論文 96-21-8-15

순환코드를 이용한 효율적 버스트 동기/에러 검출 방법 및 성능분석

正會員 최 양 호*

An Efficient Method and Performance Analysis for Burst Synchronization/Error Detection Using Cyclic Codes

Yang-Ho Choi* *Regular Member*

요 약

순환코드를 채널에러 뿐만아니라 버스트(또는 타임슬롯)동기의 검출에도 이용하면 버스트에 동기필드가 필요치않아 이에 따른 오버헤드를 줄일 수 있다. 본 논문에서는 순환코드를 이용하여 버스트 동기와 에러를 복합 검출하는 시스템에서 단 한번의 CRC(cyclic redundancy code) 디코딩 만으로 이를 수행하는 효율적인 방법을 제안하였다. 기존의 방식에서는 한번의 CRC 디코딩을 하여 버스트 동기를 찾은 후 에러 검출을 위해 다시 CRC 디코딩을 하여 두번의 디코딩 과정이 필요하다. 제안된 방법은 처리 시간의 단축과 시스템 구현을 용이하게할 수 있는 장점이 있으며 동기검출 성능은 기존의 방식과 동일하다.

채널에러가 발생한다면 복합 검출 시스템은 실제 전송된 코드워드가 아닌 다른 코드워드를 오검출할 수 있다. 오검출 확률은 검출방법에 좌우되지않고 발생한 전송에러 특성에 의해 결정된다는 사실에 착안, 간단한 과정을 통해 오검출 확률을 새롭게 유도하여 정확한 표현식을 제시하였다.

ABSTRACT

Cyclic Codes can be used for burst(or time slot) synchronization as well as error detection so that the overhead bits of the burst, which would be necessary to seperate burst synchronization and error detection systems, may be eliminated. In this paper a new method for combined burst synchronization and error detection is proposed which requires CRC decoding once only, while the previous method which inspects channel error after searching for burst synchronization requires CRC decoding twice. The proposed method has the advantage of simple implementation and reducing processing time over the previous one, still showing the same detection performance.

*한국통신 무선통신연구소 광대역무선연구팀
Korea Telecom Wireless Communication Research Lab. Wideband Access Team
論文番號:95140-0408
接受日字:1995年 4月 8日

It may occur that a burst different from the actually transmitted one is falsely accepted in the presence of channel errors. The exact expression for the false acceptance probability is newly presented through a simple derivation based on the fact that it is determined by channel errors but not by detection methods.

Ⅰ. 서 론

순환코드는 주로 에러검출에 널리 이용되어 왔으나, 이뿐만아니라 버스트 동기 검출에도 이용될 수 있다[1,2]. 버스트 동기의 검출을 위해 특정 패턴을 가지는 동기필드를 두는 방식이 많이 쓰이고 있으나 버스트 크기가 크지않은 경우에는 동기필드의 오버헤드로 인해 전송 효율의 저하가 커지게된다[3]. 특히 개인통신에서와 같이 버스트 크기가 100-200 비트 정도로 작은 시스템에서는 동기필드를 두지않는 방식이 효율적이다.

TDMA 개인통신시스템에서는 일반적으로 단말기는 무선고정국에 동기를 맞추어 이를 기준으로하여 무선고정국에 버스트를 전송한다[4]. 이 때 무선고정국에서의 버스트 수신시점은 항상 일정하지않고 단말기와 무선고정국간의 거리에 따른 왕복지연, 단말기 클럭의 부정확성 등의 영향으로 다르게되어 어떤 제한된 범위의 비트슬립이 발생하게 된다. 수신시점이 일정하지않음에 따라 일어날 수 있는 버스트간의 충돌을 피하기위해 경계구간(guard time)을 두는 것이 필요하다.

최근에 상향링크에서 동기필드없이 코셋코드(coset code)를 이용하여 버스트에 발생한 비트슬립을 검출하는 방법이 제안되었다[5]. 이 방법에서는 CRC(cyclic redundancy code) 디코딩을 하여 비트슬립을 검출하고, 검출된 비트슬립에 따라 구성된 코드블럭을 다시 CRC 디코딩을 하여 에러가 있는지 조사한다. 결국 두번의 디코딩을 필요로하여 하드웨어 구현이 복잡하고 처리에 지연을 가져온다.

본 논문에서는 단 한번의 CRC 디코딩만으로 비트슬립과 에러검출을 할 수 있는 새롭고 효율적인 방법을 제안하였다. 제안된 방법에서는 한번의 CRC 디코딩한 결과를 이용 두 개의 신드롬을 구하여 비트슬립과 에러를 동시에 검출해 낸다. 버스트가 전송 중에 에러가 발생하였다면 실제 발생한 비트슬립이 아닌 다른 값으로 오검출될 수 있다. 제안방식의 오검출 성능은 기존의 방식과 동일하다.

CRC 코드를 이용하여 동기를 검출하는 시스템에 대한 오검출 확률을 구하기위해 검출방법을 따르면서 복잡한 과정을 거쳐 근사 표현식이 유도되었다[5]. 본 논문에서 오검출 확률을 간단한 과정을 통해 쉽게 유도하였으며 이의 결과는 오검출 확률에 대한 정확한 표현식을 제시한다.

Ⅱ. 버스트 동기 및 에러의 검출

코드워드의 길이가 n이고 메세지의 길이가 k인 (n, k) 순환코드의 생성함수(Generator Polynomial)를 $g(X)$라 놓자. 일반적으로 $g(X)$는 아래와 같이 쓸 수 있다.

$$g(X) = X^{n-k} + g_{n-k-1}X^{n-k-1} + \ldots + g_1 X + 1 \qquad (1)$$

순환코드워드 polynomial을 $C(X)$라 하면

$$C(X) = c_0 X^{n-1} + c_1 X^{n-2} + \ldots + c_{n-1}$$

로 n-1차 이하의 polynomial로 표현할 수 있으며 여기서 C_i는 코드워드의 i+1 번째 비트를 나타낸다. systematic 코드에서는 C_0, ..., C_{k-1}의 k 비트는 메세지 필드로 구성되고 n-k 비트의 CRC 필드가 첨가되어

$$\begin{aligned} C(X) &= m_0 X^{n-1} + m_1 X^{n-2} + \ldots + m_{k-1} X^{n-k} \\ &\quad + p_0 X^{n-k-1} + \ldots + p_{n-k-1} \\ &= X^{n-k} m(X) + p(X) \end{aligned} \qquad (2)$$

와 같이 표현할 수 있다. 여기서, m_l, $l=0, 1, \ldots, k-1$은 메세지 비트이고, p_l, $l=0, 1, \ldots, n-k-1$은 메세지 polynomial $m(X)$에 X^{n-k}를 곱하여 $g(X)$로 나눈 나머지로 CRC 비트이다. $C(X)$는 순환코드이기때문에 회전쉬프트(cyclic shift)한 polynomial도 $g(X)$의 곱인 특성이 유지된다.

순환코드의 채널에러 발생여부는 수신된 코드워드를 $g(X)$로 나눈 나머지 즉 신드롬을 계산하여 알 수 있으며, 코드워드의 일부 비트을 반전시킨 코셋코드를

이용하면 이에 더하여 버스트 동기의 검출도 가능하다. 그러나 CRC 코드를 이용한 동기의 검출은 전송 버스트의 도착시점이 어떤 주어진 시간범위 즉 검출구간에 있어야 가능하다. 다시말하면 전송된 CRC 코드워드가 채널에러 없이 수신되고 수신된 코드워드의 첫번째 비트가 검출구간에 있다면 정확하게 버스트동기를 검출할 수 있으나 이 검출구간을 벗어나면 채널에러가 없더라도 정확히 동기를 검출할 수 없다.

1. 종래 방식에 의한 검출

순환코드를 버스트 동기의 검출에도 이용할 때 보통 코드워드의 첫번째 비트와 마지막 비트를 발전시킨 코셋코드를 사용하고 있으며[5], 전송 코드워드 C_T는

$$C_T=(c_0, c_1, \ldots, c_{n-2}, \bar{c}_{n-1}) \quad (3)$$

와 같이 주어진다. 여기서 $\bar{c}$는 c의 반전을 의미한다. 이 때 검출범위는 n−k−1 비트이고 따라서 기준시점을 검출구간의 중앙으로 잡는다면 기준시점으로부터 ±d 비트 이하의 비트 슬립에 대해 동기검출이 가능하게된다[5]. d는

$$d=\left[\frac{n-k-2}{2}\right]$$

와 같이 주어진다. 여기서 [a]는 a의 소수점 이하를 잘라낸 정수를 나타낸다. +, −는 슬립 방향을 나타내는 부호로, +는 왼쪽방향으로 일어난 비트슬립으로 도착시점이 기준시점보다 빠름을, −는 오른쪽방향으로 일어난 비트슬립으로 도착시점이 기준시점보다 지연되어 있음을 의미한다. 편의상 n−k는 우수라 가정한다.

비트 시퀀스 $\ldots \gamma_{-1}\ \gamma_0\ \gamma_1 \ldots$이 수신되고 기준시점에 $\gamma_{-(d+1)}$가 위치한다고 하자. 기존의 방식에서는 기준시점에 위치한 비트와 다음 n−1 비트로 구성되는 n 비트 블럭의 첫번째 비트와 n 번째 비트를 반전하고 앞의 d+1 개의 비트를 회전쉬프트하여 구성된 블럭,

$$(r_0, r_1, \ldots, r_{k+d-1}, \bar{r}_{k+d}, \bar{r}_{-(d+1)}, r_{-d}, \ldots, r_{-1}) \quad (4)$$

에 대한 신드롬을 계산하고 이 신드롬을 표 1의 동기신드롬(timing syndrome) 패턴과 비교하여 비트슬립 S를 알아낸다[5]. 표 1에서 x는 0 또는 1이거나 상관치 않음을 의미한다. 하위 d+1 비트의 동기패턴은 회전쉬프트한 d+1 비트로 부터 형성된 패턴이다. 상위 또는 하위 d+1 비트패턴으로부터 슬립방향을 알 수 있다. 다시 말하면 상위 d+1 비트가 00...01의 패턴을 가진다면 오른쪽방향으로, 하위 d+1 비트가 10...00의 패턴을 가진다면 왼쪽방향으로 슬립이 발생했음을 나타낸다. 일어난 비트슬립의 크기는 나머지 d+1 비트패턴에서 처음 또는 마지막에 나타난 1의 위치로부터 구해진다. 예를들어 검출된 신드롬패턴이 상위 d+1 비트패턴이 00...01이고 하위 d+1 비트패턴이 10110...0이라면 S는 −3으로 3 비트만큼 오른쪽방향으로 비트슬립이 일어났음을 나타낸다. 기준시점과 전송버스트의 도착시점이 일치한다면($S=0$), 블럭 (4)는 순환코드워드이기 때문에 모든 비트가 0인 신드롬패턴을 가진다.

표 1. 기존 방식에서의 동기신드롬 패턴
Table 1. Timing syndroms obtained from the previous detection method

비트슬립(S)	동기신드롬 패턴	
	상위 d+1 비트	하위 d+1 비트
−d	0 0 0 . . . 0 0 0 1	x x x . . . x x 1
−(d−1)	0 0 0 . . . 0 0 0 1	x x x x . . . x 1 0
.	.	.
.	.	.
.	.	.
−1	0 0 0 . . . 0 0 0 1	x 1 0 0 . . . 0 0 0
0	0 0 0 . . . 0 0 0 0	0 0 0 0 . . . 0 0 0
1	0 0 0 . . . 0 0 1 x	1 0 0 0 . . . 0 0 0
.	.	.
.	.	.
.	.	.
d−1	0 1 x . . . x x x x	1 0 0 0 . . . 0 0 0
d	1 x x . . . x x x x	1 0 0 0 . . . 0 0 0

그림 1은 기존의 방식에 따른 버스트동기/에러 검출을 위한 기능 구성도이다. MARKER에 의해 첫번째 비트와 n 번째 비트는 반전되고 ROTATE는 n 비트 블럭의 앞의 d+1 개의 비트를 회전쉬프트 하여 식(4)의 비트 시퀀스를 디코더에 입력한다. 디코더는 이 시퀀스에 대한 신드롬을 구하며 SYNCH LOGIC은 구한 신드롬으로 부터 비트슬립을 알아낸다. 수신

비트 시퀀스는 최소한 비트슬립이 검출될 때까지 지연되며, BLOCK DET는 알아낸 위치정보에 의해 전송버스트로 예측되는 블럭을 출력한다. 이 블럭의 첫번째 비트와 n 번째 비트를 발전하고 CRC 디코딩을 하여 검출된 코드워드에 에러가 있는지를 조사하여 신드롬이 0이면 디코딩에 성공한 것으로 판정한다.

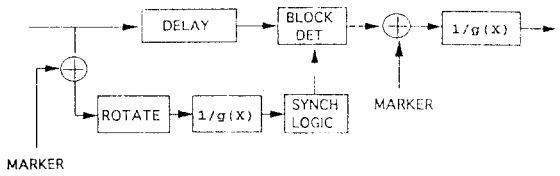


그림 1. 기존의 방법에 따른 동기/에러검출 기능 블럭도
Fig. 1 Functional block diagram of synchronization and error dectection for the previous method

기존의 방식에 있어서는 앞에서 설명한 바와 같이, 비트슬립의 검출과 코드워드의 에러 검사에 각각의 CRC 디코딩을 필요로하여 모두 두 번의 CRC 디코딩이 요구된다. 두번의 CRC 디코딩을 필요로 함에 따라 시스템 구성이 복잡해지고 처리 시간이 길어지게된다.

2. 제안 방식

그림2와 같이 2n−k 비트로 구성되는 수신블럭 $\boldsymbol{R}$을 3개의 블럭으로 나누어 생각하자.

$$\begin{aligned}\boldsymbol{R} &= (\gamma_{-(n-k)}, \gamma_{-(n-k-1)}, \ldots, \gamma_{-1}, \gamma_0, \ldots, \gamma_{n-2}, \gamma_{n-1}) \\ &= (\gamma_h, \gamma_m, \gamma_t) \end{aligned} \quad (5)$$

여기서 γ_h는 $\boldsymbol{R}$에서 앞의 n−k 비트로, γ_m은 다음 k 비트로, γ_t은 뒤의 n−k 비트로 구성되는 블럭으로

$$\begin{aligned}\gamma_h &= (\gamma_{-(n-k)}, \gamma_{-(n-k-1)}, \ldots, \gamma_{-1}), \\ \gamma_m &= (\gamma_0, \gamma_1, \ldots, \gamma_{k-1}), \\ \gamma_t &= (\gamma_k, \gamma_{k+1}), \ldots, \gamma_{n-1})\end{aligned} \quad (6)$$

이다. 편의상 γ_h를 머리블럭, γ_m을 중간블럭, γ_t를 꼬리블럭으로 부른다. 전송버스트의 도착시점에 n−k−1 비트타임 이하의 모호함이 있다고 가정하면 전송버스트를 항상 수신블럭 $\boldsymbol{R}$ 내에 있게 할 수 있다. 이 때 가장 빨리 도착하는 경우 전송버스트의 첫번째 비트와 마지막 비트는 각각 $\gamma_{-(n-k-1)}$과 γ_k에 위치하고, 가장 늦게 도착하는 경우에는 각각 γ_{-1}과 γ_{n-2}에 위치하게된다. 다시 말하면 수신블럭을 식(5)와 같이 분할할 때 항상 전송버스트의 첫번째 비트는 γ_h에, 마지막 비트는 γ_t에 있게된다.

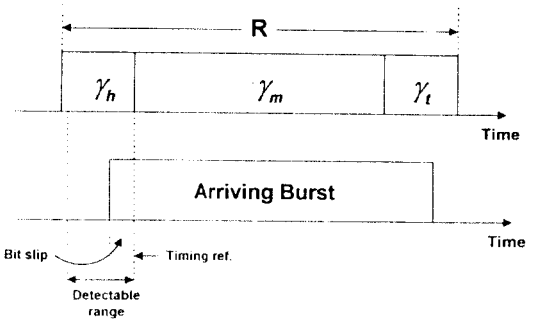


그림 2. 수신블럭에서 머리, 중간, 꼬리 블럭의 위치
Fig. 2 Positions of head, middle and tail blocks in a received block

전송에러가 없는 경우를 상정하자. 이 때 전송버스트의 첫번째 비트가 γ_h의 n−k+1−j 번째 비트에 해당한다면

$$\begin{aligned}\gamma_h &= (\gamma_{-(n-k)}, \ldots, \gamma_{-(j+1)}, c_0, c_1, \ldots, c_{j-1}), \\ \gamma_m &= (c_j, \ldots, c_{k-1+j}), \\ \gamma_t &= (c_{k+j}, \ldots, c_{n-2}, c_{n-1}, \gamma_{n-j}, \ldots, \gamma_{n-1})\end{aligned} \quad (7)$$

와 같이 주어진다. 중간블럭에 꼬리블럭을 첨가하여 구성한 n 비트블럭(γ_m, γ_t)과 중간블럭에 머리블럭을 첨가하여 구성한 블럭(γ_m, γ_h)은

$$\begin{aligned}(\gamma_m, \gamma_t) &= (c_j, \ldots, c_{k-1+j}, c_{k+j}, \ldots, c_{n-2}, c_{n-1}, c_0, \ldots, c_{j-1}) \\ &\quad +(0, \ldots, 0, 0, \ldots, 0, 1, y_{n-j}, \ldots, y_{n-1}), \\ (\gamma_m, \gamma_h) &= (c_j, \ldots, c_{k-1+j}, c_{k+j}, \ldots, c_{n-1}, c_0, c_1, \ldots, c_{j-1}) \\ &\quad +(0, \ldots, 0, y_{-(n-k)}, \ldots, y_{-(j+1)}, 1, 0, \ldots, 0)\end{aligned} \quad (8)$$

처럼 쓸 수 있다. 여기서 y_i은 랜덤비트 γ_i와 이에 대응하는 코드워드 비트의 합을 나타낸다. (γ_m, γ_t)에 대한 신드롬을 꼬리 신드롬, (γ_m, γ_h)에 대한 신드롬을 머리 신드롬이라 부르자. 식(8)에서 우변의 각 첫번째 항은 코드워드 임에 유의하면 머리, 꼬리 신드롬은 표 2와 같이 주어짐을 쉽게 확인할 수 있다. 표 2에서 항상 비트슬립이 + 값을 갖는 것은 그림 2에 보인 것처럼 중간블럭의 첫번째 비트 γ_0의 위치를 기준시점으로 정했기 때문이다. 전송버스트의 첫번째 비트는 항상 머리블럭에 존재하여 중간블럭의 첫번째 비트보다 최소 1 비트이상 왼쪽에 위치하여 비트슬립은 항상 + 값을 가지게된다. 앞에서 서술한 바와 같이 기존의 방식에서는 $\gamma_{-(d+1)}$가 위치한 검출구간의 중앙을 기준시점으로 하고있어 +, −의 비트슬립을 가진다.

$S=j$ 일 때 꼬리 신드롬은 왼쪽에서 $n-k-j$ 번째에 처음으로 1의 비트가 나타나고 머리 신드롬은 오른쪽에서 j 번째에 처음으로 1의 비트가 나타난다. 따라서 머리 또는 꼬리 신드롬에서 처음으로 나타나는 1 비트의 위치를 찾으면 비트슬립을 알 수 있게된다. 처음으로 나타나는 이러한 1의 비트는 식(8)에서 보듯이 순환코드워드의 반전으로 기인한다. 만약 머리, 꼬리 신드롬이 표 2의 $S=j$ 일 때의 패턴과 같이 주어졌다면 중간블럭에서 j 비트 만큼 왼쪽으로 쉬프트한 위치로부터 구성한 n 비트 블럭은 항상 코셋코드워드이게 된다. 결론적으로 머리, 꼬리 신드롬을 구하면 비트슬립 뿐만아니라 채널에러 존재 여부도 알 수 있다.

이러한 사실을 이용하여 다음과 같은 방법을 통해 버스트 동기 및 에러를 검출할 수 있다.

1) 중간블럭 γ_m에 n−k 개의 0 비트를 첨가하여 구성된 n 비트의 동기검출 블럭 γ_S를 CRC 디코딩하여 n−k 비트의 나머지 γ_R을 구한다. 여기서 γ_S는

$$\gamma_S=(r_0, r_1, ..., r_{k-1}, 0, 0, ..., 0) \quad (9)$$

이고, X의 polynomial로 표현하면

$$\gamma_S(X)=r_0X^{n-1}+r_1X^{n-2}+...+r_{k-1}X^{n-k} \quad (10)$$

와 같으며, $\gamma_R(X)$는 $\gamma_S(X)/g(X)$이다.

2) 나머지에 각각 γ_h와 γ_t를 더하여 머리신드롬 패턴 S_h와 꼬리신드롬 패턴 S_t를 구한다.

$$S_h=\gamma_h+\gamma_R,$$
$$S_t=\gamma_t+\gamma_R \quad (11)$$

3) 구한 머리신드롬과 꼬리신드롬 모두가 표 2에 보인 동기신드롬 패턴과 일치하면 채널에러 없이 성공적으로 버스트동기의 위치를 검출한 것으로 간주되어 검출 된 위치정보에 의해 메세지 블럭이 출력된다. 출력되는 메세지 블럭의 첫번째 비트는 반전된다. 만약 동기신드롬 패턴과 일치하지 않는다면 에러가 있는 것으로 처리된다.

한번의 디코딩으로 꼬리신드롬과 머리신드롬을 구함으로써 비트슬립 및 코드워드에 대한 에러여부가 쉽게 검출된다. 꼬리신드롬, 머리신드롬 중 하나만 구하면 발생한 비트슬립을 알 수 있으며 다른 하나의 신드롬으로부터 코드워드의 에러여부가 검사된다고 볼 수 있다. 제안된 방식의 비트슬립 검출 범위는 기존의 방식과 마찬가지로 n−k−1 비트이다.

표 2. 제안된 방식에서의 동기신드롬 패턴
Table 2. Timing syndroms obtained from the propsed detection method

비트슬립(S)	동기신드롬 패턴	
	꼬리신드롬	머리 신드롬
1	0 0 0 0 . . . 0 0 0 1 x	x x x x x . . . x x x 1
2	0 0 0 0 . . . 0 0 1 x x	x x x x x . . . x x 1 0
3	0 0 0 0 . . . 0 1 x x x	x x x x x . . . x 1 0 0
.	.	.
.	.	.
.	.	.
n−k−3	0 0 1 x . . . x x x x x	x x x 1 0 . . . 0 0 0 0
n−k−2	0 1 x x . . . x x x x x	x x 1 0 0 . . . 0 0 0 0
n−k−1	1 x x x . . . x x x x x	x 1 0 0 0 . . . 0 0 0 0

표 2를 보면, 꼬리신드롬에서 0 또는 1로 규정되어 있지않은 x 비트위치에 대응하는 머리 신드롬의 비트패턴은 규정되어있고 반대의 경우도 마찬가지로 머리신드롬에서 규정되지 않은 비트는 꼬리신드롬에서는 규정되어있다. 따라서 두 신드롬에서 x로 표시된 비트를 빼고 합쳐서 하나의 신드롬으로 만들 수 있다. 예를들어 $S=2$인 경우 합친 신드롬은 0000... 00110의 패턴을 가진다. $S=j$ 일 때 합친 신드롬 패턴

은 뒤에서 j 번째와 j+1번째만 1의 비트를 가지며, 이는 전송버스트의 앞의 j 비트를 회전쉬프트한 블럭의 신드롬과 일치한다. 따라서 제안된 방식은 발생한 비트슬립 만큼 회전쉬프트하여 신드롬 패턴을 구하는 것처럼 등가적으로 볼 수 있다.

그림 3은 제안된 방식에 따른 버스트동기/에러 검출을 위한 기능 구성도이다. 디코더에 중간블럭이 입력되면 CRC 디코딩이 시작된다. 디코더에 중간블럭이 전부 입력되면 n-k개의 0 비트가 입력되어 중간블럭에 n-k개의 0 비트를 더한 블럭에 대한 신드롬을 계산한다. SYNCH/ERROR LOGIC에서는 이 신드롬에 머리블럭과 꼬리블럭을 더해 머리, 꼬리 신드롬을 구하고 구한 결과가 표 2의 패턴과 일치하는지를 조사하여 비트슬립과 에러 발생여부를 검출해 낸다. 일치하면 검출된 비트슬립의 위치에 맞춰 k 비트의 블럭이 출력되며 이 블럭의 첫번째 비트만 반전된다. 일치하지않으면 코드워드의 검출을 실패한 것으로 간주한다.

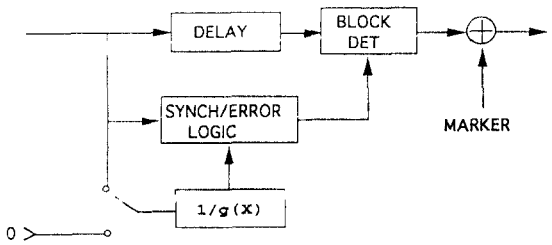


그림 3. 제안방식에 따른 동기/에러 검출 기능 구성도
Fig. 3 Functional block diagram of synchronization and error detection for the proposed method

제안된 방식에서는 단 한번 디코딩을 한다는 점외에 동기신드롬 계산과정에서 비트의 반전이 필요하지않다.

Ⅲ. 오검출 확률

2n-k-2 비트블럭에 순환코드의 첫번째와 마지막 비트를 반전시킨 코셋코드워드는 단지 하나만 존재할 수 있다. 이는 end-around 버스트를 포함하여 n-k 버스트에 에러가 있다면 에러를 검출할 수 있는 순환코드의 성질에 기인한다[6]. 다시 말하면 어떤 순환코드워드와 k 비트의 패턴은 같고 나머지 n-k 버스트의 비트패턴의 일부 또는 전부가 다른 순환코드워드는 존재하지않기 때문에 2n-k-2 비트 블럭에는 하나의 코셋코드워드만이 존재할 수 있다.

전송 중에 에러가 있으면 실제 전송된 버스트가 아닌 다른 버스트를 검출 할 수 있다. 오검출은 다음 두 가지 경우에 발생한다. 첫째 버스트 동기 위치를 잘못 검출 했지만 코드워드로 인식되거나, 둘째 동기위치는 정확히 검출됐으나 에러가 있는 코드워드에 대해 에러를 검출하지 못할 때 오검출은 발생한다. 제안된 방식의 오검출 확률(false acceptance probability)은 기존의 방식과 동일하다. 그 이유는 앞에서 설명한 바와 같이 수신 블럭 내에 코드워드가 있다면 하나만 존재하고 제안된 방식이나 기존의 방식은 그 코드워드의 위치를 정확히 검출한다. 코드워드의 위치를 정확히 검출할 수 있다면 검출방식에 관계없이 오검출 확률은 전송에러 특성에 정해지며 따라서 두 방식의 오검출 확률은 동일하다.

전송에러에 의해 결정된다는 사실을 이용하면 복잡한 검출방식을 따름이 없이 쉽게 오검출확률 $\Pr(fa)$을 유도할 수 있다. 실제 전송된 버스트의 비트슬립을 S라 하고 검출된 비트슬립을 $\hat{S}$라 하자. $\Pr(fa)$는 다음과 같이 표현할 수 있다.

$$\Pr(fa) = \sum_{S} \Pr(fa|S)\Pr(S)$$

$$= \sum_{S}\sum_{\hat{S}} \Pr(\hat{S}|S)\Pr(S) \qquad (12)$$

여기서 $\Pr(S)$는 전송버스트가 S의 비트슬립을 일으킬 확률로 시스템 특성에 종속된 값이며 $\Pr(\hat{S}|S)$는 S의 실제 비트슬립이 $\hat{S}$의 비트슬립으로 검출될 확률로 전송에러 특성에 의해 결정된다. 수신버스트가 항상 검출구간에 있다고 가정하고 Ⅱ.2에서와 같이 기준시점을 정하면 1보다 작거나 n-k-1보다 큰 S에 대해서는 $\Pr(S)$는 0이다. $\Pr(\hat{S}|S)$은 $\hat{S}$으로 검출된 블럭이 순환 코드워드라는 사실을 이용하여 유도할 수 있다.

비트슬립 $\hat{S}$로 검출된 n 비트 블럭을 C_R이라 하자. 예를들어 $\hat{S}-S$가 3 이라면 C_R은 전송버스트와 에러벡터 E로 분해하여 다음과 같이 표현된다.

$$C_R = (r_0, \ldots, r_{n-2}, r_{n-1})$$

$$=(r_0, r_1, r_2, \bar{c}_0, c_1, ..., c_{n-4}) + (0, 0, 0, e_0, e_1, ..., e_{n-4}) \quad (13)$$

여기서 e_i는 메세지블럭의 i 번째 비트에 대한 에러 비트를 나타낸다. $\Pr(\hat{S}|S)$를 구하기위해서는 C_R을 코셋코드워드로 만드는 모든 에러패턴을 구하고 이러한 에러패턴을 일으킬 확률 $\Pr(E)$의 계산이 필요하다. 각각의 에러패턴의 발생은 서로 독립적이므로

$$\Pr(\hat{S}=S+3|S)=2^{-3}\sum_{E}\Pr(E), \quad 1\le S\le n-k-1 \quad (14)$$

로 주어진다. 2^3으로 나눈 이유는 검출블럭이 코셋코드워드이기위해서는 3 개의 랜덤비트는 어떤 특정한 비트이어야하기 때문이다. i 개의 전송에러를 가지는 에러패턴이 N_i개 있다면 식(14)는

$$\Pr(\hat{S}=S+3|S)=2^{-3}\sum_{i=1}^{n-3} N_i p^i (1-p)^{n-3-i} \quad (15)$$

와 같이 주어진다. 여기서 p는 BER이며 랜덤에러 채널을 가정하였고 N_i는 실제 전송버스트에서 3개의 랜덤비트를 제외한 나머지 n−3개의 비트에서 i개의 에러가 있는 에러패턴의 수이다. 식(13)에서 보듯이 전송버스트중 뒤에서 3 비트, C_{n-3}, C_{n-2}, C_{n-1}은 실제 전송버스트의 위치가 S일때 $S+3$으로 디코딩될 확률과 무관하다. 검출된 벡터가 코셋코드워드이라는 사실을 이용하여 N_i를 계산할 수 있다. C_R은 첫번째 비트와 n번째 비트를 반전시키면 순환코드워드로 된다. 식(13)을 첫번째와 n 번째 비트를 반전하고 왼쪽 방향으로 4 비트 회전쉬프트 시키면

$$(c_1, c_2, ..., c_{n-3}, \bar{c}_{n-4}, \bar{r}_0, r_1, r_2, \bar{c}_0) + (e_1, ..., e_{n-1}, 0, 0, 0, e_2) \quad (16)$$

처럼 되며, 식(16)의 첫번째 벡터는

$$(c_1, ..., c_{n-3}, c_{n-4}, c_{n-3}, c_{n-2}, c_{n-1}, c_0) + (0, ..., 0, 1, y_1, y_2, y_3, 1) \quad (17)$$

와 같이 코드워드를 분리하여 쓸 수 있다. y_i은 랜덤비트와 대응하는 코드워드 비트와의 합으로 예를들면 y_1은 r_0+c_{n-3}이다. 식(16)에 대한 신드롬은 0이라는 사실과 식(17)을 이용하여 식(16)에 대한 신드롬을 구할 수 있다.

$$S=(0, ..., 0, 1, y_1, y_2, y_3, 1) \quad (18)$$

따라서 N_i는 식(16)의 두번째 벡터의 weight가 i이고 그 신드롬이 식(18)과 같은 벡터의 수이다.

$\hat{S}-S$가 −3인 경우에 대해서 위와 유사하게 코셋코드워드 C_R의 첫번째 비트와 n번째 비트를 반전시키고 1 비트 오른쪽으로 회전 쉬프트한 다음, 쉬프트된 에러벡터에 대한 신드롬을 구하면 식(18)과 똑같이 주어지며 이는 $|\hat{S}-S|$가 같다면 $\Pr(\hat{S}|S)$는 같음을 의미한다. 일반적으로 $|\hat{S}-S|$가 m이고 비트슬립 S가 검출범위 내에 있을 때 쉬프트된 에러벡터의 신드롬은

$$S=(0, ..., 0, 1, y_1, ..., y_m, 1) \quad (19)$$

으로 $\hat{S}$, S에 값에 관계없이 $|\hat{S}-S|$이 같다면 오검출을 일으키는 에러벡터는 동일한 신드롬 패턴을 가진다.

실제 전송버스트의 위치가 S일 때 이로부터 m 비트만큼 어긋난 위치로 버스트 동기를 검출할 확률은

$$\Pr(|\hat{S}-S|=m|S)=2^{-m}\sum_{i=1}^{n-m} N_i p^i (1-p)^{n-m-i}, \quad 0\le m\le n-k-2 \quad (20)$$

와 같이 표현된다. N_i는 식(19)와 같은 신드롬을 가지는 에러벡터 수이다. 이 값은 단지 $\hat{S}$과 S간의 차에 의해 정해져서 이 차가 같으면 $\hat{S}$ 또는 S에 관계없이 식(20)은 같은 값을 가지게된다. 위의 결과는 m=0에 대해서도 적용할 수 있으며 이 때 에러벡터의 신드롬이 (0, ..., 0, 1, 1)이면 오검출이 발생한다.

BER이 1/2일 때 식(20)은

$$\Pr(|\hat{S}-S|=3|S)=2^{-n}\sum_{i=1}^{n-3} N_i \quad (21)$$

와 같이 주어진다. 여기서 m은 편의상 3으로 가정하였다. $\sum_{i=1}^{n-3} N_i$은 식(16)의 두번째 벡터와 같은 형태의 전체 2^{n-3}개의 벡터로 구성되는 벡터공간(vector space)에서 식(18)과 같은 신드롬을 가지는 벡터의 갯수이다. 이러한 벡터는 3개의 비트가 0으로 규정되었지만

표 3. 오검출을 일으키는 에러패턴 수
Table 3. The numbers of error vectors to cause false detection

$\lvert\hat{S}-S\rvert=m$	0	1	2	3	4	5	6	7	8	9	10	11	12
N_1	0	0	0	0	0	0	0	2	1	4	12	18	42
N_2	0	6	6	6	12	18	38	96	160	330	764	1412	2746
N_3	0	37	116	282	552	1153	2260	4716	8870	17692	35544	68948	134826

신드롬은 3개의 비트가 0 또는 1 어떠한 값을 가질 수 있어 구하는 벡터의 갯수는 2^n 벡터로 구성되는 벡터 공간에서 특정 신드롬을 가지는 벡터의 갯수와 동일하게되며 따라서 2^k이다. 이를 식(21)에 대입하면 $1/2^{n-k}$로 계산된다. 다른 m에 대해서도 위와 유사한 과정을 거치면 같은 결과를 얻을 수 있다. 이러한 결과는 직관에 의해 유도될 수 있다. 즉 랜덤하게 선택된 어떤 한 벡터가 특정 코셋 코드워드에 속할 확률은 $1/2^{n-k}$이므로 특정 위치로 오검출될 확률은 앞에서 유도한 결과와 같이 $1/2^{n-k}$로 주어진다. 전체 검출구간 $n-k-1$ 비트위치에서 오검출이 발생할 수 있으므로 오검출 확률은 $(n-k-1)/2^{n-k}$이다.

BER이 매우 작아 에러가 3개 이상 발생할 확률이 2 개 이하 발생할 확률보다 훨씬 작다면 식(20)은 다음과 같이 i=1과 2에 대한 합으로 근사화 시킬 수 있을 것이다.

$$\Pr(|\hat{S}-S|=m|S)\sim 2^{-m}\sum_{i=1}^{2}N_i p^i(1-p)^{n-m-i} \qquad (22)$$

표 3은 콤퓨터 시뮬레이숀을 통해 구한 N_i값을 보여준다. 여기서 채널은 Binary Symmetric Channel(BSC) 채널로 가정하였으며, [5]에서와 같은 (161, 147) 순환 코드를 이용하였다. 표 3에 보인 에러패턴 수는 [5]에서 구한 결과와 비교할 때 i=1인 경우는 같지만 i=2인 경우는 약간의 차이가 있다. 본 분석에서는 랜덤 비트가 차지하는 부분은 제외하고 나머지 코드워드에서 발생한 에러패턴 수를 구했으나, [5]에서의 값은 전 코드워드에 대하여 구한 에러패턴 수에서 1을 뺀 결과와 일치함을 확인하였다. 앞의 유도과정에서 보듯이 본 분석 방법이 정확하다.

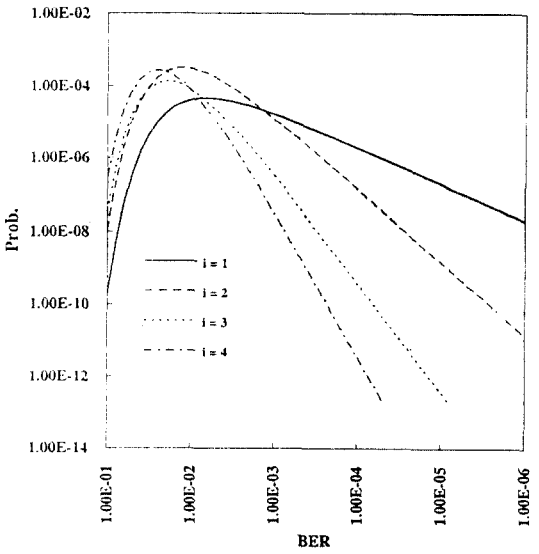
콤퓨터 시뮬레이숀에 의하지 않고도 아래와 같이 N_i의 대략적인 값을 추정할 수 있다.

$$N_i\sim\frac{\binom{n-m}{i}2^m}{2^{n-k}} \qquad (23)$$

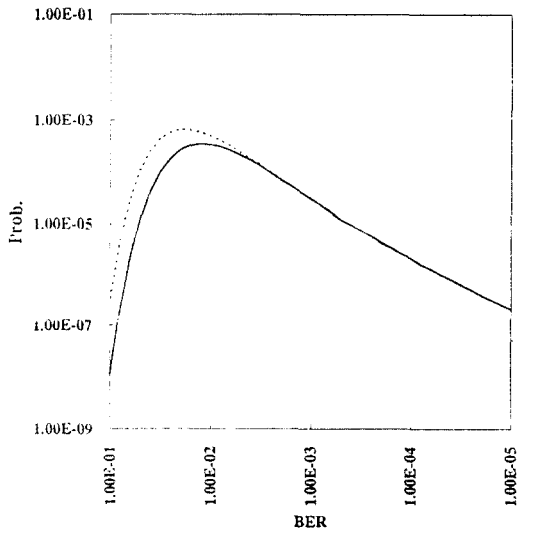
에러 패턴에 대한 신드롬 패턴이 균일하게 분포한다고 가정하면 어떤 한 에러패턴에 대한 신드롬이 특정 패턴일 확률은 $1/2^{n-k}$이다. 분모의 첫번째 항은 에러가 발생할 수 있는 비트의 수가 n-m일 때 i개의 에러가 발생하는 가지수이고 두번째 항은 실제 동기위치가 m 만큼 어긋난 것으로 검출되게하는 신드롬의 가지수이다. 실제로 신드롬 패턴은 균일하게 분포되어 있지 않지만, m과 i가 크면 추정식은 실제값에 가깝게 추정할 수 있다. 식(21)에 의한 계산과 표 2와 비교한 결과, i=1, 2, 3일 때 각각 m이 11, 6, 3 이상이면 ±20% 이하의 오차로 추정할 수 있었고 i=2, 3 일 때 각각 m이 11, 5 이상이면 ±5% 이하의 오차로 추정할 수 있었다.

표 3의 결과를 이용하여 에러 수 N_i의 오검출 성능에 대한 영향을 그림 4에 나타내었다. A_i는 발생한 에러수가 i개 일 때의 오검출 확률이다. 그림 4(a)는 발생한 에러수의 오검출 확률에 대한 기여 정도를 보여준다. BER이 작아질수록 기여 정도는 작아짐을 알 수 있다. 그림 4(b)에서도 이를 확인할 수 있다. 그림 4(b)는 N_i를 2개 까지 더한 결과와 4개 까지 더한 결과의 비교로 BER이 2×10^{-3} 보다 작으면 두 결과가 실질적으로 같아 두 선이 중첩되어있음을 보여준다. BER이 작아질수록 3비트 이상에서 에러가 발생하여 오검출될 가능성은 하나나 두개에 비해 매우 작아지게되어 식(22)와 같이 근사시켜 오검출 확률을 구할 수 있다.

BER이 2×10^{-3} 이하에 대해 식(22)를 이용하여 비트슬립의 함수로 오검출 확률 $\Pr(fa|S)$를 그림 5에 나타내었다. $S=7$일 때 dip이 발생하는 데, 이는 i=1의 오검출 확률에 대한 기여분은 m이 0, 1, ..., 6 모두에 대해 N_1이 0이므로 없고 2개 이상의 비트에러가 발생할 때 오검출되기 때문이다. BER이 10^{-4} 보다 작으면 10^{-5}이하의 오검출 성능을 얻을 수 있음을 보여준다.



(a)



(b)

그림 4. 발생 에러 갯수의 오검출 확률에 대한 영향, S = 5
(a) A_i(i = 1, 2, 3, 4) (b) $\sum_{i=1}^{2} A_i$(실선), $\sum_{i=1}^{4} A_i$(점선)

Fig. 4 Effect of the numbers of errors on the false acceptance probability, S = 5 (a) A_i(i = 1, 2, 3, 4) (b) $\sum_{i=1}^{2} A_i$ (solid line), $\sum_{i=1}^{4} A_i$(dashed line)

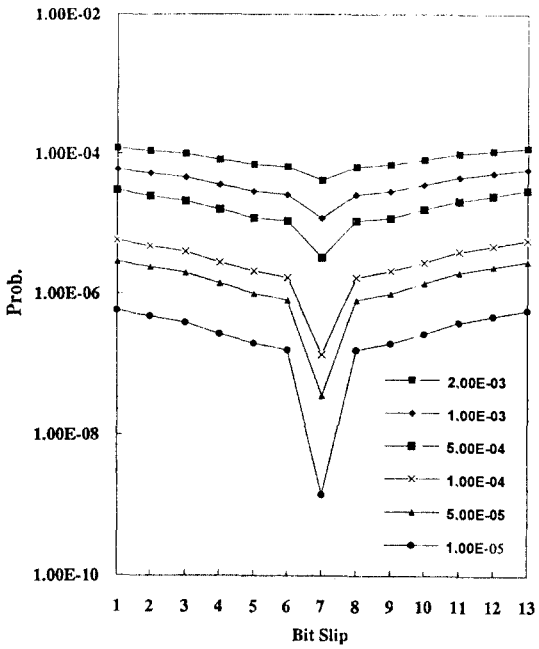


그림 5. 비트슬립에 대한 오검출 확률, Pr(fa|S)

Fig. 5 False acceptance probabilitiy in terms of bit slip, Pr (fa|S)

Ⅳ. 결 론

CRC 코드를 에러 및 버스트동기 검출에 이용하는 시스템에서 한 번의 CRC 디코딩에 의해 두가지 검출 모두를 수행하는 효율적인 방법을 제안하였다. 제안된 방법은 한번의 CRC 디코딩으로 얻은 신드롬을 이용하여 동시에 머리, 꼬리 신드롬을 계산하고 구한 하나의 신드롬으로부터 버스트 동기를, 다른하나로부터 채널에러를 검출해 낸다. 단 한번의 CRC 디코딩만 필요하여 시스템 구성이 용이하고 처리속도를 빠르게 할 수 있는 이점이 있으며 검출과정에서 비트의 반전을 필요로 하지않아 시스템 구성을 더욱 간단히 할 수 있다. 제안된 방식의 동기검출 성능은 기존의 방식과 동일하다.

오검출 확률은 동기검출 방법에 관계없이 일정하며 단지 전송에러에 의해 결정된다. 이러한 사실에 기초하여 오검출 확률을 간단히 유도하여 제시하였다. 제시한 결과는 오검출 확률에 대한 정확한 수식을 나타

낸다. BER이 0.5일 때 오검출 확률은 $(n-k-1)/2^{n-k}$ 이며 구한 결과식으로 부터 이를 유도할 수 있었다. 오검출 확률의 유도 결과에 의거하여 콤퓨터 계산을 통해 (161, 147) 코드에 대한 성능 분석을 수행하였다.

참 고 문 헌

1. S. Y. Tong, "Synchronization recovery techniquies for binary cyclic codes," Bell Syst. Tech. J., vol. 45, pp. 561-596, Apr. 1966.
2. S. E. Trvares and M. Fukada, "Furtuer results on the synchronization of bynary cyclic codes," IEEE Trans. Inform. Theory, vol. IT-11, pp 238-241, Mar. 1970.
3. P. Bylanski and D. G. Ingram, Digital Transmission Systems, Peter Petergrinus, 1980.
4. N. R. Sollenberger, J. C. I. Chuang, L. F. Chang, S. Ariyavisitakul, and H. Arnold, "Architecture and impletation of an eficient and robust TDMA frame structure for a digital portable communications," IEEE Trans. Veh. Tech., vol. VT-40, pp. 250-259, Feb. 1991.
5. L. F. Chang, N. R. Sollenberger and S. Ariyarisitakul, "Performance of a TDMA portabale radio system using a cyclic block code for burst synchronization and error detection," IEEE Trans. Commun., vol. COM-41, pp. 22-31, Jan. 1993.
6. S. Lin and D. J. Costcllo, Jr., Error Control Coding. Englewood Cliffs, NJ: Prentice-Hall, 1983.

최 양 호(Yang-Ho Choi) 정회원

1982년 2월 : 연세대학교 전자공학과 (공학사)

1984년 8월 : 한국과학기술원 전기 및 전자공학과 (공학석사)

1989년 2월 : 한국과학기술원 전기 및 전자공학과 (공학박사)

1989년 3월~현재 : 한국통신 무선통신연구소 선임연구원

※주관심분야 : 이동통신, 무선접속기술, 디지틀신호처리