논문 16-41-02-02 The Journal of Korean Institute of Communications and Information Sciences '16-02 Vol.41 No.02
http://dx.doi.org/10.7840/kics.2016.41.2.163

고속철도용 트랜스폰더 텔레그램의 병렬 디스크램블링 기법

권 순 희•, 박 성 수*, 신 동 준°, 이 재 호**, 고 경 준***

Parallel Descrambling of Transponder Telegram for High-Speed Train

Soon-Hee Kwon•, Sungsoo Park*, Dong-Joon Shin°, Jae-Ho Lee**, Kyeongjun Ko***

요 약

고속으로 주행하는 열차의 정확한 위치를 차상에서 검지하기 위해서는 지상에 설치된 트랜스폰더 태그로부터 위치정보를 정확하고 신속하게 수신하는 것이 필수적이다. 본 논문에서는 고속용으로 개발중인 트랜스폰더시스템의 텔레그램 적용을 위해 텔레그램 복호화(decoding) 속도를 개선하기 위한 병렬 디스크램블링 기법을 제안하였다. 텔레그램은 유저 데이터를 스크램블링(scrambling)하는 부호화(encoding) 과정을 거쳐 트랜스폰더 태그에 저장되므로, 트랜스폰더 리더가 유저 데이터를 복호화(decoding)하는 과정에서 디스크램블링(descrambling)이 필수적이다. 본 논문에서는 디스크램블링 시프트 레지스터 회로 구조 분석을 통해 텔레그램의 병렬 디스크램블링 기법을 제안하고, 제안된 기법을 사용할 경우 기존 방식에 비해 필요 클락 수를 현저히 낮출 수 있음을 보였다.

Key Words : Transponder, Eurobalise, Telegram, Descrambling, Parallel Processing

ABSTRACT

In order to detect the exact position of high-speed train, it is necessary to obtain location information from the transponder tag installed along the track. In this paper, we proposed parallel descrambling scheme for high-speed railway transponder system, which aims for reducing the processing time required to decode telegram. Since a telegram is stored in a tag after information bits are scrambled by an encoder, decoding procedure includes descrambling of received telegram to recover the original information bits. By analyzing the structure of the descrambling shift register circuit, we proposed a parallel descrambling scheme for fast decoding of telegram. By comparing the required number of clocks, it is shown that the proposed scheme significantly outperforms the original one.

※ 본 연구는 국토교통부 철도기술연구사업인 고정밀 철도교통 위치검지 기술개발 연구비 지원에 의해 수행되었습니다. (15RTRP-B067297-03)
◆ First Author : Hanyang University Department of Electronic Engineering, soonhee.kwon@ccrl.hanyang.ac.kr, 학생회원
° Corresponding Author : Hanyang University Department of Electronic Engineering, djshin@hanyang.ac.kr, 종신회원
* Signalling & Communication Research Team, Korea Railroad Research Institute, sspark@krri.re.kr, 정회원
** Signalling & Communication Research Team, Korea Railroad Research Institute, prolee@krri.re.kr
*** Small & Medium Enterprises Cooperation Team, Korea Railroad Research Institute, kkj8000@krri.re.kr
논문번호 : KICS2015-12-394, Received December 15, 2015; Revised Januay 18, 2016; Accepted Januay 18, 2016

Ⅰ. 서 론

유럽에서는 서로 다른 국가들의 철도 상호운영과 효율적인 열차 관리를 목적으로 표준화된 ERTMS (European Railway Traffic Management System)/ETCS(European Train Control System)를 사용하고 있다. ERTMS/ETCS의 기본적인 구성은 지상장치와 차상장치간의 정보전송에 유로발리스(Eurobalise), 유로루프(Euroloop) 및 무선(GSM-R)을 사용한다. Eurobalise는 열차에 대한 정보전송과 위치검지를 위하여 선로에 설치되며, 고정 발리스(Fixed Balise)와 가변 발리스(Controlled Balise) 두 가지가 존재한다[1]. 고정 발리스는 곡선, 구배, 영구속도제한 등의 고정된 텔레그램(telegram)을 차상장치로 전송하고, 가변 발리스는 선로변제어유니트(Line-side Electronic Unit)로부터 신호변환정보 등의 가변 텔레그램을 수신하여 차상장치로 전송한다.

한국에서도 철도위치 정보의 정확성과 활용성 증대를 위하여 Eurobalise 텔레그램을 기반으로 하여 고속열차의 위치를 검지하기 위한 고속철도 트랜스폰더시스템에 관한 연구가 활발히 진행되고 있다[2-5]. 고속열차의 정확한 위치를 검지하기 위해서는 지상에 설치된 트랜스폰더 태그로부터 수신된 텔레그램을 차상에 설치된 트랜스폰더 리더에서 빠르게 복호화(decoding)하여 텔레그램에 포함된 정보를 정확히 얻는 과정이 필요하다.

텔레그램의 복호화 과정에는 부호화(encoding) 과정에서 스크램블링(scrambling)에 의해 섞인 유저 데이터 비트를 디스크램블링(descrambling) 하는 과정이 포함되어 있다[1]. 디스크램블링을 위해서는 디스크램블링 시프트 레지스터 회로(shift register circuit)를 이용하게 된다. 텔레그램의 종류에는 830 비트의 유저 데이터 비트를 갖는 긴(long) 텔레그램과 210 비트의 유저 데이터 비트를 갖는 짧은(short) 텔레그램이 있다. 이러한 텔레그램의 디스크램블링에 필요한 클락수는 긴 텔레그램의 경우 830 클락이 필요하고 짧은 텔레그램의 경우 210 클락이 필요하다. 짧은 텔레그램의 경우에는 디스크램블링 시프트 레지스터 회로에서 필요한 클락 수가 비교적 적지만 긴 텔레그램의 경우에는 4배 정도 많은 클락 수가 필요하게 되므로 두 가지의 텔레그램을 처리하는 시간 차이가 생긴다.

최근 하드웨어의 발전으로 한 클락에 소요되는 시간이 많이 줄어듦에 따라 데이터 처리 시간이 감소하였지만 더욱 빠른 데이터 처리를 위해서 병렬처리를 통한 데이터 처리가 필요하다. 또한, 다양한 분야에서 신호처리 시간을 줄이기 위한 병렬처리 연구가 활발히 진행되고 있다[6-8]. 특히, 선형 피드백(feedback) 시프트 레지스터 회로의 병렬 처리에 대한 연구 결과들이 많이 있지만[9-11] 본 논문에서 다루고 있는 피드백이 없는 디스크램블링 시프트 레지스터 회로의 병렬 처리에 대한 연구 결과는 찾아보기 힘들다.

텔레그램의 복호화 과정중 하나인 디스크램블링에서 사용되는 시프트 레지스터 회로에서는 지연(delay) 소자의 상태 값(state value)이 입력 데이터에 의하여 계속 변하고 이후의 출력 데이터에 영향을 주기 때문에 병렬처리를 위해서는 각별한 주의가 필요하다. 따라서 본 논문에서는 고속철도용 트랜스폰더시스템 개발에 적용된 텔레그램의 스크램블링 방법과 디스크램블링 방법을 소개하고 디스크램블링에서 사용되는 시프트 레지스터 회로의 새로운 병렬 처리 기법을 제안한다. 제안된 병렬 처리 기법은 긴 텔레그램과 짧은 텔레그램의 디스크램블링 처리 속도 차이를 해결하고 더 나아가 두 텔레그램의 처리 시간을 모두 단축시킬 수 있다.

Ⅱ. 텔레그램의 부호화와 복호화

트랜스폰더 태그에 저장하기 위한 텔레그램은 다음과 같은 부호화 과정을 거쳐서 생성된다[1].

1) 12개의 스크램블링(scrambling) 비트를 선택한다.
2) 원래의 유저 데이터 비트를 1단계에서 선택된 12개의 스크램블링 비트를 이용하여 섞는다.
3) 10-to-11 비트 변환 테이블(bit transformation table)을 이용하여 섞여진 비트를 10 비트씩 11 비트로 변환한다.
4) 성형 제약(shaping constraint)을 만족하는지 확인한다. 만약 만족하지 않는다면 1단계로 돌아간다.
5) 10개의 성형(shaping) 비트를 설정한다.
6) 85개의 패리티 비트(parity bit)를 형성한다.
7) 성형 제약을 만족하는지 확인한다. 만약 만족하지 않는다면 5단계로 돌아간다.

위의 7단계로 이루어진 부호화 과정을 거쳐 생성된 텔레그램은 트랜스폰더 태그에 저장되며, 열차가 트랜스폰더 태그를 통과할 때마다 열차의 리더에 전송된다. 10-to-11 비트 변환 테이블과 성형 제약은 [1]에 자세히 설명되어 있다.

열차의 리더에서 수신된 텔레그램을 복호화하는 과정은 다음과 같다[1].

1) 텔레그램의 전체 길이를 n이라고 하였을 때 n+r

만큼의 윈도우를 고려한다. r의 값은 긴 텔레그램의 경우 77이며 짧은 텔레그램의 경우 121로 지정한다. 2, 3, 4, 5단계에서 돌아온 경우, 윈도우가 이전과 동일하지 않도록 1 비트 또는 여러 비트 이동한다.

2) 윈도우 안의 처음 n개의 비트가 패리티 체크를 만족하는지 확인한다. 만족하지 않는다면 1단계로 돌아간다.
3) 윈도우 안의 처음 r개의 비트와 마지막 r개의 비트가 동일한지 확인한다. 만족하지 않는다면 1단계로 돌아간다.
4) 텔레그램의 처음 시작 비트를 찾는다. 만약 찾을 수 없다면 1단계로 돌아간다.
5) 찾아진 텔레그램을 11 비트씩 나누었을 때 모든 11 비트들이 10-to-11 비트 변환 테이블에 포함되어 있는지 확인한다. 하나라도 포함되어 있지 않다면 1단계로 돌아간다.
6) 5단계까지 만족한 텔레그램은 안전한 텔레그램으로 판단한다.
7) 인버젼 비트(inversion bit)가 1인지 확인한다. 만약 인버젼 비트가 1이라면 모든 텔레그램의 비트를 0은 1로 변환하고 1이면 0으로 변환한다.
8) 콘트롤 비트(control bit)가 (0, 0, 1)이 되는지 확인한다. 만약 (0, 0, 1)이 아니라면 알 수 없는 텔레그램 구조이므로 복호화를 종료한다.
9) 8단계까지 통과한다면 10-to-11 비트 변환 테이블을 이용하여 텔레그램의 11 비트씩 10 비트로 복원한다.
10) 텔레그램에 포함된 스크램블링 비트를 이용하여 디스크램블링(descrambling)을 실시한다.

복호화에 대한 자세한 설명은 [1]에 제시되어 있으며 본 논문에서는 부호화 과정에서의 스크램블링 기법과 복호화 과정에서의 기존 디스크램블링 기법을 주로 살펴보고 제안하는 디스크램블링의 병렬처리 기법에 대하여 설명한다. 또한, 기존의 디스크램블링 기법과 제안하는 디스크램블링 병렬처리 기법의 클락 수를 비교하여 성능이 향상되었음을 보인다.

Ⅲ. 텔레그램 스크램블링 기법

트랜스폰더 태그에 텔레그램을 저장하기 위한 부호화 과정의 일부로써 유저 데이터 비트를 스크램블링하게 된다. 스크램블링 과정에서 긴 텔레그램과 짧은 텔레그램에서 12개의 스크램블링(scrambling) 비트를 이용하여 유저 데이터 비트를 섞어주게 되며 다음 3개의 단계를 거친다[1].

3.1 스크램블링 1단계

유저 데이터 비트가 식 (1)과 같을 때 처음 10개((1)에서 가장 왼쪽 10개)의 데이터 비트를 변형한다.

$$\mathbf{u}=(u_{m-1}, u_{m-2}, \cdots, u_0) \tag{1}$$

여기서 m은 유저 데이터 비트의 개수이며 긴 텔레그램의 경우는 830이고 짧은 텔레그램의 경우는 210이다.

$$\begin{aligned} U_{k-1} &= (u_{m-1}, u_{m-2}, \cdots, u_{m-10}) \\ U_{k-2} &= (u_{m-11}, u_{m-12}, \cdots, u_{m-20}) \\ &\vdots \\ U_0 &= (u_9, u_8, \cdots, u_0) \end{aligned} \tag{2}$$

$$U'_{k-1,dec} = \sum_{j=0}^{k-1} U_{j,dec} \bmod 2^{10} \tag{3}$$

$$U'_i = U_i \quad , i = 0, 1, \cdots, k-2 \tag{4}$$

식 (2)에서와 같이 u를 사용하여 k개의 블록을 만들고 식 (3)과 같이 처음 10개 비트를 변형한다. k개의 각 블록은 10개의 비트를 갖고 있으며 k값은 긴 텔레그램의 경우 83이고 짧은 텔레그램의 경우 21이 된다. 식 (3)에서 $U_{j,dec}$은 $U_j = (\alpha_9, \alpha_8, \cdots, \alpha_0)$ 일 때 $U_{j,dec} = \alpha_9 \times 2^9 + \alpha_8 \times 2^8 + \cdots + \alpha_0 \times 2^0$의 값을 갖는다. $U'_{k-1,dec}$값을 비트로 바꾸어 10개의 비트 값 U'_{k-1}을 생성하게 된다. 처음 10개의 비트를 제외한 나머지 유저 데이터 비트는 식 (4)에서와 같이 변형시키지 않는다. 결과적으로 구해진 k개의 블록들은 식 (3)과 (4)를 이용하여 식 (5)와 같이 표기할 수 있다.

$$\begin{aligned} U'_{k-1} &= (u'_{m-1}, u'_{m-2}, \cdots, u'_{m-10}) \\ U'_{k-2} &= (u'_{m-11}, u'_{m-12}, \cdots, u'_{m-20}) \\ &\vdots \\ U'_0 &= (u'_9, u'_8, \cdots, u'_0) \end{aligned} \tag{5}$$

스크램블링 1단계를 거쳐 만들어진 비트들을 식 (6)과 같이 표기하고 스크램블링 3단계에서 사용한다.

$$\mathbf{u}' = (u'_{m-1}, u'_{m-2}, \cdots, u'_0) \tag{6}$$

3.2 스크램블링 2단계

12개의 스크램블링 비트를 이용하여 스크램블링

시프트 레지스터 회로에서 사용하는 32개의 초기 비트값들을 생성하는 단계이다. 12개의 스크램블링 비트는 식 (7)과 같이 표기한다.

$$\mathbf{b}=(b_{11}, b_{10}, \cdots, b_0) \tag{7}$$

32개의 초기 비트값들은 12개의 스크램블링 비트를 이용하여 식 (8)에서 B를 구한 후 식 (9)와 같이 만든다.

$$B = b_{11}\times 2^{11} + b_{10}\times 2^{10} + \cdots + b_0 \times 2^0 \tag{8}$$

$$S_{dec} = (2801775573 \times B) \bmod 2^{32} \tag{9}$$

식 (9)에서 얻은 S_{dec}값을 비트로 바꾸어 32개의 초기 비트값 S를 생성하게 된다. 2801775573은 랜덤수를 생성할 때 사용되는 일반적인 수이다. 스크램블링 2단계를 거쳐 만들어진 32개의 초기 비트를 식 (10)과 같이 표기하고 스크램블링 3단계에서 사용한다.

$$\mathbf{S}=(S_{31}, S_{30}, \cdots, S_0) \tag{10}$$

3.3 스크램블링 3단계

그림 1에 있는 스크램블링 시프트 레지스터 회로와 스크램블링 2단계에서 구한 32개의 초기 비트값 S를 이용하여 스크램블링 1단계에서 구한 $\mathbf{u}'$을 섞는 단계이다.

그림 1에서 $(u'_{m-1}, u'_{m-2}, \cdots, u'_0)$ 은 스크램블링 1단계에서 만들어진 $\mathbf{u}'$이며 u'_{m-1}부터 입력으로 들어가 u'_0이 마지막 입력으로 들어가게 된다. 입력 값이 들어감에 따라 s_{m-1}이 첫 출력 값이 되고 s_0이 마지막 출력 값이 된다. $\oplus$기호는 모듈로 2 덧셈(modulo 2 sum)을 의미하고 $(S_{31}, S_{30}, \cdots, S_0)$ 은 스크램블링 2단계에서 만들어진 $\mathbf{S}$이며 지연 소자의 초기 상태 값이다. $(h_{31}, h_{30}, \cdots, h_0)$ 에서 1은 연결됨을 의미하고 0은 연결되지 않음을 의미하며 $(h_{31}, h_{30}, h_{29}, h_{27}, h_{25}, h_0)$은 1이고 나머지는 0을 사용한다. 식 (11)은 3단계로 이루어진 스크램블링 과정을 모두 거치고 나온 스크램블링된 유저 데이터 비트이다.

$$\mathbf{s}=(s_{m-1}, s_{m-2}, \cdots, s_0) \tag{11}$$

Ⅳ. 텔레그램 디스크램블링 기법

트랜스폰더 태그가 리더로 전송하는 텔레그램은 스크램블링된 유저 데이터 비트 s와 12개의 스크램블링 비트 b를 포함하고 있다. 열차 하부에 설치된 트랜스폰더 리더에서는 텔레그램을 수신하여 s와 b를 구한 후 텔레그램이 오류(error)없이 수신되었을 경우, 복호화 과정 중 하나인 디스크램블링 과정을 거치게 된다[1]. 디스크램블링은 트랜스폰더 리더가 수신한 텔레그램으로부터 구해진 b를 이용하여 스크램블링 2단계에서와 같이 S를 생성한 후 그림 2와 같은 디스크램블링 시프트 레지스터 회로를 이용하여 섞여있는 유저 데이터 비트의 배열을 복구한다. 즉, s에서 $\mathbf{u}'$을 복원하게 된다.

그림 2에서 s_{m-1}이 처음 입력으로 들어가게 되고 s_0가 마지막 입력으로 들어가게 된다. 입력 값이 들어감에 따라 u'_{m-1}이 처음 출력 값으로 나오고 u'_0이 마지막 출력 값으로 나오게 된다.

그림 2에서의 $(h_{31}, h_{30}, \cdots, h_0)$ 은 '3.3. 스크램블링 3단계'에서 사용한 값과 동일하다. 디스크램블링 시프트 레지스터 회로를 이용하여 $\mathbf{u}'$을 복원하면 스크램블링 1단계에서 실시한 방법을 역으로 실시하여 $\mathbf{u}'$에서 $\mathbf{u}$를 구하게 된다. 그림 2와 같은 시프트 레지

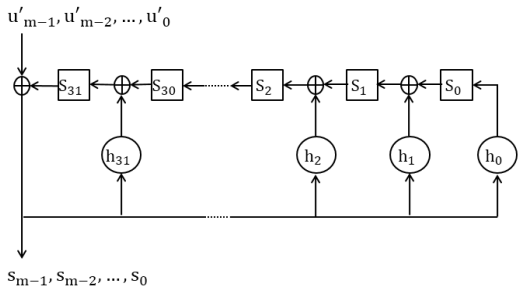

그림 1. 스크램블링 시프트 레지스터 회로
Fig. 1. Scrambling shift register circuit

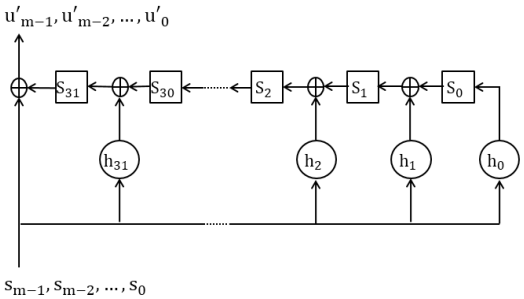

그림 2. 디스크램블링 시프트 레지스터 회로
Fig. 2. Descrambling shift register circuit

스터 회로를 사용하여 디스크램블링을 할 경우 시프트 레지스터에 필요한 클락 수는 긴 텔레그램의 경우 830이고 짧은 텔레그램의 경우 210이 된다. 본 논문에서는 두 가지 텔레그램의 디스크램블링 처리 시간 차이를 줄이고 더 나아가 두 가지 텔레그램의 디스크램블링 처리 시간을 줄이기 위하여, 디스크램블링 시프트 레지스터 회로의 병렬처리 기법을 제안한다.

V. 제안하는 텔레그램 디스크램블링 기법

트랜스폰더 리더가 태그로부터 s와 b를 오류없이 수신하였을 경우, 디스크램블링 시프트 레지스터 회로의 병렬 처리를 위해서는 다음의 3가지 단계를 거치게 된다.

첫 번째 단계는 그룹화로써 s를 병렬 처리하기 위하여 여러 개의 그룹으로 나누는 단계이다. 나누어진 그룹들은 각각 독립적인 여러 개의 시프트 레지스터 회로의 입력으로 들어가게 된다.

두 번째 단계는 여러 개로 나누어진 그룹들을 각각 입력으로 하여 처리하는 시프트 레지스터 회로를 설정하는 단계이다. 각각의 시프트 레지스터 회로에 대한 연결 상태 설정 방법과 지연 소자의 초기 값들을 정하는 단계이다.

세 번째 단계는 첫 번째 단계에서 만들어진 그룹들이 각각의 시프트 레지스터 회로에 입력으로 들어가서 나오게 된 출력 그룹들을 결합하는 방법이다. 디스크램블링 시프트 레지스터 회로는 하나의 입력이 32클락 시간 동안만 다음의 출력에 영향을 주고 그 이후의 출력에는 영향을 주지 않는다는 원리를 이용하여 결합하게 된다.

3개의 단계를 블록도로 나타내면 그림 3과 같다.

$SR(1), SR(2), \cdots, SR(g)$ 는 서로 독립적인 시프트 레지스터 회로이다. 그룹화, 시프트 레지스터 설정 방법과 결합 방법은 다음과 같다.

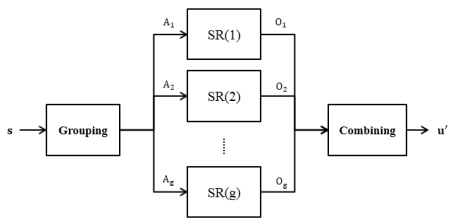

그림 3. 제안하는 병렬 디스크램블링 기법의 블록도
Fig. 3. Block diagram of the proposed parallel descrambling scheme

5.1 1단계 : 그룹화(Grouping)

수신된 유저 데이터 비트 s를 $p_1, p_2, \cdots, p_g$ 개의 비트로 나누어 g개의 그룹을 만들고 각 그룹의 뒤에 32개만큼의 영채우기(zero-padding)를 한다. 이때, $p_1, p_2, \cdots, p_g$ 는 양의 정수이고 $p_1 + p_2 + \cdots + p_g = m$ 이다.

$$\begin{aligned} A_1 &= (s_{m-1}, s_{m-2}, \cdots, s_{m-p_1}, 0, \cdots, 0) \\ A_2 &= (s_{m-p_1-1}, s_{m-p_1-2}, \cdots, s_{m-p_1-p_2}, 0, \cdots, 0) \\ &\vdots \\ A_g &= (s_{m-Q-1}, s_{m-Q-1}, \cdots, s_{m-Q-p_g}, 0, \cdots, 0) \end{aligned} \tag{12}$$

$$Q = \sum_{i=1}^{g-1} p_i \tag{13}$$

식 (12)와 같이 g개의 그룹이 만들어지게 되며 각각의 그룹은 가장 왼쪽의 비트를 처음 입력으로 하고 오른쪽 순으로 한 비트씩 시프트 레지스터의 입력으로 들어가게 된다. 영채우기를 32개씩 하는 이유는 나누어진 그룹들이 시프트 레지스터 회로의 입력으로 들어간 후 남아있는 지연 소자의 상태 값을 모두 출력으로 꺼내어 결합 과정에서 사용하기 위함이다. 이는 '5.3. 3단계 : 결합'에서 자세히 설명한다.

5.2 2단계 : 시프트 레지스터 설정

그룹화 단계를 통해 생성된 g개의 유저 데이터 비트 그룹을 병렬처리하기 위해서는 그림 3에서 보듯이 g개의 독립된 시프트 레지스터 회로가 필요하다. 또한, 그림 2의 디스크램블링 시프트 레지스터 회로에서와 같이 32개의 지연 소자의 초기 상태 값이 적용되어야 한다. 초기 상태 값은 32개의 입력에 대한 32개의 출력에만 영향을 주게 된다. 따라서 그룹 A_1만 초기 상태 값의 영향을 직접적으로 받고 나머지 그룹들은 초기 상태 값에 의하여 생긴 출력 값들의 영향을 받게 된다. 이러한 성질을 이용하여 g개의 시프트 레지스터 회로를 만들 수 있다.

SR(1)은 그림 4와 같은 디스크램블링 시프트 레지스터 회로를 사용한다. 수신된 데이터 값 b를 이용하여 S를 구하고 이를 초기 상태 값으로 사용하고, $(h_{31}, h_{30}, \cdots, h_0)$ 에서 $(h_{31}, h_{30}, h_{29}, h_{27}, h_{25}, h_0)$은 1로 하고 나머지는 0으로 한다.

그림 4에서 그룹 O_1은 그룹 A_1을 SR(1)의 입력으

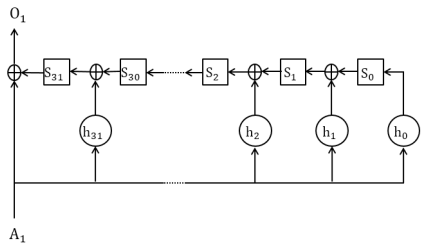

그림 4. SR(1)의 구조
Fig. 4. Structure of SR(1)

로 하였을 때의 출력 값이다. SR(1)을 제외한 SR(2), ⋯,SR(g)는 그림 5와 같은 시프트 레지스터 회로를 사용한다. 그림 5의 시프트 레지스터 회로는 초기 상태 값을 모두 0으로 하고 $(h_{31},h_{30},\cdots,h_0)$ 값은 그림 4의 SR(1)과 동일한 시프트 레지스터 회로이다.

그림 5에서 그룹 O_i는 그룹 A_i를 SR(i)의 입력으로 하였을 때의 출력 값이다. 각각의 출력 그룹은 식 (14)와 같이 표기한다.

$$O_i = (o^i_{p_i+31}, o^i_{p_i+30}, \cdots o^i_0) \tag{14}$$

각각의 출력 그룹 O_i는 그에 해당되는 입력 그룹 A_i가 갖는 비트와 동일한 개수의 비트를 포함하게 된다. 이와 같이 얻어진 g개의 출력 그룹들은 5.3장에 설명된 결합 방법을 통하여 $\mathbf{u}'$을 생성하게 된다.

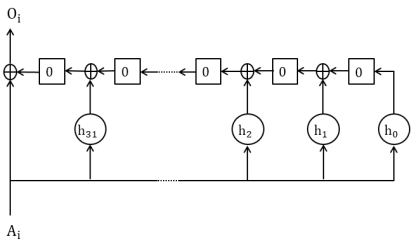

그림 5. SR(i)의 구조, i=2,3,⋯,g
Fig. 5. Structure of SR(i), i=2,3,⋯,g

5.3 3단계 : 결합 (Combining)

그룹 A_i가 SR(i)의 입력으로 들어가게 되면 영채우기에 의한 마지막 32개의 0에 의해서 p_i개의 입력이 들어간 후의 지연 소자에 남아있는 32개의 상태 값들도 그대로 출력 값으로 얻을 수 있다. 이렇게 얻어진 지연 소자의 상태 값은 SR(i+1)의 초기 상태 값으로 지정 되어야 한다. 하지만 SR(i+1)에 SR(i)의 마지막 32개의 출력 값을 초기 상태 값으로 지정하기 위해서는 SR(i)의 출력 값이 전부 얻어지기 전까지 SR(i+1)은 동작하지 않아야하기 때문에 디스크램블링 시프트 레지스터 회로의 처리 시간을 단축시킬 수 없다.

병렬처리를 적용하지 않은 디스크램블링 시프트 레지스터 회로의 출력 값을 a라고 하였을 때, 지연 소자의 초기 상태 값을 모두 0으로 정한 동일한 시프트 레지스터 회로의 동일한 입력 값에 대한 출력을 구한 후 원래의 초기 상태 값을 출력 값의 MSB(most significant bit)부터 차례대로 더하면 a와 동일한 값을 얻을 수 있다는 성질이 있다. 이러한 성질을 이용하여 병렬 처리되어 얻어진 출력 그룹 $O_1,O_2,\cdots,O_g$을 적절히 결합하여 $\mathbf{u}'$을 얻을 수 있으며, 이러한 결합 방법은 그림 6과 같다.

그림 6에서 보면 각각의 인접한 출력 그룹들의 마지막과 처음의 32개 비트들을 포개어 모듈로 2 덧셈을 하게 된다. 즉, O_i의 마지막 32개의 비트는 O_{i+1}의 처음 32개의 비트와 각각 더해지게 된다(i=1, 2, ⋯,g-1). 이러한 과정을 거쳐 얻어지게 되는 비트는 총 m+32개이고 이 중에서 마지막 32개의 비트 $(u'_{-1},u'_{-2},\cdots,u'_{-32})$를 제외한 나머지 m개의 비트를 취하면 $\mathbf{u}'$과 같아지게 된다.

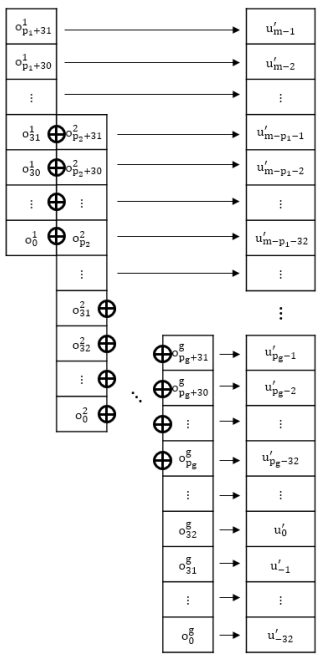

그림 6. 출력 그룹 O_i의 결합 방법, (i=1, 2,⋯,g-1)
Fig. 6. Combining method of output groups O_i, (i=1, 2, ⋯,g-1)

Ⅵ. 복잡도 및 클락 수 비교

본 논문에서 제안된 디스크램블링 병렬처리 기법과 기존의 디스크램블링 기법의 복잡도를 비교하기 위하여 지연 소자와 XOR (exclusive OR) 소자의 개수를 비교한다. 기존 디스크램블링 기법에서 사용하는 디스크램블링 시프트 레지스터 회로에서 필요한 지연 소자의 개수는 32개이고 XOR 소자의 개수는 $(h_{31}, h_{30}, \cdots, h_0)$ 에서 6개의 값만 1이기 때문에 6개가 된다. 본 논문에서 제안된 디스크램블링 병렬처리 기법에서 필요한 지연 소자의 개수는 각각의 SR(i)에서 32개씩 필요하기 때문에 총 32g개가 되고 XOR 소자의 개수는 각각의 SR(i)에서 6개가 필요하고 결합과정에서 32(g-1)개가 필요하기 때문에 총 개수는 38g-32개가 된다. 이때, 그룹화 과정에는 **s** 의 병렬화와 영채우기에 의한 0을 입력으로 추가해주는 과정만을 포함하기 때문에 지연 소자의 개수와 XOR 소자의 개수를 포함하지 않는다. 따라서 제안된 디스크램블링 병렬처리 기법을 위해서는 32(g-1)개의 지연소자, 38(g-1)개의 XOR 소자, 그리고 결합과정에서 출력 그룹 O_i $(m+32g$ bit)를 저장할 수 있는 메모리가 추가적으로 필요함에 따라 복잡도가 증가하게 된다.

클락 수를 비교하기 위하여 디스크램블링에 필요한 클락 수를 계산한다. 제안된 디스크램블링 병렬처리 기법을 사용하였을 경우, 디스크램블링을 위해 필요한 클락 수 N_{Clocks}은 식 (15)와 같이 나타낼 수 있다.

$$N_{Clocks} = \max(p_1+32, p_2+32, \cdots, p_g+32)+1 \quad (15)$$

식 (15)에서 우변 '+1' 값은 결합 과정에서 출력 그룹을 모듈로 2 덧셈을 해주기 위한 추가적인 클락 수이다. N_{Clocks} 값은 g가 고정되어 있다면 $p_1 = p_2 = \cdots = p_g$ 일 때 가장 작아지게 된다. 만약, 모든 입력 그룹들이 갖는 값의 개수를 동일하다고 한다면 필요한 클락 수 N_{Clocks}은 식 (16)과 같이 나타낼 수 있다.

$$N_{Clocks} = \frac{m}{g}+33 \quad (16)$$

본 논문에서 제안된 디스크램블링 병렬처리 기법의 지연 소자의 개수, XOR 소자의 개수, 필요한 메모리 (bit) 수와 디스크램블링을 위해 필요한 클락 수를 g가 2, 5, 10인 경우에 대하여 수치화 하면 표 1, 2, 3과 같다. 참고로 클락 수 N_{Clocks}는 처리속도 또는 처리시

표 1. g=2인 경우의 지연 소자 개수, XOR 소자 개수, 필요한 메모리 수와 필요한 클락 수
Table 1. The number of delay elements, XOR gates, needed memory (bits), and clocks for g=2

	Conventional Descrambling Scheme		Proposed Parallel Descrambling Scheme	
	Long	Short	Long	Short
# of Delay Elements	32	32	64	64
# of XOR Gates	6	6	44	44
Memory (bits)	-	-	894	274
# of Clocks	830	210	448	138

표 2. g=5인 경우의 지연 소자 개수, XOR 소자 개수, 필요한 메모리 수와 필요한 클락 수
Table 2. The number of delay elements, XOR gates, needed memory (bits), and clocks for g=5

	Conventional Descrambling Scheme		Proposed Parallel Descrambling Scheme	
	Long	Short	Long	Short
# of Delay Elements	32	32	160	160
# of XOR Gates	6	6	158	158
Memory (bits)	-	-	990	370
# of Clocks	830	210	199	75

표 3. g=10인 경우의 지연 소자 개수, XOR 소자 개수, 필요한 메모리 수와 필요한 클락 수
Table 3. The number of delay elements, XOR gates, needed memory (bits), and clocks for g=10

	Conventional Descrambling Scheme		Proposed Parallel Descrambling Scheme	
	Long	Short	Long	Short
# of Delay Elements	32	32	320	320
# of XOR Gates	6	6	348	348
Memory (bits)	-	-	1150	530
# of Clocks	830	210	116	56

간과 같음을 주목하기 바란다.

표 1, 2, 3에서 볼 수 있듯이 제안된 디스크램블링 병렬 처리 기법에서는 지연 소자의 개수와 XOR 소자의 개수가 증가하여 복잡도가 증가하지만 긴 텔레그램과 짧은 텔레그램의 처리 속도 차이를 줄일 수 있고 두 가지 텔레그램의 디스크램블링 처리 시간을 모두 현저히 감소시킬 수 있다는 장점이 있다. 이와 같은 복잡도와 처리시간의 트레이드오프(tradeoff)를 잘 활용하여 주어진 상황에 적합한 디스크램블링 기법을 사용할 수 있다.

Ⅶ. 결 론

본 논문에서 제시한 디스크램블링 병렬 처리 기법은 하드웨어 복잡도(비용)는 증가하지만 처리 속도를 높이는 기법이다. 또한, 하드웨어 복잡도를 지연 소자의 개수, XOR 소자의 개수, 메모리의 수를 통하여 분석하였으며 그에 따른 클락 수의 감소를 수치화 하여 보였다. 지연소자의 개수 D, XOR 소자의 개수를 X, 메모리의 수를 M, 그리고 클락 수를 N_{Clocks}라고 하면 그룹의 개수 g에 따라서 각각의 수치는 본 논문에서 제시한 것과 같이 $D=32g$, $X=38g-32$, $M=m+32g$, $N_{docks}=m/g+33$이 된다. 만약 D, X, M의 최댓값이 정해져 있다면 역으로 g의 최댓값을 쉽게 계산할 수 있다. 따라서 고정된 D, X, M의 최댓값에 따른 최소 N_{Clocks}값을 구할 수 있다. 이는 제안된 기법이 하드웨어 복잡도와 처리속도간의 트레이드오프를 가짐을 보이며, 위와 같이 이러한 트레이드오프를 해당 시스템에 맞도록 적절히 조절할 수 있다. 또한, 본 논문에서 제시된 내용은 피드백이 없는 하나의 시퀀스의 입력에 따른 하나의 시퀀스 출력을 얻는 시프트 레지스터의 병렬 처리 기법을 제안하였지만 다중 시퀀스의 입력을 통한 다중 시퀀스의 출력을 얻는 시프트 레지스터 회로와 같은 더욱 복잡한 시프트 레지스터에서의 병렬 처리 기법으로 확장 할 수 있을 것이다.

References

[1] UNISIG, *The Specification of FFFIS for Eurobalise*, SUBSET-036, Issue 3.0.0, Feb. 2012.

[2] K. H. Shin and J. H. Lee, "Position detection technology in railway transportation," *J. Korean Soc. Railway*, vol. 15, no. 1, pp. 16-21, Feb. 2012.

[3] S. H. Kim, J. W. Lee, H. M Ryu, and Y. C. Hwang, "A study on test method for dynamic transport response speed of eurobalise based on ETCS," *2009 Fall Conf. Korean Soc. Railway*, pp. 1731-1739, Jeju Island, Korea, Nov. 2009.

[4] M. C. Lee, C. H. Kim, J. G. Ji, and J. W. Lee, "A study on the balise failure analysis & effects for ETCS application," *2011 Fall Conf. Annual Meeting of the Korean Soc. Railway*, pp. 717-723, Jeju Island, Korea, Oct. 2011.

[5] K. J. Ko, S. S. Park, H. Y. Kim, and J. H. Lee, "Bit error rate analysis of eurobalise protocol," *2014 Fall Conf. Annual Meeting of the Korean Soc. Railway*, pp. 569-572, Jeju Island, Korea, Oct. 2014.

[6] G. S. Lim, S. H. Lee, and Y. I. Eom, "Task parallelism system of application for multicore based mobile platform," *J. KICS*, vol. 38C, no. 06, pp. 521-530, Jun. 2013.

[7] M. S. Kim and E. N. Huh, "VDI performance optimization with hybrid parallel processing in thick client system under heterogeneous multi-core environment," *J. KICS*, vol. 38B, no. 03, pp. 163-171, Mar. 2013.

[8] S. H. Yoon, J. S. Park, and M. S. Kim, "Performance improvement of a real-time traffic identification system on a multi-core CPU environment," *J. KICS*, vol. 37B, no. 05, pp. 348-356, May 2012.

[9] M. Ayinala and K. K. Parhi, "High-speed parallel architectures for linear feedback shift registers," *IEEE Trans. Signal Processing*, vol. 59, no. 9, pp. 4459-4469, Sept. 2011.

[10] J. Y. Lee, E. P. Hong, D. S. Har, and H. J. Lim, "Parallel processing architecture for parity checksum generator complying with ITU-T J.83 ANNEX B," *J. KICS*, vol. 34, no. 6, pp. 619-625, Jun. 2009.

[11] H. B. Yi, S. J. Park, P. W. Min, and C. W. Park, "A design of high performance parallel CRC generator," *J. KICS*, vol. 29, no. 9A, pp. 1101-1107, Sept. 2004.

권 순 희 (Soon-Hee Kwon)

2015년 2월 : 한양대학교 융합전자공학부 학사
2015년 3월~현재 : 한양대학교 전자컴퓨터통신공학과 석사과정
<관심분야> 디지털 통신, 오류정정부호

박 성 수 (Sungsoo Park)

2006년 2월 : 연세대학교 전기전자공학과 학사
2008년 2월 : 연세대학교 전기전자공학과 석사
2010년 8월~2011년 2월 : Purdue Univ. 방문연구원
2012년 2월 : 연세대학교 전기전자공학과 박사
2012년 3월~2013년 2월 : 연세대학교 박사후연구원
2013년 3월~현재 : 한국철도기술연구원 선임연구원
<관심분야> 철도 위치검지, LTE, LTE-R, 재난망, NFV, SDN

신 동 준 (Dong-Joon Shin)

1990년 2월 : 서울대학교 전자공학과 학사
1991년 12월 : Northwestern University 전기공학과 석사
1998년 12월 : University of Southern California 전기공학과 박사
1999년 4월~2000년 8월 : Hughes Network Systems, MTS
2000년 9월~현재 : 한양대학교 융합전자공학부 교수
<관심분야> 디지털 통신, 오류정정부호, 시퀀스, 이산수학

이 재 호 (Jae-Ho Lee)

1989년 : 광운대학교 대학원 전자공학과 석사
2005년 : 고려대학교 대학원 메카트로닉스학과 박사
1995년~현재 : 한국철도기술연구원 책임연구원
<관심분야> 열차 신호, 제어 이론

고 경 준 (Kyeongjun Ko)

2006년 2월 : 서울대학교 전기공학부 졸업
2012년 8월 : 서울대학교 전기컴퓨터공학부 박사 졸업
2012년 9월~2013년 8월 : 서울대학교 무선신호처리연구실 박사후 과정
2013년 9월~현재 : 한국철도기술연구원 선임연구원
<관심분야> 전자공학, 통신공학