

표준으로 채택되었고, 덴소 웨이브 사에서는 특허권을 행사하지 않아[1], google(c)의 ZXing 등 오픈 소스나 부분 자료들이 생겨나고 많은 이들이 이용하게 되었다.

하지만 기존의 연구들의 문제점을 요약하면 다음과 같다. 첫째, 기존연구들은 주로 QR Code의 영상인식과 인코딩 부분에 초점이 맞추어져 있고 처리속도 개선에 중점을 두고 있다. 둘째, QR Code의 디코딩 과정에 대한 구체적인 문서화가 존재하지 않는다. 셋째, QR Code의 디코딩 과정은 불필요한 부분까지 해석하는 문제점이 존재한다.

1.2 연구 목표 및 내용

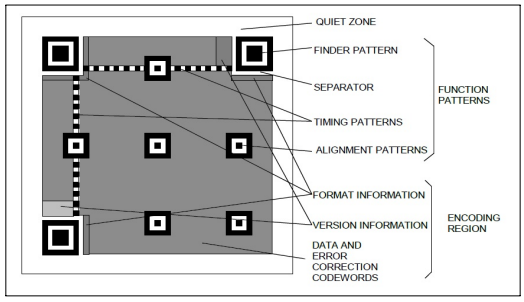
따라서 본 논문에서는 효율적인 QR Code의 디코딩 방법을 제안하고 이의 처리 알고리즘을 구체적으로 문서화하여 제안하고자 한다. 그 세부적인 연구내용은 다음과 같다. 첫째, 먼저 QR Code의 구조를 검토한다. 둘째, QR Code의 인코딩 과정에 따른 디코딩 과정을 예를 들어 설명하고 이에 따른 효율적인 디코딩 알고리즘을 제안한다. 여기서 인코딩 모드에는 숫자, 영숫자, 바이트 유형이 존재한다.

본 논문의 구성은 다음과 같다. 먼저 2장에서는 QR Code의 전체적인 심볼 구조를 살펴본다. 3장에서는 QR Code의 디코딩 과정을 검토하고 단계별로 효율적인 디코딩 알고리즘을 설계한다. 마지막으로 4장은 결론을 요약한다.

2. QR Code의 구조

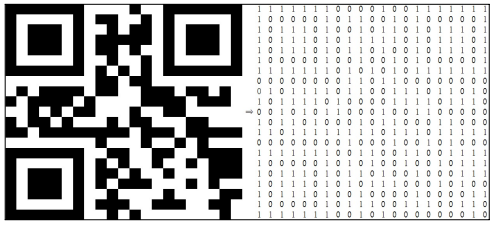
2.1 QR Code의 심볼 구조

[그림 1]은 QR Code에서 각 심볼의 구조이다. 하나의 QR Code는 기능 패턴(function pattern)과 인코딩 영역(encoding region)으로 구분된다. 그리고 세부적인 기능 패턴에는 finder pattern, alignment pattern, timing pattern 등으로 구성되며 인코딩 영역에는 format information, version information, data and error correction codewords 등의 요소들로 구성된다. 여기서 기능 패턴(function pattern)들은 항상 QR Code의 같은 자리와 공간을 차지한다. 기능 패턴들로 하여금 고속 인식과 인식률 향상에 도움이 된다. 그리고 각 구성요소들의 상세 정보는 다음과 같다. 첫째, finder pattern은 고정적으로 세 개의 방향에 사각형이 위치한다. 둘째, 세 개의 finder pattern을 이어주는 2개 점선의 timing pattern이 존재한다. 셋째, format information은 finder pattern 주변을 둘러싸고 있는 사각형 박스이다. 넷째, 버전 정보는 버전 7 이상부터 방향 패턴 좌측과 상측에 version information이 자리하고 버전 2 이상부터는 alignment pattern이 고정된 위치에 자리 잡는다. 하지만 QR Code의 구성요소는 버전에 따라 다소 상이하게 구성될 수 있음을 유의하기 바란다.



[Fig. 1] QR Code Symbol structure [1]

2.2 2차원 QR Code의 데이터



(a) (b)

[Fig. 2] Encoding of QR Code: (a)QR Code Encoded; (b)QR Code represented by binary code.

QR Code는 [그림 1]과 같이 버전에 따라 기능 패턴들을 항상 같은 자리에 위치시킨다. 기능 패턴들을 제외한 공간에 데이터가 0과 1의 이진수로 표현 된다. [그림2(b)]는 [그림2(a)]의 QR Code를 이진수로 표현한 2차원 배열 형태이다.

* ISO/IEC(International Organization for Standardization/ International Electrotechnical Commission Joint Technical Committee 1) : ISO와 IEC의 정보 기술 표준안의 충돌을 막은 효율적인 표준.

2.3 버전에 따른 데이터 수용량

QR Code는 버전에 따라 데이터를 수용할 수 있는 크기가 다르다. [표1]은 버전 1부터 40 중에 25까지 보여주는 버전에 따른 데이터 수용 가능한 양을 보여준다. 실질적으로 QR Code의 데이터가 저장되는 데이터 모듈의 저장 공간은 QR Code의 전체 크기에서 기능 패턴과 Format/Version Information를 제거한 영역이다. 이 데이터 모듈 공간에는 모듈에 데이터가 인코딩된 이진수의 수들이 수록된다.

[Table 1] Codeword capacity according to versions [1]

version	size (A)	Function pattern (B)	Format / version modules (C)	Data modules ($D = A^2-B-C$)
1	21	202	31	208
2	25	235	31	359
3	29	243	31	567
4	33	251	31	807
5	37	259	31	1079
6	41	267	31	1383
7	45	390	67	1568
8	49	398	67	1936
9	53	406	67	2336
10	57	414	67	2768
11	61	422	67	3232
12	65	430	67	3728
13	69	438	67	4256
14	73	611	67	4651
15	77	619	67	5243
16	81	627	67	5867
17	85	635	67	6523
18	89	643	67	7211
19	93	651	67	7931
20	97	659	67	8683
21	101	882	67	9252
22	105	890	67	10068
23	109	898	67	10916
24	113	906	67	11796
25	117	914	67	12708

2.4 모드의 종류

QR Code에는 숫자, 영숫자, 바이트, kanji의 네 가지 저장 모드가 있으며, 각 모드에 따라 인코딩 방법이 다르다. 한편 본 논문에서는 GS1 규정에 따라 Kanji모드는 기술하지 않으며[6], 각 모드별 세부적인 인코딩 방법은 다음과 같다.

1) 숫자 모드

숫자 0부터 9까지 표현된다. 2진수 10 비트로 3개의 데이터가 인코딩되어 입력된다.

2) 영숫자 모드

영숫자 모드는 ISO규격의 [Encoding/decoding table for Alphanumeric mode]의 표에 따른다. 결과적으로 영숫자 인코딩에는 45개의 문자집합, 10개의 숫자, 26개의 알파벳, 9개의 특수기호가 제공된다.

3) 바이트 모드

바이트 모드는 8 비트 데이터로 인코딩되며, ACSII 문자 세트를 따른다.

4) 총 데이터 비트수 산출 공식

인코딩되어야 할 문자의 자릿수로 (식1)의 산출 공식에 의해 데이터가 저장되는 총 데이터 비트수를 산출할 수 있고 실제 데이터 값이 몇 번째 모듈까지 인코딩되었는지를 알 수 있다. 모드에 따른 산출 공식이 다른 이유는 모드별 요구하는 인코딩 데이터 비트수가 다르기 때문이다.

총 비트수 = 4 +
문자수 지시자 +
모드에 따른 데이터 비트수 산출식 (식1)

간단한 예제를 (식1)에 적용하여 설명에 이해를 돕겠다.

[예제1] 'HELLO WORLD'의 비트수 연산의 예이다.
단계1: 'HELLO WORLD'는 영숫자모드이다.
단계2: 'HELLO WORLD'는 11자리수이다.
단계3: 산출 공식 적용

$4 + 9 + 11 \times ((\text{자릿수}:11)/2 = 5) + 6(\text{남음} = 1)$
$= 74$

이렇게 총 74bit의 공간에 'HELLO WORLD'가 저장된 것을 확인할 수 있다.

[Table 2] Computation of the number of data bits according to the indicator.

문자수 지시자	데이터 비트수 산출 공식		
숫자 10(bit)	숫자 10 X (자릿수 / 3)의 몫	+	남음 = 0 일 때 0
			남음 = 1 일 때 4
			남음 = 2 일 때 7
영숫자 9(bit)	영숫자 11 X (자릿수 / 2)의 몫	+	남음 = 0 일 때 0
			남음 = 1 일 때 6
바이트 8(bit)	Binary 8 X 자릿수		

2.5 데이터의 변환

앞 절에서 설명했듯이 QR Code는 모드에 따른 데이터의 변환 방식이 다르다. 요구하는 모듈의 수와 변환 공식이 다르기 때문에 그에 따른 디코딩 과정도 다르게 설정해야 한다.

1) 숫자 모드

세 개의 원소를 한 묶음으로 해서 10bit의 크기에 맞는 이진수로 표현하여 저장한다. 만약 원소의 개수가 2개 또는 1개이면 [표2]에 따른 bit수에 맞게 저장한다.

2) 영숫자 모드

두 개의 원소를 11bit의 크기에 맞는 이진수로 표현하여 저장한다. 이진수로 변환할 때 원소들은 영숫자 테이블에 맞는 색인 값으로 변환하고 첫 원소에 45를 곱한 후 두 원소에 합을 이진수로 변환하여 저장한다.

[예제2] 영숫자 HE의 인코딩 변환 과정

단계1: H→17, E→14 영숫자 테이블에 따른 변환

단계2: 17 × 45 + 14 = 779 영숫자 모드 산출공식

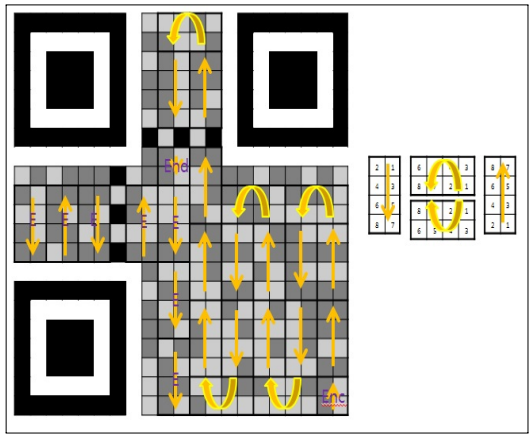
단계3: 779→01100001011 11bit크기의 이진수 변환

3) 바이트 모드

아스키 테이블을 참조하고 있으며 한 단어를 8 비트 크기에 맞게 이진수로 변환하여 저장한다.

2.6 모듈의 위치 선정

[그림1]과 같이 기능 패턴과 format/version information는 QR Code의 버전에 따라 고정적인 좌표에 위치한다. 이러한 영역들을 제외한 위치에 데이터와 에러 코드워드는 데이터 비트수에 맞게 인코딩된다. QR Code는 이 영역들에 대해 항상 일정 순서에 따라 인코딩 이 실행된다. [그림3]은 기능패턴 및 format/version information을 제외한 영역에 데이터가 인코딩되는 순서를 보여준다. 시작은 우측 아래의 끝, 즉 행렬로 표현하면 (maxsize, maxsize)부터 시작하여 항상 우측에서 좌측으로 한 칸 이동 후 위쪽으로 이동한다. 그 후 기능 패턴 및 QR Code의 한 열을 모두 읽으면 똑같은 작업을 아래쪽으로 이동한다. 이 과정은 QR Code의 데이터가 인코딩되는 순서이므로 디코딩 과정도 이 순서에 따라 데이터를 읽어야 한다.



[Fig. 3] The order of storing data in QR Code

2.7 마스킹

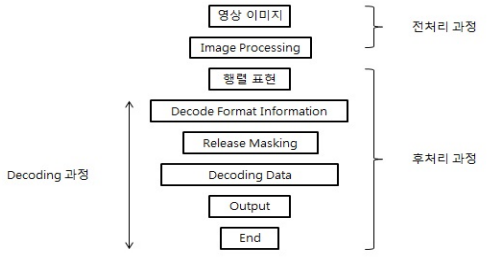
인코딩 과정이 끝난 후에 QR Code는 다른 2차원 코드와는 달리 Masking을 한다. 일정한 연산에 따라 검은색과 흰색의 모듈을 바꿔주는 작업이다. QR Code에서 제공하는 Data Masking은 8가지의 연산으로 구성되어 있고, 가장 최적화된 연산으로 하여 QR Code를 마스킹한다. [표 3]은 마스킹 연산의 8가지이고, 행과 열의 연산으로 이루어져 있다.

[Table 3] Masking patterns

Pattern	Operations
0	(row+col)%2 == 0
1	(row)%2 == 0
2	(col)%3 == 0
3	(row+col)%3 == 0
4	((row/2)+(col/3))%2==0
5	(row*col)%2+(row*col)%3==0
6	((row*col)%2+(row*col)%3)%2==0
7	((row+col)%2+(row*col)%3)%2==0

3. 2차원 QR Code의 디코딩 알고리즘

본 논문에서는 QR Code의 이미지를 행렬 데이터로 얻었다는 가정하고 있다. 얻어진 행렬 데이터를 원래의 데이터로 해석하는 알고리즘을 설명할 것이다. [그림4]는 QR Code의 전체적인 검증 과정이다.



[Fig. 4] The overall processes of verification in QR Code

본 논문에서는 QR Code의 전체적인 검증 과정 중 후처리 과정인 QR Code 디코딩 과정에 초점을 두어 설명할 것이다. 디코딩의 과정은 앞의 [그림 2]에 표시된 QR Code를 예를 들어 디코딩 과정을 소개할 것이다.

3.1 형식정보 해석

형식정보(Format Information)는 QR Code 이미지에서 세 개의 방향패턴(Finder Pattern) 주변에 위치하며, ECC (Error Correction Codeword) 레벨과 Mask Pattern의 정보를 포함하는 15자리의 2진법 수로 저장되어 있다. 읽어온 정보로 ECC 레벨과 Mask Pattern을 확인하는 법은 [그림 5]와 같다.

구분	결과
단계1: Format Information	100000011001110
단계2: XORmask(0x5412)	101010000010010
단계3: Restore	001010011011100
단계4: 상위 5Bit 추출	00101
단계5: 상위 2Bit(ECC Level)	00 **(M)**
단계6: 하위 3Bit(Mask)	101 **(5)**

[Fig. 5] The steps of decoding the format information

단계별 설명은 다음과 같다.

단계1 : 형식 정보의 좌표에서 정보를 얻는다.

단계2 : QR Code의 XORMask = 0x5412

단계3 : 얻은 형식정보에 마스크 역적용 하여 마스크를 해제한다.

단계4 : 형식정보 15비트 중 상위 5비트에 ECC Level과 Mask Pattern이 있다. shift 연산을 이용하여 상위 5비트를 추출한다.

단계5 : 5비트 중 상위 2비트가 ECC Level이다.

단계6 : 5비트 중 하위 3비트인 Mask Pattern이다.

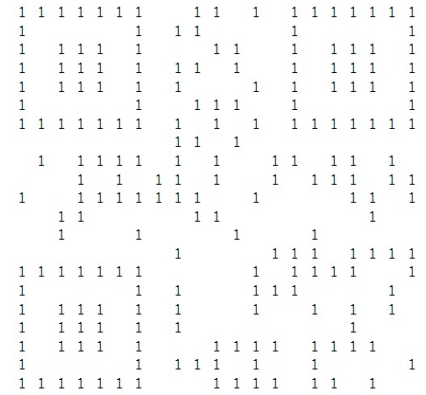
이 과정을 알고리즘으로 표현하면 [그림 6]과 같다.

```
#define XORMask 0x5412
int FormatInfo[15];
void decodeFormatInfo()
Begin
1:    // restore format information.
2:    FormatInfo := FormatInfo XOR XORMask;
3:    // 상위 5bit 추출
4:    FormatInfo := FormatInfo >> 10;
5:    // 상위 2bit 추출
6:    ErrorCorrectionLevel = FormatInfo >>  3;
7:    // 하위 3bit 추출
8:    DataMask := (byte) (FormatInfo & 0x07);
End
```

[Fig. 6] Algorithm of decoding the format information

3.2 마스킹 해제

앞 절에서 얻어진 Mask Pattern은 규격화된 7가지 연산 중 해당 연산을 이용하여 기존의 데이터가 저장된 QR Code로 복원이 가능하다. [그림7]은 [그림2]의 QR Code에서 Mask Pattern을 역적용하여 기존의 QR Code로 복원한 것이다.



[Fig. 7] An example of releasing masks in [Fig.2]

```
    bool mask(int row, int col)
    Begin
1:      // Global variable integer maskpattern;
2:      boolean condition := false
3:      switch (maskpattern) do
4:      // mask pattern 000
5:      case 0 : if ((row + col) % 2 == 0) then
6:          condition := true; break;
7:      // mask pattern 001
8:      case 1 : if (row % 2 == 0) then
9:          condition := true; break;
10:     // mask pattern 010
11:     case 2 : if (row % 3 == 0) then
12:         condition := true; break;
13:     // mask pattern 011
14:     case 3 : if ((row + col) % 3 == 0) then
15:         condition := true; break;
16:     // mask pattern 100
17:     case 4 : if ((row * col) % 2 +
18:                 (row * col) % 3 == 0) then
19:         condition := true; break;
20:     // mask pattern 101
21:     case 5 : if (((row/2) + (col/3)) % 2 == 0) then
22:         condition := true; break;
23:     // mask pattern 110
24:     case 6 : if (((row*col)%2+(row*col)%3 )%2==0) then
25:         condition := true; break;
26:     // mask pattern 111
27:     case 7 : if(((row+col)%2+(row*col)%3)%2==0) then
28:         condition := true; break;
29:     endswitch
30:     return condition;
    End
```

[Fig. 8] Algorithm of masking patterns in QR Code

예로, [그림 7]은 [표 3]의 8가지 연산 중 하나의 연산을 이용하여 마스크가 해제된 상태이다. 여기서 마스크 패턴은 각 마스크 패턴과 일치하고 행과 열에 대응되는 연산이 일치하는 모듈만을 해제된다. [그림8]은 QR Code에서 각 모듈을 마스크 하는 알고리즘이다. 결과적으로 [그림 8]은 [표 3]의 8가지 마스크 패턴 연산을 알고리즘으로 표현한 것이다.

[그림 9]는 QR Code의 열과 행의 길이를 매개변수로 받고 마스크 패턴과 행과 열의 연산으로 인해 블리언 값이 일치하면 그 행과 열의 QR Code 모듈 값을 반대로 해제하는 알고리즘이다.

```
    void releaseMask(int row, int col)
    Begin
1:      for i ∈ (1,2,...,row) do
2:        for j ∈ (1, 2,..., col) do
3:          if (mask(row, col) == true) then // call by mask
4:              QRCode[row][col] := QRCode[row][col] XOR 1;
5:        endfor
6:      endfor
    End
```

[Fig. 9] Algorithm of releasing mask

3.3 데이터 디코딩 과정

[그림 3]에서 같이 QR Code는 좌우 2칸씩 읽은 후 위쪽 또는 아래쪽으로 이동하면서 인코딩을 시작한다. 위쪽 또는 아래쪽의 방향은 기능 패턴 또는 QR Code의 끝 영역에 도달하면 방향을 바꾸어 진행한다.

본 논문에서는 QR Code의 디코딩 과정은 다음과 같이 이뤄진다. 첫째, 디코딩 연산의 효율성을 위해 기능 패턴들의 좌표의 모듈들을 다른 값(-1)으로 세트한다. 둘째, 2차원 배열에 저장된 QR Code의 모듈을 1차원 배열에 병합하여 저장한다. 셋째, 1차원 배열을 읽으면서 디코딩한다. [그림 10]은 이러한 QR Code의 기능패턴이 다른 값으로 세트된 상태에서 1차원 배열에 저장하는 알고리즘이다.

```
    void copyBit(int row,int col)
    Begin
1:      QRDecoder := (int *)malloc(DataBits * sizeof(int));
2:      int index := 0;
3:      // function pattern = -1
4:      while (DataBits != index) {
5:        int count := 0;
6:        for i∈(col,col−2,col−4,...,0) do
7:          if (count++ % 2 ==0) do //on the top of code
8:            for j ∈ (row,row−1,...,0) do
9:              if (QRcode[row][col] != -1) then
10:               QRDecoder[index++] :=
                                QRcode[row][col];
11:             endif
12:             if (QRcode[row][col - 1] != -1) then
```

```
13:                     QRDecoder[index++] :=
                                    QRcode[row][col - 1];
14:                 endif
15:               endfor
16:             else { // on the bottom of code
17:               for j∈(0,1,,...,row) do
18:                 if (QRcode[row][col] != -1) then
19:                   QRDecoder[index++] :=
                                    QRcode[row][col];
20:                 endif
21:                 if (QRcode[row][col - 1] != -1) then
22:                   QRDecoder[index++] :=
                                    QRcode[row][col - 1];
23:                 endif
24:               endfor
25:             endif
26:           endfor
27:         endwhile
    END
```

[Fig. 10] Algorithms of transforming a two-dimensional array in QR Code into one-dimensional array

QR Code의 해독은 처음에 우측 하단에서 시작하여 항상 위쪽 방향으로 진행된다. 그리고 두 열을 다 읽은 후, 열을 2칸 이동시키고 반대 방향으로 진행된다. 이러한 방법을 [그림 10]의 알고리즘으로 표현했다. [그림 10]의 설명은 다음과 같다.

단계 4부터 27까지의 while문은 1차원 배열의 색인이 데이터 비트수만큼 저장될 때까지를 나타낸다. 단계 6부터 26까지 for문은 2차원 배열의 열을 변화한다. 2개의 열씩 복사하여 열 변화의 차이는 2씩 감소한다. 단계 8부터 15까지의 for문은 위쪽 방향으로 진행하여 복사하는 알고리즘이다. 행의 변화를 이용하여 기능 패턴(function pattern)을 제외한 영역만을 1차원 배열에 저장한다. 단계 17부터 24까지 for문은 아래 방향으로 진행하여 복사하는 알고리즘이다.

그리고 QR Code의 디코딩 과정은 다음의 단계를 거쳐 순차적으로 진행된다.

1) 모드 해석

QR Code에는 [표 4]와 같이 4가지 모드가 있다.[1] 모드에 따라 자릿수 지시자와 디코딩 방법이 다르다. 그래서 디코딩 과정에서 처음 4bit에 저장된 모드를 우선적으로 읽어야 한다.

[Table 4] The modes of QR Code [1]

Mode	QR Code Symbols
Numeric	0001
Alphanumeric	0010
Byte	0100
Kanji	1000

여기서 Kanji 모드는 GS1 규격에서도 지원하지 않는다[6]. 따라서 본 논문에서도 제공하지 않는 것으로 가정한다.

예로, 1차원 배열에 저장된 QR Decoder에서 0부터 3까지 4bit의 수를 읽으면 [그림 7]의 QR Code는 '0010'이며, 영숫자(Alphanumeric) 모드임을 알 수 있다.

2) 지시자 확인

지시자는 자릿수를 저장한 영역이고, 모드에 따라 지시자가 저장된 bit수가 다르다. 지시자를 해독함으로써 데이터 영역에 몇 비트까지 데이터가 저장됨을 알 수 있다. 지시자의 수는 버전과 모드에 따라 다르지만, [표5]의 규칙에 따른다.

[Table 5] The number of bits in indicator

Version	Numeric	Alpha	Byte
1 to 9	10	9	8
10 to 26	12	11	16
27 to 40	14	13	16

[그림 7]의 지시자는 영숫자이므로 [표 5]에서 보듯이 9비트를 읽어야 하고 '000001011'이란 것을 알 수 있다. 이를 십진법으로 변환하면 '11'이라는 수를 얻을 수 있다. 정리하면 [그림 7]의 QR Code는 영숫자 11자리의 정보가 인코딩된 것임을 알 수 있다.

3) 데이터 산출 공식

자릿수를 확인했다는 것은 모드에 따라 QR Code의 몇 비트까지 데이터가 저장되어 있다는 것을 알 수 있다. 따라서 QR Code는 정해진 산출 공식에 의하여 인코딩된 값의 저장 영역을 확인할 수 있다. [표 2]의 공식에 따라 [그림 7]의 QR Code를 적용하면 다음과 같다.

$$74 = 4 + 9 + 11 \times (11/2) + 6$$

계산에 의해 총 74비트까지 데이터가 인코딩되어 있다는 것을 알 수 있고, 저장된 1차원 배열에 0부터 73까

지 데이터가 저장된 것을 확인할 수 있다.

4) 데이터 해석

본 논문에서 그 동안 모드를 위한 4비트, 지시자를 위한 9비트, 총 13비트를 읽었다. 데이터를 해석하기 위해서 13bit부터 74bit까지 읽어야한다.

```
    void AlphaMode(int indicator, int chipher, int Bits)
    Beign
1:      int start := 4 + indicator;
2:      int *temp, *temp2;
3:      temp := (int *)malloc(11 * sizeof(int));
4:      temp := (int *)malloc(6 * sizeof(int));
5:      while (start != Bits) {
6:        for i∈(1, 2, ..., 11) do
7:          if (start != Bits) then
8:            temp[i] := QRDecoding[start++];
9:          else
10:           break;
11:         endif
12:       endfor
13:       if (chipher => 2) then
14:         Save the value of Alphanumeric[temp / 45] into queue;
15:         Save the value of Alphanumeric[temp %45] into queue;
16:         chipher = chipher - 2;
17:       else if (chipher == 1) then
18:       for i∈(0,1,2,...,5) do
19:         temp2[i] := temp[i];
20:       endfor
21:         Save the value of Alphanumeric[temp2 / 45] into queue;
22:         chipher = chipher - 1;
23:       endif
24:       break;
25:     endwhile
    End
```

[Fig. 11] Algorithm of decoding the QR Code in Alphanumeric mode

[그림 11]은 읽을 데이터에 대해 영숫자 모드에 따른 11비트의 저장 공간을 할당하고, 모드와 지시자의 뒷부분인 13번째부터 74까지 읽어오는 방법이다. 알고리즘의 설명은 다음과 같다.

모드에 해당한 자릿수인 indicator와 QR Code에 인코딩된 문자열의 자릿수인 chipher, 데이터가 저장된 비트수인 Bits의 매개변수를 사용한다. 단계 1은 모드와 지시자의 덧셈의 수로 시작점을 설정한다. 단계 3과 4에서는 영숫자 모드에 따른 데이터 공간인 11비트와 6비트 저장 공간을 임시로 만든다. 단계 5부터 단계 25까지의 while 문은 시작점이 데이터 끝과 만날 때까지 수행하는 문장이다. 단계 6부터 12의 for문은 항상 데이터를 11비트 씩 저장하고 시작점이 끝점과 일치한다면 저장을 중지하는 역할을 한다. 단계 13은 자릿수가 2보다 큰 경우에는 임시배열에 두 문자가 저장되었다. 두 가지의 정보는 몫과 나머지로 구성되었다. [그림 12]의 배열을 이용하여 $'Alphanumeric[temp/45]'$, $'Alphanumeric[temp\%45]'$의 값을 각각 큐에 저장한다. 그 후 다음 단계를 위하여 자릿수를 '2'만큼 감소한다. 단계 17은 자릿수가 1일 경우이다. 임시 배열에 정보가 한 개의 문자가 들어있다. 그렇다면 나머지가 없는 몫만으로 구성되어 있고, 단계 13과 같이 [그림12]의 배열을 이용하여 몫으로 값을 구하고 큐에 저장한다.

```
    Alphanumeric table
1:      char Alphanumeric[45] =
2:              {'0','1','2','3','4','5','6','7','8','9',
3:               'A','B','C','D','E','F','G','H','I','J',
4:               'K','L','M','N','O','P','Q','R','S','T',
5:               'U','V','W','X','Y','Z',' ','$','%','*',
6:               '+','-','.','/',':'};
```

[Fig. 12] Alphanumeric table[1]

[풀이] [그림 7]의 QR Code에서 13번째부터 74번째까지 이진수를 나열하면 다음과 같다.

value	01100001011	01111000110	10001011100
index	13,...,23	24,...,34	35,...,45
value	10110111000	10011010100	001101
index	46,...,56	57,...,67	68,...,73

11자리씩 분할하여 해석하면 다음과 같다.

$$01100001011 \rightarrow 779 = 17 \times 45 + 14 \rightarrow [HE]$$
$$01111000110 \rightarrow 966 = 21 \times 45 + 21 \rightarrow [LL]$$
$$10001011100 \rightarrow 1{,}116 = 24*45 + 36 \rightarrow [O(SP)]$$
$$10110111000 \rightarrow 1{,}464 = 32 \times 45 + 24 \rightarrow [WO]$$
$$10011010100 \rightarrow 1{,}236 = 27 \times 45 + 21 \rightarrow [RL]$$
$$001101 \rightarrow 585 = 13 \times 45 \rightarrow [D]$$

결과적으로, 지금까지의 디코딩 과정을 요약하면, [그림2]의 QR Code는 'HELLO WORLD'의 데이터 내용임을 알 수 있다.

4. 결 론

본 논문에서는 QR Code 인코딩 방법에 따른 디코딩 과정의 설계와 알고리즘을 제안하였다. 이 과정에서 모드에 따라 다른 인코딩 방법과 다양한 기능 패턴들로 인해 QR Code의 디코딩 과정을 기술하는데 어려움이 있었다. 결과적으로 이러한 어려움을 해결하고 사용자의 이해를 돕기 위해 본 논문의 기여도는 QR Code 디코딩 관련하여 다음과 같은 몇 가지 장점을 제공할 것이다. 첫째, 2차원 QR Code의 디코딩 과정에 대해 상세한 처리절차와 알고리즘을 명확히 제시하였다. 이로 인해 2차원 QR Code의 디코딩에 대한 문제점 해결 방법에 기여할 것이다. 둘째, 디코딩 과정에서 불필요한 기능패턴 영역을 제외하고 저장함으로서 데이터가 저장된 부분만을 디코딩할 수 있게 하였다. 이것은 결과적으로 인라인 QR Code의 검증 과정에서 디코딩 시간을 단축하는 데 기여할 것으로 예상된다.

끝으로 앞으로의 연구과제는 QR Code의 인라인 검증 과정에서 이미지를 입력받아 행렬로 변환하는 전처리 과정이 개발되어야 할 것이며, 또한 다른 바코드 검증기와의 비교 실험을 통한 성능 우수성의 비교 검토가 필요하리라 생각된다.

REFERENCES

[1] ISO/IEC 18804:2000. Information technology - Automatic identification and data capture techniques - Bar code symbology - QR Code,2000

[2] http://en.wikipedia.org/wiki/QR_code(2013)

[3] http://code.google.com/p/zxing(2013)

[4] http://www.thonky.com/qr-code-tutorial/(2013)

[5] http://www.qrcode.com/en/(2013)

[6] GS1, GS1 General Specifications Version 13, January 2013.

[7] Yeon-Kyung Lee and Hoon Yoo, “QR-code finder recognition using four directional scanning method”, Korea Informations and Communications Conference 2011, Vol. 16, No. 6, 2012.

저자소개

박 광 욱(Park, Kwang Wook)

・2009년 3월 ~ 현재 : 충북대학교 컴퓨터교육과 학사과정

<관심분야> : 데이터베이스시스템, 색인구조, 바코드 검증기

・E-Mail : priest68@naver.com

이 종 연(Lee, Jong-Yun) **[종신회원]**

・1985년 2월 : 충북대학교 전자계산공학과(공학사)

・1987년 2월 : 충북대학교 대학원 전자계산기공학과(공학석사)

・1999년 2월 : 충북대학교 전자계산학과(이학박사)

・1990년 2월 ~ 1996년 5월 : 현대전자산업(주) SW연구소 및 현대정보 기술(주) CIM사업부 근무(책임연구원)

・1999년 3월 ~ 2003년 3월 : 강원대학교 삼척캠퍼스 정보통신공학과 조교수

・2003년 3월 ~ 현재 : 충북대학교 컴퓨터교육과 및 디지털정보융합학과 교수로 재직 중

<관심분야> : 질의처리 및 최적화, 시공간 데이터베이스, 과학 데이터베이스, u-learnimg 및 평가모델, 유통물류, GIS

・E-Mail : jongyun@chungbuk.ac.kr