논문 2009-46TC-10-3

Radix 4 Polar code의 부호 및 복호

(Encoding & Decoding of Radix 4 Polar Code)

이 문 호*, 최 은 지**, 양 재 승***, 박 주 용****

(Moon Ho Lee, Eun Ji Choi, Jae Seung Yang, and Ju Yong Park)

요 약

Polar code는 터키 Erdal Arikan교수가 2006년 입력된 채널을 나누면 Cutoff Rate이 향상되는데 착안하여 Polar code를 제안했다. 채널분극은 주어진 B-DMC(Binary-input Discrete Memoryless Channel) W에서 대칭 용량의 높은 비율을 가진 연속적인 code로 이루어져 있다. 대칭 용량은 동등한 확률을 가진 채널의 입력을 이용하여 높은 비율을 얻는데 채널분극은 주어진 B-DMC W의 N개의 독립적인 출력을 모은 것이다. 즉, N은 Binary입력 채널$\{W_N^{(i)} : 1 \le i \le N\}$ 일 때, N이 커지게 되고, I{WN(i)}에서 값이 1에 가까워지면 그 값은 I(W)로 접근되고, I{WN(i)} 값이 0에 가까워지면 1-I(W)에 접근된다. 여기에서 I(W)는 신뢰성 있는 통신상에서의 동등한 주파수를 가진 W의 입력으로 높은 비율을 나타낸다. 이로써 {WN(i)}는 결국 채널코딩을 위한 적합한 상태라고 볼 수 있다. Polar code를 바탕으로, 본 논문은 Arikan의 Polar code의 부호화와 복호화를 분석하고 새롭게 Radix4의 Polar code 부호화를 제안했다.

Abstract

Polar Code was proposed by Turkish professor Erdal Arikan in 2006 as an idea that splitted input channel is increasing the cutoff rate. The channel polarization consisted of code sequences with symmetric high rate capacity in a given B-DMC(Binary-input Discrete Memoryless Channel) W. The symmetric capacity is the highest rate achievable subject to using the input letters of the channel with equal probability. The channel polarization is said to a set of given N independent outputs of B-DMC W. In other word, N increases when N is a set of binary-input channels $\{W_N^{(i)} : 1 \le i \le N\}$, in I{WN(i)} as the fraction of indices is near to 1, which is approaching to I(W), and it is near to 0, then to 1-I(W), where I(W) presents high rates in reliable wireless communication channel as inputs of W with equal frequences. After all, {WN(i)} is shown to be a state of channel coding. On the based on this Polar codes, this paper analyzes Polar coding and decoding of Arikan and propose Radix4 Polar coding newly.

Keywords : Polar Code, cutoff rate, channel polarization, B-DMC, Radix4 Polar coding.

Ⅰ. 서 론

Polar code는 터키의 Arikan 교수가, 입력된 채널을 나누면 Cutoff Rate이 향상되는데 착안하여 이를 제안했다. Polar code에서 나누어진 채널 W는 대칭 용량의 높은 비율을 가진 연속적인 code로 이루어져 있는데, 동등한 확률을 가진 채널의 입력을 이용하여 높은 비율을 얻는다. 이를 기반으로 Polar code를 채널코딩에 적용시켜 W의 입력으로 신뢰성 있는 통신상에서의 높은 비율을 나타낸다.

본 논문은 다음과 같이 구성되어 있다. Ⅱ장에서는 Polar code의 기본개념을 서술한다. Ⅲ장에서는 Arikan 교수가 제안한 Polar code의 대수적으로 분석된 부호화를 행렬로 구현함으로써 구조와 체계를 익힌다. Ⅳ장에

* 정회원, ** 학생회원, 전북대학교 전기전자컴퓨터공학부
(Chonbuk National University)
*** 학생회원, 전북대학교 정보보호공학과
(Chonbuk National University)
**** 정회원, 신경대학교 인터넷정보통신학과
(Shingyeong University)
※ 본 연구는 WCU R 32-2008-000-200 14-0 NRF지원으로 이뤄졌음
접수일자:2009년8월18일, 수정완료일자:2009년10월14일

서는 복호의 개념과 방법을 익히고 확률로써 복호화 과정을 풀어 나간다. V장에서는 Radix 4를 제안함으로써 Arikan교수의 Radix 2와의 연산 속도비를 비교한다. VI장에서 결론을 맺는다.

Ⅱ. Polar code의 기본 개념

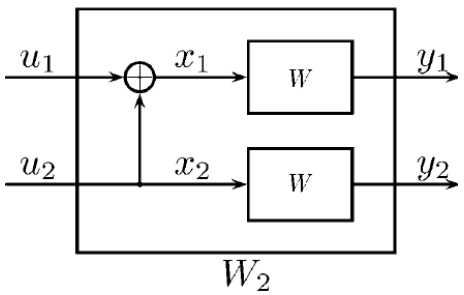
Polar code는 Arikan 교수가 부호화와 복호화의 복잡성을 가진 임의적인 대칭B-DMC를 위해 고안해낸 code이며, Polar code의 기본적인 형태는 다음과 같은 행렬로 나타낼 수 있다[1~8].

$$[G]_2 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \tag{1}$$

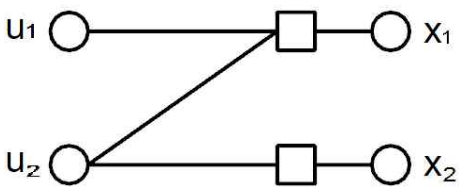
이와 같은 행렬을 바탕으로 G2⊗n은 N = 2n bit의 블록에서 B-DMC W 의 각 출력 값이 보내어진 값을 말하고, n의 값을 바꾸어 적용 시켜 나간다. 여기서 n의 값이 커질수록 채널은 n에 비례하여 분극화된다.

Polar code에서 X는 입력, Y는 출력, 변환확률(transition probability) $W(y|x), x \in X, y \in Y$과 함께 일반적인 B-DMC에서 $W: X \rightarrow Y$라고 표시한다. 여기에서 입력 X는 항상 {0,1}을 갖고, Y와 $W(y|x)$는 임의의 값을 갖는다.

N개의 bit를 가진 W에 상응하는 채널을 다음 (2)와



(a) W2 채널 블록도
(a) The Channel W2's block diagram



(b) 2×2 Polar code 등가회로
(b) 2×2 Polar code equivalent circuit

그림 1. 2×2 Polar code
Fig. 1. 2×2 Polar code.

같이 나타낸다.

$$W^N(y_1^N|x_1^N) = \Pi_{i=1}^{N} W(y_i|x_i)$$
$$W^N: X^N \rightarrow Y^N \tag{2}$$

그림 1은 Polar code의 기본구조인 2×2 W2의 채널 블록도와 2×2 등가회로이다.

그림 1에서 보는 바와 같이 채널 W2는 $W_2: X^2 \rightarrow Y^2$ 와 같이 나타낼 수 있고, 이는 두 개의 채널 W로 구성 되어져 있다. 그리고 변환확률을 구하면 다음과 같다.

$$W_2(y_1, y_2|u_1, u_2) = W(y_1|u_1 \oplus u_2)\, W(y_2|u_2) \tag{3}$$

Polar code의 기본적인 code 행렬은 아래와 같은 재귀적인 행렬로 주어진다. 여기에서 [G]N은 N의 크기를 가진 Generator 행렬이고, F⊗n은 [G]N과 크기는 같지만, Bit-reverse 되어진 값에 곱해짐으로써 Generator 행렬을 얻게 된다.

$$[G]_2^n = [R]_2^n([F]_2 \otimes [G]_2^{n-1}),$$
$$[G]_2 = [F]_2 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}. \tag{4}$$

Polar code는 이러한 기본 식을 가지고 시작하며, 이 식의 확장을 통해 N×N의 고차 행렬에서도 적용이 가능하다. 위 (4)에서 ⊗는 Kronecker product 이고, m×n 행렬 A=[A]ij와 r×s B=[B]ij를 가정하면 다음과 같이 표현 할 수 있다.

$$[A]_{ij} \otimes [B]_{ij} = \begin{bmatrix} A_{11}B & \cdots & A_{1n}B \\ \vdots & \ddots & \vdots \\ A_{m1}B & \cdots & A_{mn}B \end{bmatrix} \tag{5}$$

(5)에 나타낸 행렬은 Polar code의 기본적인 행렬연산이며 이를 바탕으로 한 Polar code의 확장은 다음 (6)처럼 $[G]_2^2$, $[G]_2^3$, $[G]_2^4$를 전개 할 수 있다.

$$[G]_2^2 = ([I]_2 \otimes [G]_2)\ [R]_2^2\ ([I]_2 \otimes [G]_2)$$
$$[G]_2^3 = ([I]_{22} \otimes [G]_2)\ [R]_2^3\ ([I]_2 \otimes [G]_2^2)$$
$$[G]_2^4 = ([I]_{23} \otimes [G]_2)\ [R]_2^4\ ([I]_2 \otimes [G]_2^3) \tag{6}$$

(6)을 일반화 하면 다음과 같다.

$$[G]_N = ([I]_{N/2} \otimes [F]_2)[R]_N([I]_2 \otimes [G]_{N/2}) \tag{7}$$

이때, $[I]_{N/2}$은 Identity 행렬이고, $[R]_N$은 Permutation 행렬이다.

Ⅲ. Polar code의 부호화

Polar code의 부호화에 대해 증명하고 분석하기 위해 Polar code의 부호화를 정의하면 다음의 식과 같이 나타낼 수 있다.

$$[y]_1^N = [u]_1^N[G]_N \quad (8)$$

(8)은 Polar code 부호화를 나타내는데, 이 식을 이해하기 위해서 (7)에 정의해 놓은 $[G]_N$을 대수적으로 표시하고, $N=2^n$, $n\geq 1$이라 정하면, $[G]_1=[I]_1$이기 때문에 $N\geq 2$이다. $([I]_{N/2}\otimes[F]_2)[R]_N$은 $[R]_N$ $([F]_2\otimes[I]_{N/2})$과 같고, 따라서 (9)로 정리할 수 있다.

$$[G]_N=[R]_N([F]_2\otimes[I]_{N/2})([I]_2\otimes[G]_{N/2}) \quad (9)$$

(9)는 $([A]\otimes[B])([C]\otimes[D])=([A][C])\otimes([B][D])$의 공식에 의해 (10)과 같다.

$$[G]_N=[R]_N([F]_2\otimes[G]_{N/2}) \quad (10)$$

$[G]_{N/2}$는 $[G]_{N/2}=[R]_{N/2}([F]_2\otimes[G]_{N/4})$과 같이 표현 할 수 있다. 이 값을 (10)에 다시 대입하면 (11)과 같이 $[G]_N$의 식을 얻는다.

$$\begin{aligned}[G]_N&=[R]_N([F]_2\otimes([R]_{N/2}([F]_2\otimes[G]_{N/4})))\\&=[R]_N([I]_2\otimes[R]_{N/2})([F]_2^{\otimes 2}\otimes[G]_{N/4})\end{aligned} \quad (11)$$

(11)을 $[B]_N$으로 대체하여 다음과 같은 최종적 식을 갖게 되는데, 여기서 $[B]_N$은 Bit-reverse 행렬로 후에 복호화 할 때, 이 reverse된 순서를 사용하게 되며, $[G]_N$은 다음과 같다.

$$\begin{aligned}&[B]_N\triangleq[R]_N([I]_2\otimes[R]_{N/2})([I]_4\otimes[R]_{N/4})\cdots\\&([I]_{N/2}\otimes[R]_2)=[R]_N([I]_2\otimes[B]_{N/2})\\&[G]_N=[B]_N[F]_2^{\otimes n}.\end{aligned} \quad (12)$$

앞으로, (7), (12)를 바탕으로 N=4일 때 Polar code의 식과 부호화 행렬을 보이고 그 다음엔 N=8일 경우와 N=16일 경우를 전개한다.

3.1. 4×4 Polar code의 부호화

N=4일 때 Polar code의 식은 항등행렬 $[I]_2$와 Polar code의 기본적인 행렬 $[G]_2$와 입력비트를 홀수부분과 짝수부분으로 나누어주는 $[R]_2^2$의 연산을 통해 (13)과 같이 나타낼 수 있다.

$$[G]_2^2 = ([I]_2\otimes[G]_2)\,[R]_2^2\,([I]_2\otimes[G]_2) \quad (13)$$

그림 2는 (13)을 등가회로로 표현한 것이다.

다음 4×4 Polar code에서 $[G]_4$를 구하기 위한 기본 (7)과 Bit-reverse 행렬이 사용된 (12)의 두 가지 계산

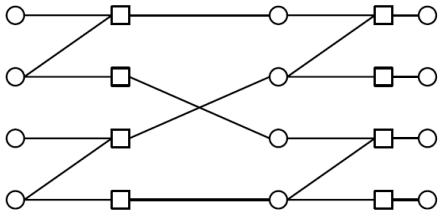


그림 2. 4×4 Polar code
Fig. 2. 4×4 Polar code.

방법에 의해 행렬을 나타내면 우선 (12)에 나타난 $[B]_N$에서 N=4일 경우, 이를 행렬로 표현하면 다음과 같다.

$$[B]_4=[R]_4([I]_2\otimes[B]_2) = \begin{bmatrix}1&0&0&0\\0&0&1&0\\0&1&0&0\\0&0&0&1\end{bmatrix},$$

여기에서 $[B]_2=[I]_2$ 이고,

$$[F]_2^{\otimes 2} = [F]_2\otimes[F]_2 = \begin{bmatrix}1&0&0&0\\1&1&0&0\\1&0&1&0\\1&1&1&1\end{bmatrix},$$

$$[G]_4 = [B]_4\times[F]_2^{\otimes 2} = \begin{bmatrix}1&0&0&0\\1&0&1&0\\1&1&0&0\\1&1&1&1\end{bmatrix}$$

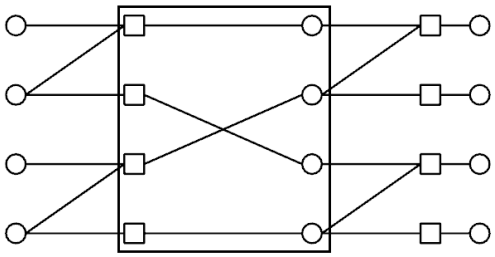
위와 같은 계산으로 $[G]_4$의 행렬을 얻을 수 있고, (7)을 기반으로 $G_2^2 = (I_2\otimes G_2)\,R_2^2\,(I_2\otimes G_2)$을 행렬로 표현하면 다음과 같다.

$$([I]_2\otimes[G]_2) = \begin{bmatrix}1&0&0&0\\1&1&0&0\\0&0&1&0\\0&0&1&1\end{bmatrix},$$

$$[R]_2^2 = \begin{bmatrix}1&0&0&0\\0&0&1&0\\0&1&0&0\\0&0&0&1\end{bmatrix},$$

$$([I]_2\otimes[G]_2) = \begin{bmatrix}1&0&0&0\\1&1&0&0\\0&0&1&0\\0&0&1&1\end{bmatrix}.$$

따라서, $[G]_4$는



$[R]_2^2$

그림 3. 4×4 Polar code permutation
Fig. 3. 4×4 Polar code's permutation.

$$[G]_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}.$$

결국 $[G]_4$의 값을 두 개의 다른 행렬 계산 과정에서 같은 값을 얻어낼 수 있다. 등가회로와 행렬을 비교해 보았을 때 다음 표시된 부분이 $[R]_2^2$ 라는 것을 알 수 있다.

3.2. 8×8 Polar code의 부호화

N=8일때 Polar code의 식은 항등행렬 $[I]_2$와 Polar code의 행렬 $[G]_2^2$와 입력비트를 홀수부분과 짝수부분으로 나누어주는 $[R]_2^3$ 의 연산을 통해 (14)와 같이 나타낼 수 있다.

$$[G]_2^3 = ([I]_2^2 \otimes [G]_2)\ [R]_2^3\ ([I]_2 \otimes [G]_2^2). \qquad (14)$$

다음 그림 4는 (14)를 등가회로로 표현한 것이다.

다음 8×8 Polar code에서 $[G]_8$을 구하기 위한 기본 (7)과 Bit-reverse 행렬이 사용된 (12)의 두 가지 계산 방법에 의해 행렬을 나타내면 먼저 (12)에 나타난 $[B]_N$에서 N=8일 경우, N=4인 경우와 같이 이를 행렬로 표현하면 다음과 같다.

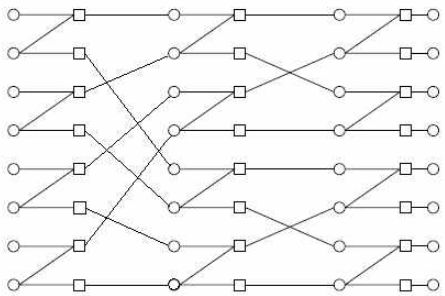


그림 4. 8×8 Polar code
Fig. 4. 8×8 Polar code.

$$[B]_8=[R]_8([I]_2\otimes[B]_4) = \begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&1&0&0&0&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&0&0&1 \end{bmatrix},$$

$$[F]_2^{\otimes 3} = \begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 1&1&0&0&0&0&0&0 \\ 1&0&1&0&0&0&0&0 \\ 1&1&1&1&0&0&0&0 \\ 1&0&0&0&1&0&0&0 \\ 1&1&0&0&1&1&0&0 \\ 1&0&1&0&1&0&1&0 \\ 1&1&1&1&1&1&1&1 \end{bmatrix},$$

$$[G]_8 = [B]_8\times[F]_2^{\otimes 3} = \begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 1&0&0&0&1&0&0&0 \\ 1&0&1&0&0&0&0&0 \\ 1&0&1&0&1&0&1&0 \\ 1&1&0&0&0&0&0&0 \\ 1&1&0&0&1&1&0&0 \\ 1&1&1&1&0&0&0&0 \\ 1&1&1&1&1&1&1&1 \end{bmatrix}.$$

위와 같은 계산으로 $[G]_8$을 얻어낼 수 있고, (7)을 기반으로 $[G]_2^3 = ([I]_2^2\otimes[G]_2)\ [R]_2^3\ ([I]_2\otimes[G]_2^2)$을 행렬로 표현하면 다음과 같다.

$$([I]_2^2\otimes[G]_2) = \begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 1&1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&1&1&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&1&1&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&1&1 \end{bmatrix},$$

$$[R]_2^3 = \begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 1&0&1&0&0&0&0&0 \\ 1&1&0&0&0&0&0&0 \\ 1&1&1&1&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&1&0&1&0 \\ 0&0&0&0&1&1&0&0 \\ 0&0&0&0&1&1&1&1 \end{bmatrix},$$

$$([I]_2\otimes[G]_2^2) = \begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 1&1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&1&1&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&1&1&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&1&1 \end{bmatrix},$$

따라서, $[G]_8$은

$$[G]_8 = \begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 1&0&0&0&1&0&0&0 \\ 1&0&1&0&0&0&0&0 \\ 1&0&1&0&1&0&1&0 \\ 1&1&0&0&0&0&0&0 \\ 1&1&0&0&1&1&0&0 \\ 1&1&1&1&0&0&0&0 \\ 1&1&1&1&1&1&1&1 \end{bmatrix}.$$

결국 $[G]_8$의 값은 두 개의 다른 행렬 계산 과정에서 서로 같음을 알 수 있다. 등가회로와 행렬을 비교해 보

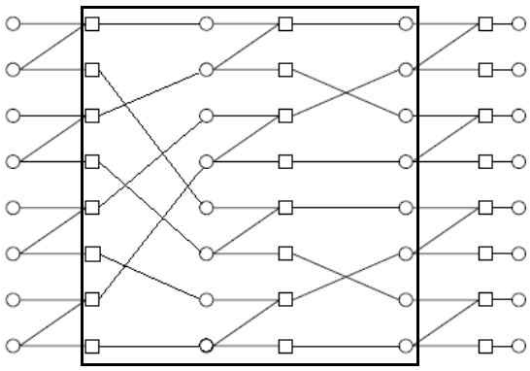


그림 5. 8×8 Polar code permutation
Fig. 5. 8×8 Polar code's permutation.

았을 때 다음 표시된 부분이 $[R]_2^3$라는 것을 알 수 있다.

3.3. 16×16 Polar Code의 부호화

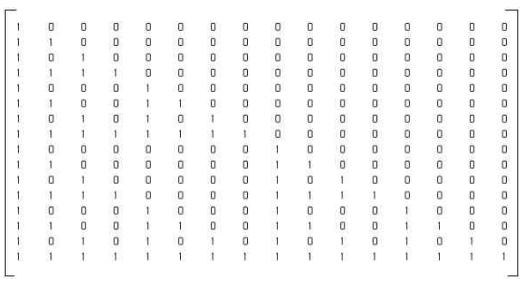
N=16일때 Polar code의 식은 항등행렬 $[I]_2$와 Polar code의 행렬 $[G]_2^3$과 입력비트를 홀수부분과 짝수부분으로 나누어주는 $[R]_2^4$의 연산을 통해 (15)와 같이 나타낼 수 있다.

$$[G]_2^4 = ([I]_2^3 \otimes [G]_2)\ [R]_2^4\ ([I]_2 \otimes [G]_2^3) \qquad (15)$$

다음 그림 6은 (15)를 등가회로로 표현한 것이다.

위에서 4×4 Polar code와 8×8 Polar code를 계산한 것과 같이 16×16을 계산 하면 다음과 같은 행렬이 된다.

$[G]_{16}=$



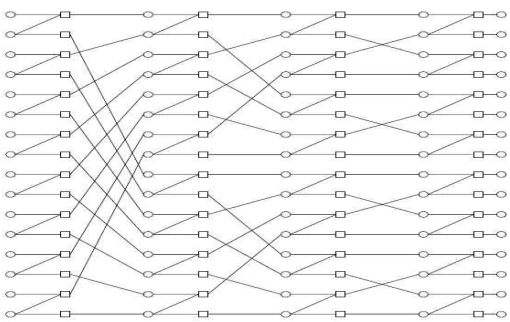
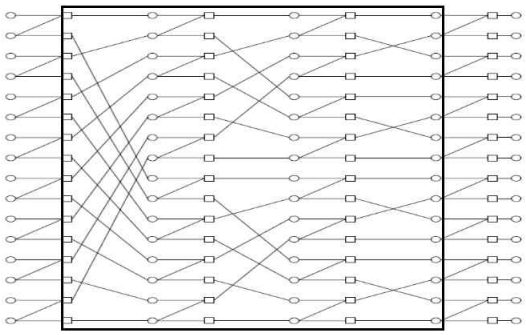


그림 6. 16×16 Polar Code
Fig. 6. 16×16 Polar Code.



$[R]_2^4$

그림 7. 16×16 Polar code permutation
Fig. 7. 16×16 Polar code's permutation.

4×4와 8×8과 마찬가지로 16×16도 등가회로와 행렬을 비교해 보았을 때 다음 표시된 부분이 $[R]_2^4$ 라는 것을 알 수 있다.

3.4. Bit-reverse 행렬로 reverse된 순서를 사용한 8×8 Polar code의 부호화

앞에서 언급한 바와 같이 $[B]_N$은 bit-reverse 행렬로 reverse된 순서를 사용하게 된다.

그림 8은 N=8일 때의 $[B]_N$을 수식의 행렬을 이용하여 그림으로 표현하였다. 앞쪽의 블록을 통해 {000,001,010…}의 Binary 값들이 {000,010,100,110…}으로 짝수, 홀수로 구분되었고, 뒤쪽의 블록에서는 그림2의 4×4 등가회로에서 $[R]_4$회로가 각각 두 개로 복사 되어진 것처럼, $[I]_2 \otimes [B]_4$ 과정으로 순서를 바꿔 줌으로써 Bit-reversed index order가 완성된다.

그림 9는 $[F]_2^{\otimes 3}$의 등가회로를 나타내었다.

여기서 앞부분이 bit-reversed된 과정 $[B]_N$이고, 뒷부분의 행렬부분이 $[F]_2^{\otimes 3}$ 과정이다.

이런 과정을 통해 구해진 부호화 코드의 결과 값은

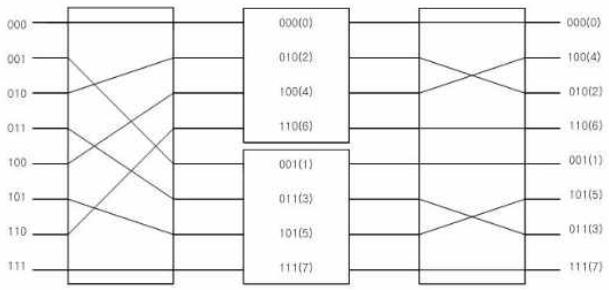


그림 8. Bit reversed index order 를 나타내기 위한 B8의 블록도
Fig. 8. B8's block diagram show bit reversed index order.

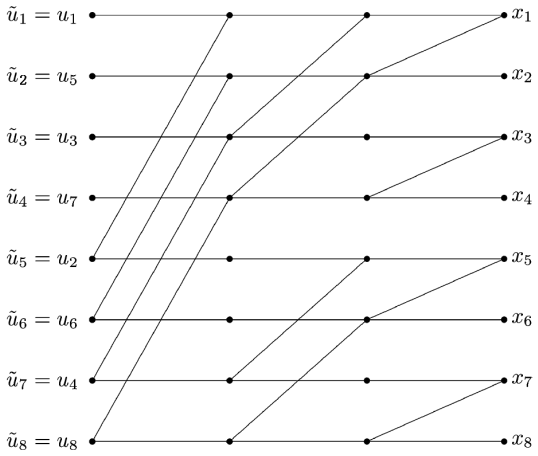


그림 9. $[F]_2^{\otimes 3}$의 등가회로
Fig. 9. $[F]_2^{\otimes 3}$'s equivalent circuit.

표 1. Arikan의 8×8 Polar code
Table 1. Arikan's 8×8 Polar code.

입력	출력
$u_1,u_2,u_3,u_4,u_5,u_6,u_7,u_8$	x_1
u_5,u_6,u_7,u_8	x_2
u_3,u_4,u_7,u_8	x_3
u_7,u_8	x_4
u_2,u_4,u_6,u_8	x_5
u_6,u_8	x_6
u_4,u_8	x_7
u_8	x_8

표 1과 같이 나타낼 수 있다.

표 1에서 보는 것과 같이 N=8일 때의 8×8 Polar code에서 출력 x_1은 입력신호 u_1,u_2, u_3,u_4,u_5,u_6,u_7,u_8을 받고, 부호화 시 이 값들은 각 노드와 연결되어 출력 값을 찾아내게 된다.

x_2역시 입력신호 u_5,u_6,u_7,u_8를 받고 부호화 시 출력과 연결되어 출력 x_2의 값을 찾을 수 있다. 표1은 그림4의 등가회로 결과 값과 같다. 여기서 결과 값을 x로 표기한 것은 등가회로 형태의 출력 값이기 때문이다. 그림 1(a)의 W_2 채널 블록도에서 보는 바와 같이 실제 출력은 y가 된다.

3.5. 부호화의 예

예로서 입력비트 u=[0 0 0 1 0 0 1 1] 이라고 하면, 이 입력비트 u를 Polar code에 의해 부호화하면, 다음과 같은 결과를 얻는다.

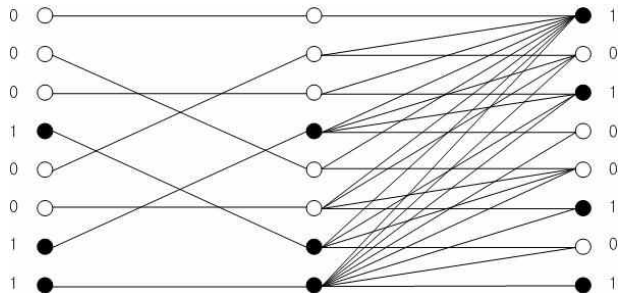


그림 10. 8×8의 부호화
Fig. 10. 8×8 Encoding.

$$[00010011]\begin{bmatrix}10000000\\10001000\\10100000\\10101010\\11000000\\11001100\\11110000\\11111111\end{bmatrix}=[10100101] \qquad (16)$$

부호화 된 값은 y=[1 0 1 0 0 1 0 1]로 나타나게 된다. 이 과정을 등가회로를 그려 표현하게 되면 그림10과 같이 그려 질 수 있는데, 등가회로는 $[G]_8 = [B]_8 \times F_2^{\otimes 3}$ 의 식을 기반으로 그려진 그림으로써 y값을 찾아 가면서 부호화 되어지는 과정을 설명 할 수 있다.

그림 10에서 보면 입력 u=[0 0 0 1 0 0 1 1]이 8×8 부호화 과정을 통해 출력 값 y=[1 0 1 0 0 1 0 1]를 갖는다.

Ⅳ. Polar code의 복호화

Polar code의 복호화를 정의하기위해 다음과 같은 내용을 기반으로 두고 설명하도록 하면, 채널 W_N와 입·출력 u_1^N, y_1^N은 확률 $W_N(y_1^N|u_1^N)$로 나타낼 수 있다. DE (Decision Element)가 1부터 N까지 정해진 뒤에 각 각 DE들의 $\hat{u}_i = u_i$ 으로 정의되어진 값들 중 그 전에 결정된 $\hat{u}_1^{i-1}$값을 받을 때 까지 기다리고, 그 값들을 모두 다 받게 되면 LR(Likelyhood Ratio)을 추정하게 된다.

Binary-input 채널 쌍 $W' : \mathcal{X} \to \tilde{\mathcal{Y}}$과 $W'' : \mathcal{X} \to \tilde{\mathcal{Y}} \times \mathcal{X}$은 이진 입력채널 $W : \mathcal{X} \to \mathcal{Y}$의 두 독립적인 값들을single-step 변형으로 얻어 진다. 따라서 다음과 같이 $(W,W) \mapsto (W',W'')$로 나타나게 되고 $u_1, u_2 \in \mathcal{X}, y_1, y_2 \in \mathcal{Y}$이란 기반 아래 채널의 복호화를 위한 기본 식이 정의된다.

$$W'(f(y_1,y_2)|u_1) \quad (17)$$
$$=\sum_{u_2'}\frac{1}{2}W(y_1|u_1\oplus u_2')W(y_2|u_2')$$

$$W''(f(y_1,y_2),u_1|u_2) \quad (18)$$
$$=\frac{1}{2}W(y_1|u_1\oplus u_2)W(y_2|u_2)$$

이 식은 그림 1(a)에서 보인 W_2 채널 블록도를 의미하고 있다. 부호화 시 입력 값과 출력 값을 대수적으로 표현하면 다음과 같은 $u_1+u_2 = y_1$, $u_2 = y_2$ 연산 식으로 표현된다.

이를 복호화 하기위해서는 u1에서 바라본 W′의 값은 두 개로 나누어진 채널의합으로 표현될 수 있고, u2에서 바라본 W′′의 값은 나누어진 하나의 채널로 써 표현이 가능하다.

우선, 다음과 같이 치환하면서 식을 전개해 나간다.

$$W\leftarrow W_N^{(i)},\quad W'\leftarrow W_{2N}^{(2i-1)},$$
$$W''\leftarrow W_{2N}^{(2i)},\ u_1\leftarrow u_{2i-1},$$
$$u_2\leftarrow u_{2i},\qquad y_1\leftarrow\left(y_1^N,u_{1,o}^{2i-2}\oplus u_{1,e}^{2i-2}\right),$$
$$y_2\leftarrow\left(y_{N+1}^{2N},u_{1,e}^{2i-2}\right),\ f(y_1,y_2)\leftarrow\left(y_1^{2N},u_1^{2i-2}\right)$$
$$W_2^{(1)}\left(y_1^2|u_1\right)\triangleq\sum_{u_2}\frac{1}{2}W_2\left(y_1^2|u_1^2\right) \quad (19)$$
$$=\sum_{u_2}\frac{1}{2}W(y_1|u_1\oplus u_2)W(y_2|u_2).$$

$$W_2^{(2)}(y_1^2,u_1|u_2)\triangleq\frac{1}{2}W_2(y_1^2|u_1^2) \quad (20)$$
$$=\frac{1}{2}W(y_1|u_1\oplus u_2)W(y_2|u_2).$$

(19),(20)은 W를 갖기 위한 기본 식의 변환 식이고 다음 (21),(22),(23)들로 다른 표현이 가능해진다.

$$\left(W_N^{(i)},W_N^{(i)}\right)\mapsto\left(W_{2N}^{(2i-1)},W_{2N}^{(2i)}\right) \quad (21)$$

$$W_{2N}^{(2i-1)}\left(y_1^{2N},u_1^{2i-2}|u_{2i-1}\right)$$
$$=\sum_{u_{2i}}\frac{1}{2}W_N^{(i)}\left(y_1^N,u_{1,o}^{2i-2}\oplus u_{1,e}^{2i-2}|u_{2i-1}\oplus u_{2i}\right)$$
$$\cdot W_N^{(i)}\left(y_{N+1}^{2N},u_{1,e}^{2i-2}|u_{2i}\right) \quad (22)$$

$$W_{2N}^{(2i)}\left(y_1^{2N},u_1^{2i-1}|u_{2i}\right)$$
$$=\frac{1}{2}W_N^{(i)}\left(y_1^N,u_{1,o}^{2i-2}\oplus u_{1,e}^{2i-2}|u_{2i-1}\oplus u_{2i}\right)$$
$$\cdot W_N^{(i)}\left(y_{N+1}^{2N},u_{1,e}^{2i-2}|u_{2i}\right) \quad (23)$$

여기서 $n\geq 0, N=2^n, 1\leq i\leq N$ 이다.

식 (22), (23)은 (19), (20)을 지수(i)를 사용하여 변형시켰다. 위에 나열해 놓은 식으로부터 LR을 구하면 다음과 같이 나타낼 수 있다.

$$L_N^{(i)}\left(y_1^N,\hat{u}_1^{i-1}\right)\triangleq\frac{W_N^{(i)}\left(y_1^N,\hat{u}_1^{i-1}|0\right)}{W_N^{(i)}\left(y_1^N,\hat{u}_1^{i-1}|1\right)} \quad (24)$$

4.1. SC(Successive Cancellation) 복호화 방법

다음은 Butterfly 구조로 된 복호화 방법으로 오른쪽에서 왼쪽으로 읽어간다.

각각 $(W_{2^i}^{(j)},W_{2^i}^{(j)})\mapsto(W_{2^{i+1}}^{(2j-1)},W_{2^{i+1}}^{(2j)})$의 형태의 채널 변형을 나타내는데 (19),(20)을 토대로 접근되어진다. 그림11을 통해 알 수 있듯이 오른쪽에는 각 각 8개의 W가 노드에 따라 정해져 있고 이는 Butterfly 구조로 연결되어져 있는 것을 알 수 있다.

그림 12를 통하여 Polar code의 SC 복호화의 과정을 나타내었다.

SC 복호화방법은 Polar code 행렬에 부호화된 “y”비트를 사용한다. 부호화된 y는 이와 인접한 각 node로부터 정보의 확률을 계산하여 구할 수 있다. (24)는 이 내

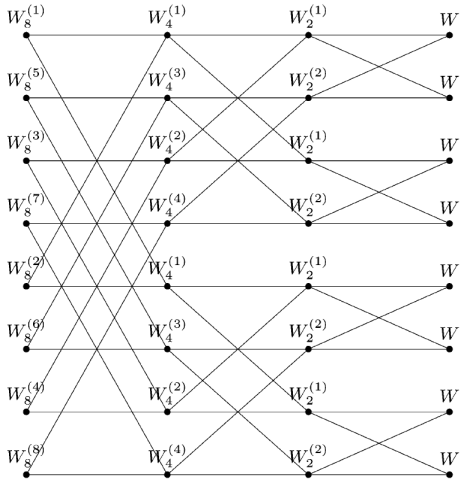


그림 11. N=8 채널들의 변환과정
Fig. 11. The channels's transformation process with N=8.

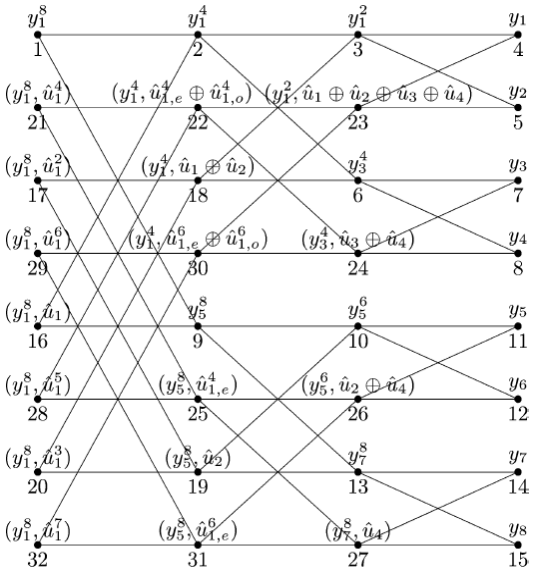


그림 12. 8×8 Polar code의 SC 복호화
Fig. 12. 8×8 Polar code's SC decoding.

용을 수식으로 표현한 것으로 LR을 찾을 수 있다. 0이 될 확률이 클 때 분자의 값이 분모의 값보다 커져서 1보다 큰 수가 되고, 반대로 1이 될 확률이 더 크면 분모의 값이 분자의 값보다 더 커져서 1보다 작은 수가 되는 것을 보인다.

이로써, 입력 추정치 û값은 아래와 같이 결정된다.

$$\hat{u}_i = \begin{cases} 0, & \text{if } L_N^{(i)}(y_1^N, \hat{u}_1^{i-1}) \geq 1 \\ 1, & \text{otherwise} \end{cases} \quad (25)$$

Bit-reversed index order의 순서로 복호화 하므로 그림12에서 node1, node21, node17, node29, node16, node28, node20, node32의 순서로 복호화가 진행된다.

예를 들면 node26의 $L_8^{(2)}(y_5^6, \hat{u}_2 \oplus \hat{u}_4)$를 보면, y_5^6은 계산되어진 LR값을 가리키고, $\hat{u}_2 \oplus \hat{u}_4$은 이 노드에서 26번째로 계산되어진 값을 말한다.

이 node 들은 Butterfly형식으로 서로 연결되어 값이 나타나게 된다. DE_1과 DE_2는 Butterfly형태로 엮여있고, 또 node1과 연결된 node2역시도 node 3,18,6과 함께 엮여져있다. node1은 이와 인접한 node2와 node9로부터 정보를 받고 그 확률을 계산해 0인지 1인지를 판단한다. node2와 node9는 각각 node3, node6과 node10, node13과 인접해있다.

이렇게 node2와 node9도 위와 같은 방법으로 각각의 인접한 node로부터 정보를 받아 0인지 1인지를 판단하게 된다. 마찬가지로, node3, node6과 node10, node13도

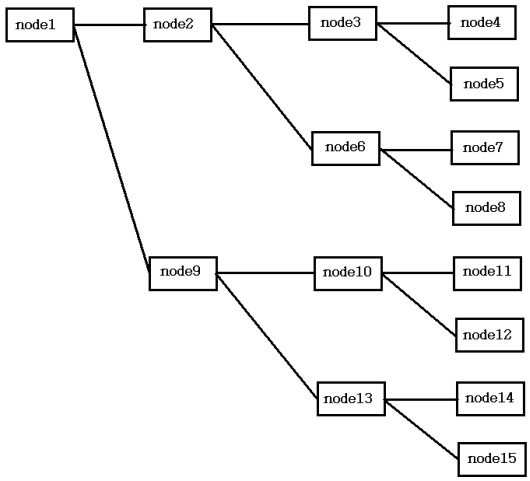


그림 13. DE1의 연결 node
Fig. 13. DE1's connection node.

각각의 node4, node5와 node7, node8과 node11, node12와 node14, node15와 인접해있고, 이렇게 인접한 node로부터 0과 1일 가능성을 받아 그 확률을 계산하여 그와 왼쪽으로 인접한 node로 정보를 보내준다.

결국, 모든 node를 통하게 되었는데 각각의 node를 통해 얻어진 정보는 node2와 node9의 계산으로 node1의 복호화 추정 값 즉, $\hat{u}_1$ 을 구하게 된다. 이 과정을 좀 더 쉽게 알아보기 위해서 DE_1의 블록도를 그림12에서 나타냈다.

위의 그림 13은 node1의 복호화 추정 값 û을 구하는 과정을 설명하였다. 이 그림을 보면 node1은 부호화된 모든 값들을 가지고 복호화 한다.

4.2. 4×4 Polar code 복호화

다음 (19), (20)을 가지고 Binary 행렬 채널에서의 복호화 과정을 보인다[8].

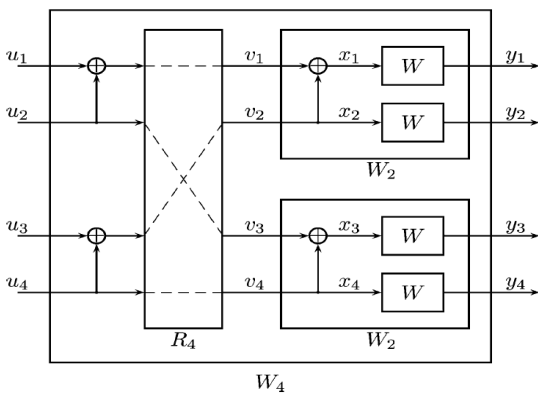


그림 14. Binary 채널
Fig. 14. Binary Channel.

(1) 채널에 Error가 존재하지 않을 경우

입력 신호 $u_1=0,\ u_2=0,\ u_3=1,\ u_4=1$ 이 전달되어 진다고 가정한다. 그리고 확률 p=0.2이라고 놓고 입력추정치를 찾는다.

$$
\begin{aligned}
&\cdot\ W(y_1|u_1+u_2+u_3+u_4=0)=0.8\\
&\cdot\ W(y_1|u_1+u_2+u_3+u_4=1)=0.2\\
&\cdot\ W(y_1|u_2+u_4=0)=0.2\\
&\cdot\ W(y_1|u_2+u_4=1)=0.8\\
&\cdot\ W(y_1|u_3+u_4=0)=0.8\\
&\cdot\ W(y_1|u_3+u_4=1)=0.2\\
&\cdot\ W(y_1|u_4=0)=0.2\\
&\cdot\ W(y_1|u_4=1)=0.8
\end{aligned}
\tag{26}
$$

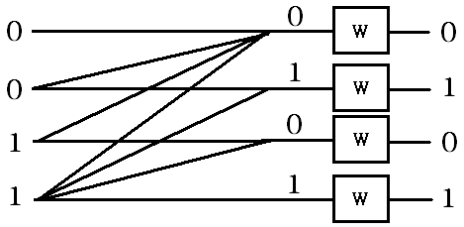


그림 15. Error 가 존재 하지 않을 경우의 입력과 출력
Fig. 15. Original Input & Output.

먼저 $u_1=0$ or 1 의 값을 고정시키고, u_2,u_3u_4 값은 계속 바꾸어 계산한다.

1) $u_1=0$이고 u_2,u_3u_4 값이 바뀔 때,

$$
\begin{aligned}
&W_4{}^1({y_1}^4|u_1=0)=\frac{1}{4}W(y_1|u_1+u_2+u_3+u_4=0)\cdot W(y_2|u_2+u_4=0)\cdot W(y_3|u_3+u_4=0)\cdot W(y_4|u_4=0)\\
&+\frac{1}{4}W(y_1|u_1+u_2+u_3+u_4=1)\cdot W(y_2|u_2+u_4=1)\cdot W(y_3|u_3+u_4=1)\cdot W(y_4|u_4=1)\\
&+\frac{1}{4}W(y_1|u_1+u_2+u_3+u_4=1)\cdot W(y_2|u_2+u_4=0)\cdot W(y_3|u_3+u_4=1)\cdot W(y_4|u_4=0)\\
&+\frac{1}{4}W(y_1|u_1+u_2+u_3+u_4=0)\cdot W(y_2|u_2+u_4=1)\cdot W(y_3|u_3+u_4=0)\cdot W(y_4|u_4=1)\\
&+\frac{1}{4}W(y_1|u_1+u_2+u_3+u_4=1)\cdot W(y_2|u_2+u_4=1)\cdot W(y_3|u_3+u_4=0)\cdot W(y_4|u_4=0)\\
&+\frac{1}{4}W(y_1|u_1+u_2+u_3+u_4=0)\cdot W(y_2|u_2+u_4=0)\cdot W(y_3|u_3+u_4=1)\cdot W(y_4|u_4=1)\\
&+\frac{1}{4}W(y_1|u_1+u_2+u_3+u_4=1)\cdot W(y_2|u_2+u_4=0)\cdot W(y_3|u_3+u_4=1)\cdot W(y_4|u_4=1)\\
&+\frac{1}{4}W(y_1|u_1+u_2+u_3+u_4=0)\cdot W(y_2|u_2+u_4=1)\cdot W(y_3|u_3+u_4=0)\cdot W(y_4|u_4=1)\\
&\Rightarrow 0.0064+0.0064+0.0004+0.1024+0.0064+0.0064+0.0064+0.0064=0.1442
\end{aligned}
\tag{27}
$$

2) $u_1=1$이고 u_2,u_3u_4 값이 바뀔 때,

$$
\begin{aligned}
&W_4{}^1({y_1}^4|u_1=1)=\frac{1}{4}W(y_1|u_1+u_2+u_3+u_4=1)\cdot W(y_2|u_2+u_4=0)\cdot W(y_3|u_3+u_4=0)\cdot W(y_4|u_4=0)\\
&+\frac{1}{4}W(y_1|u_1+u_2+u_3+u_4=0)\cdot W(y_2|u_2+u_4=1)\cdot W(y_3|u_3+u_4=1)\cdot W(y_4|u_4=1)\\
&+\frac{1}{4}W(y_1|u_1+u_2+u_3+u_4=0)\cdot W(y_2|u_2+u_4=0)\cdot W(y_3|u_3+u_4=1)\cdot W(y_4|u_4=0)\\
&+\frac{1}{4}W(y_1|u_1+u_2+u_3+u_4=1)\cdot W(y_2|u_2+u_4=1)\cdot W(y_3|u_3+u_4=0)\cdot W(y_4|u_4=1)\\
&+\frac{1}{4}W(y_1|u_1+u_2+u_3+u_4=0)\cdot W(y_2|u_2+u_4=1)\cdot W(y_3|u_3+u_4=0)\cdot W(y_4|u_4=0)\\
&+\frac{1}{4}W(y_1|u_1+u_2+u_3+u_4=1)\cdot W(y_2|u_2+u_4=1)\cdot W(y_3|u_3+u_4=1)\cdot W(y_4|u_4=0)\\
&+\frac{1}{4}W(y_1|u_1+u_2+u_3+u_4=1)\cdot W(y_2|u_2+u_4=0)\cdot W(y_3|u_3+u_4=1)\cdot W(y_4|u_4=1)\\
&+\frac{1}{4}W(y_1|u_1+u_2+u_3+u_4=0)\cdot W(y_2|u_2+u_4=0)\cdot W(y_3|u_3+u_4=0)\cdot W(y_4|u_4=1)\\
&\Rightarrow 0.0016+0.0256+0.0016+0.0256+0.0256+0.0016+0.0016+0.0256=0.1088
\end{aligned}
\tag{28}
$$

위에서 계산된 값들을 (24)에 대입해보면 ${L_N}^i(y_1^N,\hat{u}^{i-1})\geq 1$ 이라는 것을 알 수 있다. 이 값을 기반으로 입력추정치 û값을 찾을 수 있다.

$$
[W_4{}^1({y_1}^4|u_1=0)\ >\ W_4{}^1({y_1}^4|u_1=1)]\ \Rightarrow\ \hat{u}_1=0. \tag{29}
$$

$\hat{u}_1$값을 찾았으므로, 이 값은 고정시키고 û2 값도 0 or 1로 고정시킨다. 그리고 u_3,u_4는 값을 바꾸어 계산한다.

3) $u_1=0, u_2=0$이고 u_3u_4 값이 바뀔 때,

$$W_4^{\,2}(y_1^{\,4}|u_2=0)=\frac{1}{4}W(y_1|u_1+u_2+u_3+u_4=0)\cdot W(y_2|u_2+u_4=0)\cdot W(y_3|u_3+u_4=0)\cdot W(y_4|u_4=0)$$
$$+\frac{1}{4}W(y_1|u_1+u_2+u_3+u_4=1)\cdot W(y_2|u_2+u_4=1)\cdot W(y_3|u_3+u_4=1)\cdot W(y_4|u_4=1)$$
$$+\frac{1}{4}W(y_1|u_1+u_2+u_3+u_4=1)\cdot W(y_2|u_2+u_4=0)\cdot W(y_3|u_3+u_4=1)\cdot W(y_4|u_4=0)$$
$$+\frac{1}{4}W(y_1|u_1+u_2+u_3+u_4=0)\cdot W(y_2|u_2+u_4=1)\cdot W(y_3|u_3+u_4=0)\cdot W(y_4|u_4=1)$$
$$\Rightarrow 0.0064+0.0064+0.0004+0.1024=0.1156 \quad (30)$$

4) $u_1=0, u_2=1$이고 u_3u_4 값이 바뀔 때,

$$W_N^{\,2}(y_1^4|u_2=1)=\frac{1}{4}W(y_1|u_1+u_2+u_3+u_4=1)\cdot W(y_2|u_2+u_4=1)\cdot W(y_3|u_3+u_4=0)\cdot W(y_4|u_4=0)$$
$$+\frac{1}{4}W(y_1|u_1+u_2+u_3+u_4=0)\cdot W(y_2|u_2+u_4=0)\cdot W(y_3|u_3+u_4=1)\cdot W(y_4|u_4=1)$$
$$+\frac{1}{4}W(y_1|u_1+u_2+u_3+u_4=0)\cdot W(y_2|u_2+u_4=1)\cdot W(y_3|u_3+u_4=1)\cdot W(y_4|u_4=0)$$
$$+\frac{1}{4}W(y_1|u_1+u_2+u_3+u_4=1)\cdot W(y_2|u_2+u_4=0)\cdot W(y_3|u_3+u_4=0)\cdot W(y_4|u_4=1)$$
$$\Rightarrow 0.0064+0.0064+0.0064+0.0064=0.0256 \quad (31)$$

위에서 계산된 값들을 (24)에 대입해보면 $L_N^{\,i}(y_1^N,\hat{u}^{i-1})\geq 1$ 이라는 것을 알 수 있고, 이 값을 기반으로 입력추정치 û값을 찾을 수 있다.

$$[W_4^{\,2}(y_1^{\,4}|u_2=0) > W_4^{\,2}(y_1^{\,4}|u_2=1)] \Rightarrow \hat{u}_2=0. \quad (32)$$

$\hat{u}_2$값을 찾았으므로, 이 값 역시 $\hat{u}_1$과 함께 고정시키고 $\hat{u}_3$값도 0 or 1로 고정시킨다. 그리고 u_4는 값을 바꾸어 계산한다.

5) $u_1=0, u_2=0, u_3=0$이고, u_4 값이 바뀔 때,

$$W_4^{\,3}(y_1^{\,4}|u_3=0)=\frac{1}{4}W(y_1|u_1+u_2+u_3+u_4=0)\cdot W(y_2|u_2+u_4=0)\cdot W(y_3|u_3+u_4=0)\cdot W(y_4|u_4=0)+\frac{1}{4}W(y_1|u_1+u_2+u_3+u_4=1)\cdot W(y_2|u_2+u_4=1)\cdot W(y_3|u_3+u_4=1)\cdot W(y_4|u_4=1)$$
$$\Rightarrow 0.0064+0.0064=0.0128 \quad (33)$$

6) $u_1=0, u_2=0, u_3=1$이고, u_4 값이 바뀔 때,

$$W_4^{\,3}(y_1^{\,4}|u_3=1)=\frac{1}{4}W(y_1|u_1+u_2+u_3+u_4=1)\cdot W(y_2|u_2+u_4=0)\cdot W(y_3|u_3+u_4=1)\cdot W(y_4|u_4=0)+\frac{1}{4}W(y_1|u_1+u_2+u_3+u_4=0)\cdot W(y_2|u_2+u_4=1)\cdot W(y_3|u_3+u_4=0)\cdot W(y_4|u_4=1)$$
$$\Rightarrow 0.0016+0.1024=0.104. \quad (34)$$

위에서 계산된 값들을 (24)에 대입해보면 $L_N^{\,i}(y_1^N,\hat{u}^{i-1})\geq 1$ 이 아닌 'Otherwise'라는 것을 알 수 있고, 이 값을 기반으로 입력추정치 û값을 찾을 수 있다.

$$[W_4^{\,3}(y_1^{\,4}|u_3=0) < W_4^{\,3}(y_1^{\,4}|u_3=1)] \Rightarrow \hat{u}_3=1 \quad (35)$$

$\hat{u}_3$값을 찾았으므로, 이 값을 고정시키고 $\hat{u}_1$,$\hat{u}_2$ 값도 고정시킨다. 그리고 u_4값을 바꾸어 계산한다.

7) $u_1=0, u_2=0, u_3=1$이고, $u_4=0$일 때,

$$W_4^{\,4}(y_1^{\,4}|u_4=0)=\frac{1}{4}W(y_1|u_1+u_2+u_3+u_4=1)\cdot W(y_2|u_2+u_4=0)\cdot W(y_3|u_3+u_4=1)\cdot W(y_4|u_4=0)$$
$$\Rightarrow 0.0004. \quad (36)$$

8) $u_1=0, u_2=0, u_3=1$이고, $u_4=1$일 때,

$$W_4^{\,4}(y_1^{\,4}|u_4=1)=\frac{1}{4}W(y_1|u_1+u_2+u_3+u_4=0)\cdot W(y_2|u_2+u_4=1)\cdot W(y_3|u_3+u_4=0)\cdot W(y_4|u_4=1)$$
$$\Rightarrow 0.1024. \quad (37)$$

위에서 계산된 값들을 (24)에 대입해보면 $L_N^{\,i}(y_1^N,\hat{u}^{i-1})\geq 1$이 아닌 'Otherwise'라는 것을 알 수 있어, 이 값을 기반으로 입력추정치 û값을 찾을 수 있다.

$$[W_4^{\,4}(y_1^{\,4}|u_4=0) < W_4^{\,3}(y_1^{\,4}|u_4=1)] \Rightarrow \hat{u}_4=1 \quad (38)$$

결과적으로, 입력추정치 û1,û2,û3,û4를 위와 같이 얻음으로써 입력 값을 재연해 낼 수 있다.

(2) 채널에 Error가 존재할 경우

입력 신호 $u_1=0,\ u_2=0,\ u_3=1,\ u_4=1$ 이 전달되어 진다고 가정한다. 그리고 확률 p=0.2이라고 놓고 입력추정치를 찾는다.

여기에서 입력 $u_1=0,\ u_2=0$는 Frozen bit이고 $u_3=1,\ u_4=1$는 Information bit이다. 그리고 마지막 채널에서 Error가 존재하여 출력 값이 {0,1,0,0}을 갖는다고 가정하고 입력 값을 찾는다.

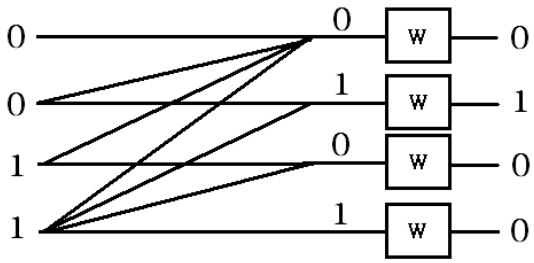


그림 16. Error 가 존재 할 경우의 입력과 출력
Fig. 16. Output obtains error bit.

- $W(y_1|u_1+u_2+u_3+u_4=0)=0.8$
- $W(y_1|u_1+u_2+u_3+u_4=1)=0.2$
- $W(y_1|u_2+u_4=0)=0.2$
- $W(y_1|u_2+u_4=1)=0.8$
- $W(y_1|u_3+u_4=0)=0.8$
- $W(y_1|u_3+u_4=1)=0.2$
- $W(y_1|u_4=0)=0.8$
- $W(y_1|u_4=1)=0.2$ (39)

먼저 u_1,u_2는 Frozen bit 이므로 $u_1,u_2=0$ 으로 고정시키고, Information bit 인 u_3u_4 값은 계속 바꾸어 계산한다.

1) $u_3=0$이고 u_4 값이 바뀔 때,

$$W_4^{3}(y_1^{4}|u_3=0)=\frac{1}{4}W(y_1|u_1+u_2+u_3+u_4=0)\cdot W(y_2|u_2+u_4=0)\cdot W(y_3|u_3+u_4=0)\cdot W(y_4|u_4=0)+\frac{1}{4}W(y_1|u_1+u_2+u_3+u_4=1)\cdot W(y_2|u_2+u_4=1)\cdot W(y_3|u_3+u_4=1)\cdot W(y_4|u_4=1)$$

$$\Rightarrow 0.0256+0.0016=0.0272. \quad (40)$$

2) $u_3=1$이고 u_4 값이 바뀔 때,

$$W_4^{3}(y_1^{4}|u_3=1)=\frac{1}{4}W(y_1|u_1+u_2+u_3+u_4=1)\cdot W(y_2|u_2+u_4=0)\cdot W(y_3|u_3+u_4=1)\cdot W(y_4|u_4=0)+\frac{1}{4}W(y_1|u_1+u_2+u_3+u_4=0)\cdot W(y_2|u_2+u_4=1)\cdot W(y_3|u_3+u_4=0)\cdot W(y_4|u_4=1)$$

$$\Rightarrow 0.0016+0.0256=0.0272 \quad (41)$$

$u_3=0, u_3=1$일 때 같은 계산 값이 나온다. 이는 확률에서 0 or 1로 추정이 가능하다. 여기서 û3=1로 고정시키고 다음 입력추정치 û4를 찾을 수 있다.

또한 u_1,u_2는 Frozen bit 이므로 $u_1,u_2=0$ 으로 고정시키고, 정보비트 인 u_3이 위에서와 같이 구해졌으므로 u_3역시 '1'로 고정시켜놓은 뒤 u_4 값을 바꾸어 계산한다.

3) $u_4=0$ 일 때,

$$W_4^{4}(y_1^{4}|u_4=0)=\frac{1}{4}W(y_1|u_1+u_2+u_3+u_4=1)\cdot W(y_2|u_2+u_4=0)\cdot W(y_3|u_3+u_4=1)\cdot W(y_4|u_4=0)$$

$$\Rightarrow 0.0016 \quad (42)$$

4) $u_4=1$ 일 때,

$$W_4^{4}(y_1^{4}|u_4=1)=\frac{1}{4}W(y_1|u_1+u_2+u_3+u_4=0)\cdot W(y_2|u_2+u_4=1)\cdot W(y_3|u_3+u_4=0)\cdot W(y_4|u_4=1)$$

$$\Rightarrow 0.0256 \quad (43)$$

$u_4=0, u_4=1$일 경우 $u_4=1$가 $u_4=0$보다 값이 크게 나와 û4=1임을 알 수 있다.

결과적으로, Frozen bit의 고정 값 '0'과 오류를 가진 출력 값을 확률로 통해 입력 값과 동일한 정보비트를 찾아낼 수 있다.

Ⅴ. 제안한 Radix 4의 Polar code

1) Radix 4의 16×16 Polar code

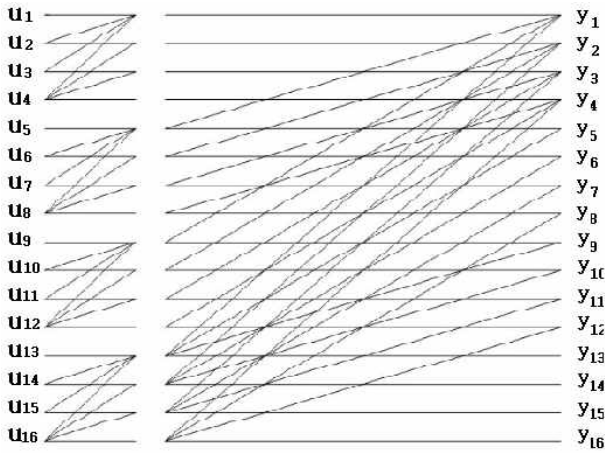


그림 17. 제안한 Radix 4 16×16 polar code
Fig. 17. Proposed Radix 4's 16×16 polar code.

다음은 Radix 4의 16×16 확장된 값은 다음과 같다.

$$[F]_{16} = ([I]4 \otimes [F]_2^{\otimes^2})([F]_2^{\otimes^2} \otimes [I]4) \quad (44)$$

위 (44)를 다음과 같은 그림 17로 나타낼 수 있다. 그림 17에서 node를 따라가다 보면 다음과 같은 결과를 얻을 수 있다.

표 2. Radix 4를 이용한 16×16 polar code 확장 결과 값
Table 2. 16x16 polar code.

입력	출력
$u_1,u_2,u_3,u_4,u_5,u_6,u_7,u_8$ $u_9,u_{10},u_{11},u_{12},u_{13},u_{14},u_{15},u_{16}$	y_1
$u_2,u_4,u_6,u_8,u_{10},u_{12},u_{14},u_{16}$	y_2
$u_3,u_4,u_7,u_8,u_{11},u_{12},u_{15},u_{16}$	y_3
u_4,u_8,u_{12},u_{16}	y_4
$u_5,u_6,u_7,u_8,u_{13},u_{14},u_{15},u_{16}$	y_5
u_6,u_8,u_{14},u_{16}	y_6
u_7,u_8,u_{15},u_{16}	y_7
u_8,u_{16}	y_8
$u_9,u_{10},u_{11},u_{12},u_{13},u_{14},u_{15},u_{16}$	y_9
$u_{10},u_{12},u_{14},u_{16}$	y_{10}
$u_{11},u_{12},u_{15},u_{16}$	y_{11}
u_{12},u_{16}	y_{12}
$u_{13},u_{14},u_{15},u_{16}$	y_{13}
u_{14},u_{16}	y_{14}
u_{15},u_{16}	y_{15}
u_{16}	y_{16}

표 2는 Radix 4의 결과 값이다. 이 결과 값을 행렬로 표현하면 다음과 같다.

$$\begin{bmatrix}
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
1&1&0&0&1&1&0&0&0&0&0&0&0&0&0&0\\
1&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
1&1&0&0&0&0&0&0&1&1&0&0&0&0&0&0\\
1&0&1&0&0&0&0&0&1&0&1&0&0&0&0&0\\
1&1&1&1&0&0&0&0&1&1&1&1&0&0&0&0\\
1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0\\
1&1&0&0&1&1&0&0&1&1&0&0&1&1&0&0\\
1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1
\end{bmatrix}$$

다음은 행렬을 이용하여 $[F]_2^{\otimes 4}$ 값을 찾는다. 위에서 언급하였던 (44)를 기반으로, 대수적인 표현을 행렬로써 나타낸다.

먼저, $[F]_2^{\otimes 4}$ radix 앞쪽수식 $([I]_4 \otimes [F]_2^{\otimes 2})$을 행렬로 나타낸다.

$([I]_4 \otimes [F]_2^{\otimes^2})=$

$$\begin{bmatrix}
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1
\end{bmatrix}$$

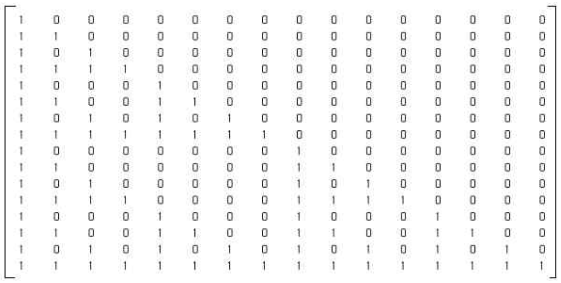
$[F]_2^{\otimes 4}$ radix 뒤쪽수식 $([F]_2^{\otimes^2} \otimes [I]_4)$을 행렬로 보이면,

$([F]_2^{\otimes^2} \otimes I_4)=$

$$\begin{bmatrix}
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0\\
1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0\\
0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0\\
0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0\\
0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1
\end{bmatrix}$$

이다.

이러한 결과 값을 가지고 (44)의 값을 행렬로 나타낸다.

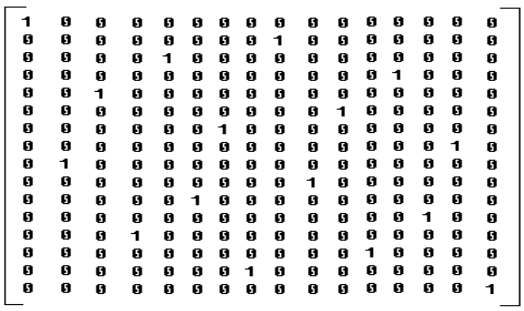


그림으로 표현 되어있는 $[F]_2^{\otimes^4}$ Radix 계산결과와 행렬로 계산 되어진 $[F]_2^{\otimes^4}$ 가 같다는 것을 알 수 있다. 따라서 확실히 $[F]_2^{\otimes^4}$ 의 값을 찾아낼 수 있다.

다음은 $[G]_{16}$을 구하기 위해 Bit-reverse 행렬을 구한다.

$$[B]_{16}=[R]_{16}([I]_2 \otimes [B]_8) \quad (45)$$

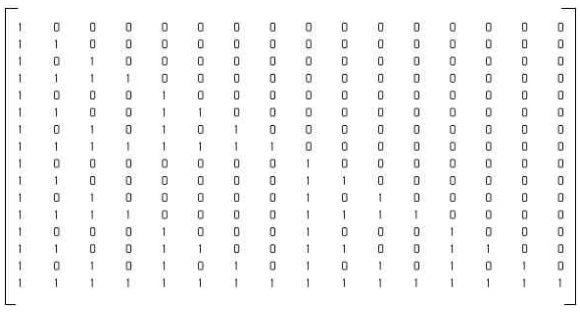
$[B]_{16}=$



이로써 Polar code 16×16 행렬을 구할 수 있다.

$$[G]_{16} = [B]_{16} \times [F]_2^{\otimes 4} \quad (46)$$

$[G]_{16} =$

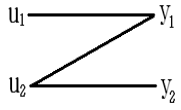
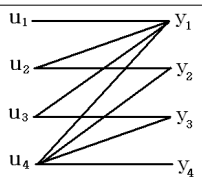


Radix 2와 Radix 4의 Polar code의 비교는 표 3에 나타냈다.

표 3에 의하면 Arikan이 제안한 Radix2보다 본 논문에서 제안한 Radix4가 2배 빠른 스피드로 부호화 할 수 있다.

표 3. Radix 2와 Radix 4 polar code 기본도와 연산속도 비교

Table 3. Compare with Radix 2 and Radix 4 polar code.

	Arikan[1]	제안
연산	Radix2 $N\log_2 N$	Radix4 $N\log_4 N$
	Ex) N=16 $16\log_2 16 = 64$	$16\log_4 16 = 32$ (2배 빠름)
기본 연결도		
기본 행렬	$F_2 =$ $\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$	$F_4 =$ $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$
확장	2^n ,n=1,2,3…	4^n ,n=1,2,3…

Ⅵ. 결 론

본 논문은 Arikan이 제안한 Polar code의 부호화와 복호화의 대수적 표현식을 행렬과 확률로 분석하여 나타냄으로써, Polar code의 부호화와 복호화의 구조와 체계를 전반적으로 분석했다.

그리고 결론적으로, Arikan이 제안한 Radix2와 본 논문에서 제안한 Radix4를 비교해본 결과 Radix4가 Radix2보다 연산속도비가 2배 빨라지는 것을 알 수 있다. 앞으로 이에 대한 H/W 구현 등의 연구가 더 요구되고 있다.

참 고 문 헌

[1] Erdal Arikan, “Channel Polarization : A Method for Constructing Capacity-Achieving codes for Symmetric Binary-Input Memoryless Channel”, IEEE Transactions on Information Theory, Vol .55, No.7, July 2009.

[2] Ryuhei Mori & Toshiyuki Tanaka “Performance and Construction of Polar codes on Symmetric Binary-Input Memoryless Channels”, IEEE ISIT 2009, Seoul, Korea, June 28-July 3, 2009.

[3] Rüdiger Urbanke, Satish Babu Korada, Eren Sasoğlu, “Polar Codes : Characterization of

Exponent, Bounds and Constructions", IEEE ISIT 2009, Seoul, Korea, June 28-July 3, 2009.
[4] Erdal Arikan, "On the rate of channel polarization", IEEE ISIT 2009, Seoul, Korea, June 28-July 3, 2009.
[5] Satish Babu Korada, Eren Sasoğlu, "A Class of Transformations that Polarize Binary-Input Memoryless Channels", IEEE ISIT 2009, Seoul, Korea, June 28-July 3, 2009.
[6] Erdal Arikan, 2009, 7, 25, private e-mail.
[7] Erdal Arikan, "Channel Combining and Splitting for Cutoff Rate Improvement" IEEE Transactions on Information Theory, VOL.52, NO.2, FEB 2006.
[8] 이문호 · Arikan 교수 "Polar Jaket code" 공동세미나, Turkey Bilkent University, August 20, 2009.

저 자 소 개

이 문 호(정회원)
1967년 전북대학교 전자공학과 학사
1984년 전남대학교 전기공학과 박사
1990년 동경대학교 정보통신공학과 박사
1980년 10월～현재 전북대학교 전기전자컴퓨터 공학부 교수
<주관심분야 : MIMO, OFDM, OFDMA, Polar Codes, 네트워크 보안, 무선통신, Cryptography, 정보이론, 디지털 통신>

최 은 지(학생회원)
2009년 전북대학교 전자공학과 4학년 재학

양 재 승(학생회원)
1988년 연세대학교 금속공학과 학사
1995년 연세대학교 산업정보 석사
2007년 9월～현재 전북대학교 정보보호공학과 박사 과정
<주관심분야 : Cryptography, Embedded System, Polar Codes, 정보시스템, 정보보안, 정보이론>

박 주 용(정회원)
1982년 전북대학교 전자공학과 학사
1994년 전북대학교 전자공학과 박사
1991년 3월～2006년 2월 서남대학교 전자공학과부 교수
2007년 3월～현재 신경대학교 인터넷정보통신 학과 부교수
<주관심분야 : MIMO, OFDM, OFDMA, Polar Codes, 무선통신, 정보이론, 디지털 통신>