Journal of The Institute of Electronics and Information Engineers Vol.54, NO.7, July 2017 https://doi.org/10.5573/ieie.2017.54.7.36
ISSN 2287-5026(Print) / ISSN 2288-159X(Online)

논문 2017-54-7-5

짝·홀 교차 사상을 이용한 Double Flow 기법 기반 병렬 터보 복호기 설계

(A Design of Parallel Turbo Decoder based on Double Flow Method Using Even-Odd Cross Mapping)

좌 유 철*, 임 종 석**

(Yu-Cheol Jwa and Chong-Suck Rim©)

요 약

오류 정정부호의 일종인 터보 코드는 우수한 BER 성능을 얻기 위하여 동일한 복호 과정을 반복 수행해야 하므로 긴 복호 시간을 필요로 한다. 따라서 복호시간을 줄이기 위하여 병렬처리를 이용할 수 있는데, 이 경우, 추가 버퍼를 필요로 하는 메모리 경합이 있을 수 있다. QPP 인터리버는 이러한 메모리 경합을 피하기 위하여 제안되었으나, double flow 복호 기법과 함께 사용하여 복호기를 구성할 경우 여전히 메모리 경합이 발생할 가능성이 있다. 본 논문에서는 double-flow 기법을 이용한 복호에서 메모리 충돌을 피할 수 있는 even-odd cross mapping 기법을 제안한다. 이 방법은 QPP 인터리버의 주소 생성 특성을 사용하며, 복호 모듈과 LLR 메모리 블록 간의 인터리빙 회로 구현에 사용될 수 있다. Double flow 기법과 제안한 방법을 적용하여 복호기를 구현하고, 이를 기존의 MDF 기법에 의한 구현과 비교하였을 때, 전체 면적은 약 8% 증가하지만, 복호시간을 최대 약 32% 줄일 수 있다.

Abstract

The turbo code, an error correction code, needs a long decoding time since the same decoding process must be repeated several times in order to obtain a good BER performance. Thus, parallel processing may be used to reduce the decoding time, in which case there may be a memory contention that requires additional buffers. The QPP interleaving has been proposed to avoid such case, but there is still a possibility of memory contention when a decoder is constructed using the so-called double flow technique. In this paper, we propose an even-odd cross mapping technique to avoid memory conflicts even in decoding using the double-flow technique. This method uses the address generation characteristic of the QPP interleaving and can be used to implement the interleaving circuit between the decoding blocks and the LLR memory blocks. When the decoder implemented by applying the double flow and the proposed methods is compared with the decoder by the conventional MDF techniques, the decoding time is reduced by up to 32% with the total area increase by 8%.

Keywords : Turbo decoder, QPP interleaving, Memory contention, Even-odd cross-mapping, MAP

Ⅰ. 서 론

터보 코드(turbo code)[1, 7] 는 통신 채널의 비트 오류 확률을 최소로 유지하는 오류 정정 부호의 일종이다. 터보 코드는 주로 MAP(maximum a posteriori)알고리즘[3, 7]을 반복 사용하여 복호를 수행하는데, 반복으로 인한 긴 복호시간을 줄이기 위하여 병렬처리 기법을 사용한다[2, 5, 9, 11]. 여기서, 병렬 처리 기법은 입력된 패킷을 나누어 한 개의 복호 모듈이 나누어진 패킷의 한 부분을 복호하여 출력한다. 즉, 길의 K의 입력 패킷을 P개의 복호 모듈이 나누어 처리 할 때, 길이 $L(=K/P)$인 각 복호 모듈의 출력을 연결하여 최종 출력을 낸다.

부분 복호 모듈의 병렬 처리 시 single flow 기법[2~3]으로는 $2L$의 복호시간이 필요하다. 또 다른 처리기법

* 학생회원, 서강대학교 (Dept. of CS&E, Sogang University)
** 평생회원, 대학교 (Dept. of CS&E, Sogang University)
© Corresponding Author (E-mail : csrim@sogang.ac.kr)
Received ; January 6, 2017 Revised ; May 25, 2017
Accepted ; June 13, 2017

Copyright © The Institute of Electronics and Information Engineers.
This is an Open-Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License (http://creativecommons.org/licenses/by-nc/3.0) which permits unrestricted non-commercial use, distribution and reproduction in any medium, provided the original work is properly cited.

인 double flow 기법[3]을 사용하면 복호시간을 L로 줄일 수 있지만, LLR(log-likelihood ratio) 값이 한 클록에 두 개씩 출력되기 때문에, QPP 인터리버[4]를 사용하여 도 메모리 경합이 발생하여 LLR 값들을 저장하기 위하여 추가 버퍼 메모리가 필요하다. 메모리 경합을 피하기 위하여 참고논문[5, 11]에서는 double flow 기법을 변형하여 복호시간이 $1.5L$로 늘어나는 MDF(modified double flow) 기법을 제안하였다.

Double flow기법의 메모리 경합은 동일 메모리로 두 개의 LLR 값이 사상되기 때문에 발생한다. LLR 값 저장에 듀얼 포트 메모리를 사용하면 경합 없이 사상이 가능하지만, 싱글 포트 메모리에 비해 그 크기가 큰 단점이 있다[11]. 본 논문에서는 double flow 기법을 사용한 복호기 구현에 싱글 포트 메모리를 사용하면서도 메모리 경합을 피할 수 있는 짝·홀 교차 사상 기법을 제시한다. 이는 복호 모듈에서 LLR 메모리 블록으로의 LLR 값 전송을 위한 인터리빙 회로 구현에 사용된다.

본 논문에서 구현한 double flow와 짝·홀 교차 사상 기법을 이용한 복호기는 MDF 기법에 의한 복호기에 비해 그 복호 시간을 최대 약 32% 줄일 수 있다. 합성을 통하여 예측한 회로의 면적을 살펴보면, 각 복호 모듈에서 두 개의 LLR 값을 동시에 계산하여야 하므로 추가 논리 회로가 필요하여 이를 위한 크기가 증가하지만, LLR 저장에 싱글 포트 메모리만을 사용하기 때문에, 기존의 MDF에 의한 복호기의 면적보다 전체 면적이 약 8% 증가하는데 그쳤다.

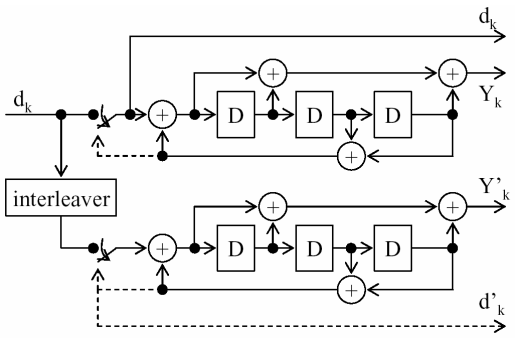

그림 1. 터보 코드 부호기[6]
Fig. 1. A turbo encoder.

본 논문의 구성은 다음과 같다. 제 2 장에서 터보 코드 부호기와 복호기, log-MAP 알고리즘, 병렬 처리 기법과 QPP 인터리버, 기존 log-MAP 알고리즘 처리 기법을 간략히 요약 소개한다. 제 3 장에서는 QPP 인터리버의 짝·홀 특성을 이용하여 double flow기법의 메모리 경합을 해결하는 메모리 사상 기법을 제시하고, 이에 따른 모듈 설계를 보인다. 제 4 장에서 구현된 전체 복호기의 구조와 동작, 그리고 실험 결과를 보이고, 제 5 장에서 본 논문의 결론을 보인다.

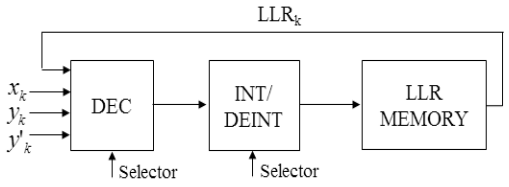

그림 2. 터보 코드 복호기
Fig. 2. A turbo code decoder.

Ⅱ. 터보 코드 알고리즘

1. 터보 코드 복호기와 부호기

가. 터보 코드 부호기

본 논문에서 사용한 터보 코드 부호기는 그림 1과 같이 두 개의 부호기로 이루어져 있고, 내부 인터리버를 통해 연결 되어 있다. 첫 번째 구성 부호기는 입력된 정보 비트 d_k를 부호화하여 첫 번째 패리티 비트(parity bit) Y_k를 출력한다. 두 번째 구성 부호기는 인터리버를 통해 순서가 재배열된 정보비트 $d_k^{'}$를 부호화하여 두 번째 패리티 비트 $Y_k^{'}$을 출력한다. 즉, 부호율 $1/3$로 d_k에 대해 튜플 $(d_k, Y_k, Y_k^{'})$를 생성하여 전송한다.

나. 송·수신 환경

본 논문에서는 AWGN 채널(Additive White Gaussian Noise Channel) 상에서 BPSK(Binary Phase Shift Keying)로 변조되어 수신 된다고 가정한다. 즉, 길이 K인 패킷은 구성 부호기를 거쳐 출력에 대해 이들이 갖는 0, 1 값이 -1, +1로 각각 바뀌어 전송된다고 가정하자. 그리고 전송 도중 평균이 0이고 분산이 σ^2인 정규분포를 갖고 서로 독립적인 잡음 p_k, q_k, r_k가 각각 더해진다고 가정하면 송신 신호$(d_k, Y_k, Y_k^{'})$에 대한 수신 신호 (x_k, y_k, y'_k)은 수식 (1)과 같다.

$$\begin{aligned} x_k &= 2(d_k - 1) + p_k \\ y_k &= 2(Y_k - 1) + q_k \\ y'_k &= 2(Y_k^{'} - 1) + r_k \end{aligned} \tag{1}$$

여기서, 이 신호에는 이미 $L_c(=2/\sigma^2)$이 곱해졌다고 가정한다[7].

다. 터보 코드 복호기

터보 코드 복호기의 일반적인 구현 형태는 그림 2와 같다. 수신 신호(x_k, y_k, y'_k)에 대해 복호 모듈이 복호를 수행하고, 이를 인터리버를 통해 순서가 재배열 되어 LLR_k값이 메모리에 저장된다. 다시 저장된 LLR_k값과 입력 값을 통해 복호를 수행하고 디인터리버를 통해 순서를 재배열하여 LLR_k값을 저장하면 한번의 반복 복호가 종료된다. 셀렉터(selector)를 통해 인터리브인지 디인터리브인지 결정하며 인터리브일 경우 x_k, LLR_k, y_k가 복호 모듈의 입력으로 들어가고 디인터리브일 경우에는 x_k, LLR_k, y'_k가 복호 모듈의 입력이 된다. 위의 과정을 반복적으로 수행하여 주어진 반복 횟수만큼 복호를 하면, 복호 모듈의 결과가 디인터리버를 거쳐 재배열된 후 출력된다.

2. Log-MAP 알고리즘

Log-MAP 알고리즘은 참고문헌[7]에 자세히 기술되어 있어, 여기서는 복호기 구현을 위한 수식만을 기술한다.

x_k, y_k 값이 L_c를 곱하여 보정된 값이고 z_k 값이 d_k에 대한 사전 정보라면 가지 값(branch metric) b_k는 수식 (2)에 의하여 계산된다.

$$b_k(d_k, Y_k) = (x_k + z_k)d_k + y_k Y_k \tag{2}$$

여기서, z_k 의 초기 값은 0이고, 이후, 이전 단계에서의 LLR 값이다. 0 또는 1을 가지는 수신 신호 d_k와 Y_k를 이용하여 b_k를 구할 수 있고, 이 값을 사용하여 FSM(foward state metric) f_k와 RSM(reverse state metric) r_k를 계산할 수 있다. f_k, r_k는 8개의 상태를 가지므로 각각 8 개의 값을 가지며, 이들의 트렐리스 다이어그램은 그림 3과 같다.

f_k들은 초기 값으로 수식 (3)과 같이 설정한다.

$$\begin{aligned} &f_0(0) = 0 \\ &f_0(1) = f_0(2) = \cdots = f_0(7) = -\infty \end{aligned} \tag{3}$$

그리고, 함수 $E(x,y)$를 수식 (4)와 같이 정의한다.

$$E(x,y) \equiv \max(x,y) + \ln(1 + e^{-|x-y|}) \tag{4}$$

나머지 값들은 $1 \le k \le K-1$, $0 \le m \le 7$ 동안 수식 (5)에 의하여 반복적으로 계산된다.

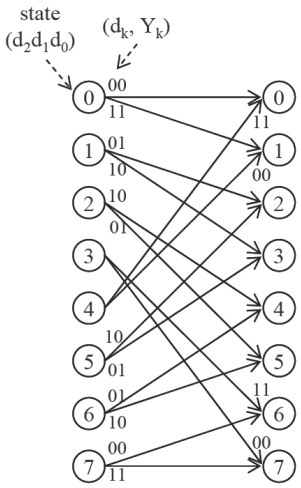

그림 3. 부호기의 트렐리스 다이어그램
Fig. 3. Trellis diagram of the encoder.

$$\begin{aligned} f_k(m) = E(&f_{k-1}(S_b^0(m)) + b_{k-1}(0, Y_{k-1}(m)), \\ &f_{k-1}(S_b^1(m)) + b_{k-1}(1, Y_{k-1}(m)) \end{aligned} \tag{5}$$

여기서, $S_b^i(m)$은 현재 상태가 m이고 입력이 i일 경우의 이전 상태 값을 의미한다.

r_k들은 초기 값으로 수식 (6)과 같이 설정되고 나머지 값들은 $1 \le k \le K+1$, $0 \le m \le 7$ 동안 수식 (7)에 의하여 반복적으로 계산된다.

$$\begin{aligned} &r_{K+2}[0] = 0 \\ &r_{K+2}[1] = r_{K+2}[2] = \cdots = r_{K+2}[7] = -\infty \end{aligned} \tag{6}$$

$$\begin{aligned} r_k(m) = E(&r_{k+1}(S_f^0(m)) + b_k(0, Y_k(m)), \\ &r_{k+1}(S_f^1(m)) + b_k(1, Y_k(m)) \end{aligned} \tag{7}$$

수식 (7)에서 $S_f^i(m)$은 현재 상태가 m이고 입력이 i일 경우의 다음 상태 값을 의미한다.

이렇게 구한 b_k, f_k, r_k들을 사용하여 LLR 값 $L(d_k)$를 계산할 수 있다. 먼저 함수 $E_{m=0}^7(a_m)$을 수식 (8)과 같이 정의한다.

$$\begin{aligned} E_{m=0}^7(a_m) = E(&E(E(a_0, a_1), E(a_2, a_3)), \\ &E(E(a_4, a_5), E(a_6, a_7))) \end{aligned} \tag{8}$$

그리고 $L^0(d_k)$와 $L^1(d_k)$를 수식 (9)와 같이 정의한다.

$$\begin{aligned} L^i(d_k) = E_{m=0}^7(&f_k(m) + r_k(S_f^i(m)) \\ &+ b_k(i, Y_k(m))), \ (i = 0, 1) \end{aligned} \tag{9}$$

그러면 수식 (10)에 의하여 d_k 값을 예측할 수 있다.

$$L(d_k) = L^1(d_k) - L^0(d_k) \tag{10}$$

계산된 $L(d_k)$ 값에서 z_k를 뺀 후 인터리브하여 사전 정보 값 z_k를 갱신한다. 두 번째 복호 모듈은 x_k 대신 0,y_k 대신 y'_k, 그리고 갱신된 z_k를 입력으로 하여 같은 계산을 반복한다. 그리고 두 번째 복호 모듈의 출력 $L(d_k)$ 값에서 z_k를 뺀 후 디인터리브하여 z_k를 갱신한다. 갱신한 z_k는 다시 수식 (2)의 z_k 값으로 사용하며 이를 반복 수행하여 복호 효율을 높인다[7].

3. QPP 인터리버

터보 코드의 복호는 다수의 복호 과정을 반복해야 하고, 각 과정에 필요한 시간은 입력 크기에 비례한다. 따라서 입력을 적당한 크기로 나누어 다수의 복호 모듈이 동시에 복호를 진행하여 복호 시간을 줄일 수 있다[9]. 여기서, 각 모듈이 생성한 LLR 값들을 동시에 저장하려면 다수의 메모리 블록이 필요한데, 이 경우, LLR 저장 과정에서 메모리 경합이 발생할 수 있다.

QPP 인터리버는 이러한 경합을 피하기 위하여 제안되었다[4, 8]. 길이가 K인 패킷에 대해, x번째 주소에 대한 인터리브된 주소는 수식 (11)과 같이 정의된다[4].

$$\Pi(x) = (f_1 x + f_2 x^2)\%K, \quad (0 \le x, f_{1,} f_2 \le K) \tag{11}$$

4. Log-MAP 알고리즘 구현 기법

길이가 K인 패킷을 길이가 L인 부분 패킷으로 나누어 $P(=K/L)$개의 복호 모듈로 각각 복호한다고 하자. 각 복호 모듈에 log-MAP 알고리즘을 구현하기 위해서는 일반적으로 다음에 보이는 세 방법을 사용할 수 있다. 물론 병렬 복호가 아닌 경우에도 이 기법들을 적용할 수 있다(이 경우, 이 절의 모든 설명에서 L 대신 K).

가. Single flow 기법[2~3]

이 방법은 그림 4와 같이 먼저 FSM을 계산하고, 그 값을 메모리에 저장한 후 RSM 계산과 동시에 저장된 FSM을 읽어 LLR 값을 계산하는 방법이다. 길이 L인 패킷에 대해 총 $2L$ 클록의 복호 시간이 필요하다.

나. Double flow 기법[3]

Double flow 기법은 FSM과 RSM을 동시에 계산하는 방법으로(그림 5), $L/2$ 클록 후부터 LLR 값을 한 클록에 두 개씩 계산할 수 있기 때문에 총 L클록의 복호 시간이 필요하다. 메모리 경합이 발생할 수 있다.

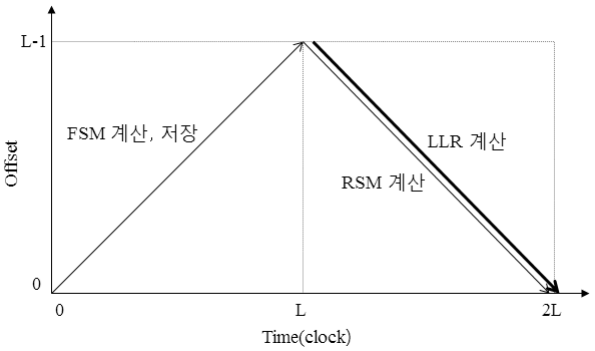

그림 4. Single flow 기법[2~3]
Fig. 4. Single flow method.

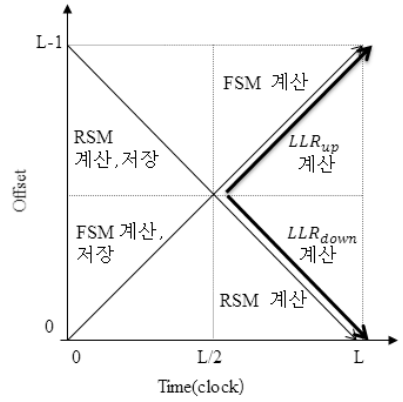

그림 5. Double flow 기법[3]
Fig. 5. Double flow method.

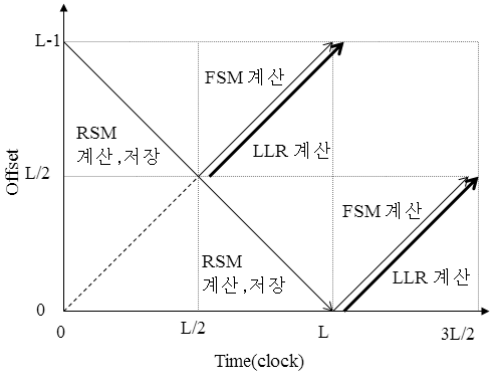

그림 6. MDF 기법[5, 11].
Fig. 6. MDF method..

다. Modified double flow(MDF) 기법[5, 11]

MDF 기법에서는 처음 $L/2$ 클록 동안 RSM을 계산한다(그림 6). $L/2$ 클록 후 이전에 저장된 RSM과 현재 계산되는 FSM을 이용하여 LLR 값을 계산한다. 동시에 RSM을 계산하여 메모리에 저장하는데, 이 경우 읽기와 쓰기가 동시에 필요하여 듀얼 포트(dual port) 메모리를 사용해야한다. 마지막 $L/2$클록 동안 나머지 LLR 값이 출력되어 복호에 총 $1.5L$ 클록이 필요하다.

Ⅲ. 짝 · 홀 교차 사상 기법

1. Double flow 기법에서의 메모리 경합

QPP 인터리버를 사용하면 식 (11)에 의하여 LLR 값 저장을 위한 주소를 생성한다. 한 번의 복호 과정에 의하여 생성된 LLR 값들의 리스트를 $LLR=[L_0, L_1, \cdots, L_{K-1}]$이라고 하자. 만일 P 개의 복호 모듈이 P 개의 LLR 값 $L_{x_i}(0 \le i \le P-1)$를 각각 동시에 생성한다면, 이들의 저장을 위하여 $\Pi(x_i)$를 계산하여야 한다. 이때, 각 x_i에 대한 해당 모듈에서의 오프셋이 모두 동일하다면, $\Pi(x_i)$로 경합 없이 LLR 메모리에 저장할 수 있다[4].

Double flow 기법을 사용한 P 개의 복호 모듈로 각각 크기가 L인 패킷을 복호한다면, 매 클록 당 두 개의 LLR 값이 출력되므로 $2P$ 개의 주소를 생성하여야 한다. 이때 LLR 메모리의 오프셋 값은 수식 (12)를 사용하여 계산한다(모듈로 K가 아닌 L이다).

$$\begin{aligned} \Pi_{LLR_{up}}(x) &= (f_1 x + f_2 x^2)\%\, L, \\ \Pi_{LLR_{down}}(x) &= (f_1(x - L/2 - 1) \\ &\quad + f_2(x - L/2 - 1)^2)\%\, L, \\ &(L/2 \le x \le L-1, 0 \le f_1, f_2 \le K) \end{aligned} \tag{12}$$

P 개의 복호 모듈이 생성한 $2P$ 개의 LLR 값을 메모리에 동시에 쓰기 위해서는 크기가 $L/2$인 $2P$개의 메모리 블록이 필요하다. 그런데 식 (12)의 입력인 각 복호 모듈에서의 두 오프셋 값이 하나는 증가하고 다른 하나는 감소하므로 이들이 모두 같을 수는 없다. 따라서 메모리 경합이 발생할 가능성이 있다.

2. 듀얼 포트 메모리 사용을 통한 해결

LLR 메모리로 크기가 L인 P개의 듀얼 포트 메모리를 사용하면 메모리 경합 문제를 해결 할 수 있다. 그림 5에 보인 LLR_{up}의 모듈 내 오프셋은 모든 복호 모듈에서 같은 값을 갖으며, 이는 LLR_{down}의 오프셋도 마찬가지다.

듀얼 포트 메모리는 쓰기 포트가 두 개 이므로(편의상 포트 A, 포트 B라고 하자) 동시에 두 개의 값을 저장할 수 있다. 따라서 모든 LLR_{up} 값들을 포트 A를 통하여 저장하고, 모든 LLR_{down} 값들을 포트 B를 통하여 저장하면 메모리 경합 없이 LLR 값들을 저장할 수 있다.

그러나 듀얼 포트 메모리는 싱글 포트 메모리에 비하여 그 면적이 크므로 이의 사용이 부담된다[10]. 특히, 병렬 터보 복호기의 경우 많은 메모리가 필요하여 메모리 면적이 논리회로를 위한 면적보다 크기 때문에, 4 장의 실험 결과에서 보인 것과 같이 다른 구현에 비하여 그 크기가 매우 크다. 다음 3 절에서는 $2P$ 개의 싱글 포트 메모리를 사용하여 LLR 값들을 메모리 경합 없이 저장할 수 있는 방법을 기술한다.

3. 짝 · 홀 메모리 사상 기법

$2P$ 개의 싱글 포트 메모리를 사용하여 구현할 경우 발생하는 메모리 경합은 메모리 그룹핑(memory grouping)을 통해 그 범위를 축소할 수 있다. $2P$ 개의 메모리 블록을 두 개씩 묶어 각 복호 모듈의 2 개의 LLR 값을 P 개의 뱅크 메모리로 사상되도록 한다(그림 7). 이 때,

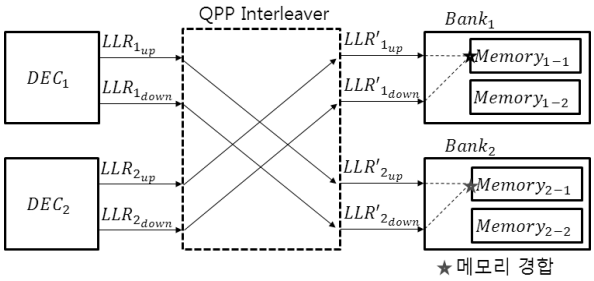

그림 7. 메모리 그룹핑
Fig. 7. Memory grouping.

뱅크 메모리는 각각 두 개의 입력 포트가 있는 것과 마찬가지이므로, 복호 모듈 내 오프셋이 증가하는(모듈끼리 같은) LLR 값들을 듀얼 메모리에서처럼 뱅크 메모리의 같은 포트를 통하여 저장하고, 오프셋이 감소하는 LLR 값들을 다른 포트로 저장하면 뱅크 메모리끼리는 경합이 발생하지 않는다. 그러나 뱅크 메모리 내의 두 메모리 블록에 오프셋 순서대로 LLR 값들을 저장하면 이 두 메모리 블록 사이에서 메모리 경합이 발생하게 된다.

가. QPP 인터리버의 주소 출력 특징

제 2 장에서 살펴본 QPP 인터리버의 주소 출력을 좀 더 자세히 살펴보자. 설명을 위해 제 2 장의 식 (11)을 식 (13)에 다시 보인다.

$$\begin{aligned} \Pi(x) &= (f_1 x + f_2 x^2)\%\, K, \\ &(0 \le x, f_1, f_2 \le K) \end{aligned} \tag{13}$$

참고문헌 [8]에 따르면, K는 모두 4의 배수이다. 이 경우 식 (13)의 f_1은 K와 서로소 관계여야 하고, f_2는 K의 인수들을 모두 포함하여야 한다[4]. 즉, 식(13)의 f_1은 K가 짝수이기 때문에 반드시 홀수가 되고 f_2는 2를 인수로 포함해야하기 때문에 짝수가 된다.

f_1는 홀수, f_2는 짝수임을 이용하여 주소 출력의 특징을 알 수 있다. 식 (13)에서 x가 짝수일 경우 $f_1 \times x$는 짝수, $f_2 \times x^2$은 짝수가 되어 $\Pi(x)$는 짝수가 되고 x가 홀수일 경우 $f_1 \times x$는 홀수, $f_2 \times x^2$은 짝수가 되어 $\Pi(x)$는 홀수가 된다. 즉, 색인 x가 0부터 $L-1$까지 증가하면서 $\Pi(x)$값은 짝·홀이 반복된다.

나. 짝 · 홀 메모리 사상 기법

짝·홀 메모리 사상 기법은 첫 번째 메모리에 색인이 짝수인 값을 저장하고, 두 번째 메모리에 색인이 홀수인 값을 저장한다. 설명을 위해 제 2 장의 식 (12)를 식 (14)에 다시 보인다.

$$\begin{aligned} \Pi_{LLR_{up}}(x) &= (f_1 x + f_2 x^2)\%\, L, \\ \Pi_{LLR_{down}}(x) &= (f_1(x - L/2 - 1) \\ &\quad + f_2(x - L/2 - 1)^2)\%\, L, \\ &(L/2 \le x \le L-1, 0 \le f_1, f_2 \le K) \end{aligned} \tag{14}$$

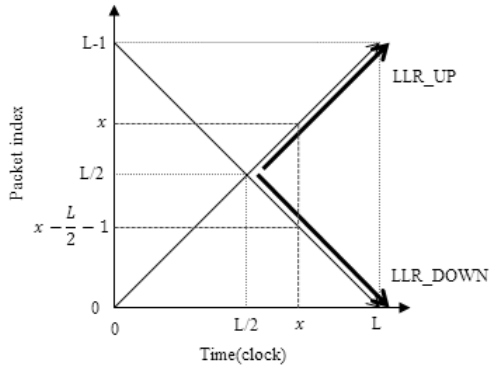

그림 8. Double flow기법과 메모리 색인
Fig. 8. Double flow method and memory index.

식 (14)와 그림 8에서 오프셋 x가 짝수일 경우를 살펴보자. x가 짝수이므로 $\Pi_{LLR_{up}}(x)$은 짝수가 된다. 그리고 $x - L/2 - 1$은 홀수 이므로 $\Pi_{LLR_{down}}(x)$은 홀수가 된다. 또한 x가 홀수인 경우에는 $\Pi_{LLR_{up}}(x)$은 홀수가 되고 $x - L/2 - 1$가 짝수 이므로 $\Pi_{LLR_{down}}(x)$은 짝수가 된다. 즉, 동시에 나오는 두 LLR 값에 대해 재배열 되는 오프셋의 짝·홀 여부가 항상 다르므로 메모리 경합이 없이 각각 다른 메모리로 사상이 가능하다. 단, 메모리에 저장을 할 때 색인을 2로 나눈 주소에 저장을 한다. 이는 각각의 메모리의 크기가 $L/2$이기 때문이다. Double flow기법의 복호에서는 그림 (8)의 $L/2$번째 클록부터 LLR_{up}, LLR_{down} 값이 LLR 프로세서로부터 출력이 된다. $L/2$번째 클록에서는 x가 $L/2$로 짝수이므로 LLR_{up}값은 짝수 메모리에, LLR_{down}값은 홀수 메모리에 사상된다. $L/2+1$번째 클록에서는 x가 홀수이므로 LLR_{up}값은 홀수 메모리에, LLR_{down}값은 짝수 메모리에 사상된다. 이와 같이 저장하는 메모리를 교차하면서 마지막에 LLR_{up}값은 홀수 메모리에 LLR_{down}값은 짝수 메모리에 사상되어 메모리 경합 없이 복호 과정을 마치게 된다.

4. 짝 · 홀 교차 사상을 적용한 인터리버 설계

가. 메모리 쓰기

K가 40일 경우에 대하여 짝·홀 교차 사상을 적용하여 메모리 쓰기를 수행하는 과정을 살펴보자. $L/2$클록에서부터 LLR 계산이 이루어지므로, K가 40인 경우에는 부분 복호 시작 후 20번째 클록에서부터 LLR 계산이 시작된다. 타이밍에 따른 인터리빙 전, 후의 오프셋 값과 사상될 메모리의 주소는 표 1과 같다.

앞의 3절 나 항에서 설명 한 바와 같이 매 클록마다 LLR_{up}과 LLR_{down}을 이들 각각의 오프셋이 짝수이면 짝수 메모리에 홀수이면 홀수 메모리에 저장한다. 그림 9에 이렇게 저장된 메모리 상태를 보인다. 사상되는 메모리 주소는 인터리빙 된 오프셋을 2로 나눈 값이다. 짝수 오프셋은 0, 2, 4, ..., 38 등 20 개 이고 홀수 주소는 사용하지 않으므로 크기가 20인 메모리에 사상이 가능하다. 홀수 오프셋 또한 마찬가지다. 그림 9의 (a)는 짝수메모리, (b)는 홀수 메모리의 저장 상태이다.

표 1. 짝·홀 교차 사상의 메모리 쓰기($K=40$)
Table1. Memory writing of Even-Odd cross mapping.

LLR 종류	LLR_{down}					LLR_{up}				
인터리빙 전 오프셋 (x)	0	…	17	18	19	20	21	22	…	39
인터리빙 후 오프셋	0	…	21	14	27	20	33	26	…	7
메모리 사상 주소	0		10	7	13	10	16	13		3
쓰기 타이밍 (~번째 클록)	39	…	22	21	20	20	21	22	…	39

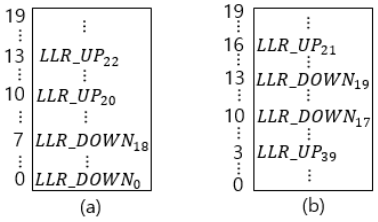

그림 9. LLR 쓰기 후 LLR 메모리 내용($K=40$)
Fig. 9. LLR memory contents after writing($K=40$).

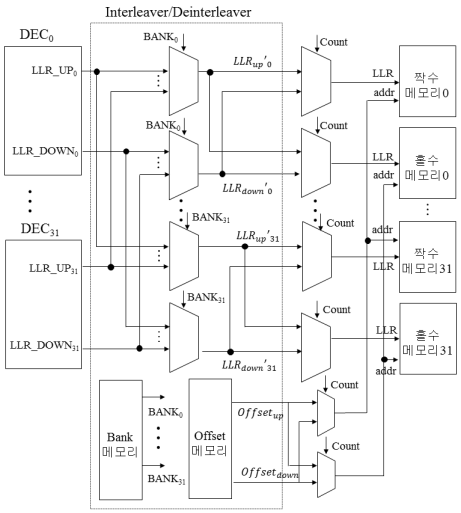

그림 10. 메모리 쓰기 제어 설계
Fig. 10. Memory writing control design.

메모리 쓰기를 위한 제어 회로의 블록도를 그림 10에 보인다. 패킷은 최대 32 개로 나누어 병렬 복호한다. 각 복호 모듈이 생성한 최대 32 쌍의 LLR 값들은 그림 내 좌측 32 개의 32:1 멀티플렉서를 통하여 어떤 메모리 뱅크로 보낼지 결정된다. 우측 64 개의 2:1 멀티플렉서는 뱅크 메모리에 도착한 LLR 값들을 어떤 메모리 블록에 저장될지 결정한다. 이 멀티플렉서의 control 신호는 Count인데, 이 값은 $L/2$클록일 때 0이며 한 클록마다 반전된다.

저장할 메모리 색인 또한 짝·홀 교차 사상에 맞추어 메모리에 제공되어야 한다. 인터리빙/디인터리빙 될 메모리 색인은 주소 발생기에 의해 복호 시작 전 미리 생성되어 오프셋 메모리에 저장되고, 이들은 매 클록마다 읽혀 각 메모리에 입력된다. 뱅크 메모리의 내용 역시 주소 발생기에 의해 사전에 생성된다.

나. 메모리 읽기

메모리 쓰기와 마찬가지로 K가 40일 경우에 대하여 이어지는 복호 과정을 위한 LLR 메모리 읽기를 수행하는 과정을 살펴보자. 저장된 LLR 값들은 FSM 계산과 RSM 계산을 위해 각각 오프셋 0부터 39까지 증가하는 방향, 오프셋 39부터 0까지 감소하는 방향으로 매 클록마다 동시에 두 개를 읽어 복호 모듈에 전달해야 한다.

짝수 메모리와 홀수 메모리에는 그림 11과 같이 LLR 값들이 저장되어있다(이 값들은 그림 9에 보인 메모리 내용을 re-indexing한 것이다). 따라서 FSM을 위해 순방향, RSM을 위해 역방향으로 LLR 값을 읽기 위해서는 메모리를 교차하면서 값을 읽어야 한다. 즉, FSM 계산을 위해서는 짝수 메모리(그림 11(a))의 오프셋 i, 홀수 메모리(그림 11(b))의 오프셋 i, $i=0,1,...,19$의 순서로 번갈아 LLR 값을 읽어온다. 마찬가지로 RSM 계산을 위해서 홀수 메모리의 오프셋 i, 짝수 메모리의 오프셋 i, $i=0,1,...,19$의 순서로 읽는다.

표 2. 복호 중 읽어야 할 LLR 메모리 오프셋
Table2. Memory offsets for LLR reading.

클록	1	2	3	4	...	L	$L-1$
짝수 메모리	$\underline{0}$	$\frac{L}{2}-1$	$\underline{1}$	$\frac{L}{2}-2$	...	$\frac{L}{2}-1$	0
홀수 메모리	$\frac{L}{2}-1$	$\underline{0}$	$\frac{L}{2}-2$	$\underline{1}$	...	0	$\frac{L}{2}-1$

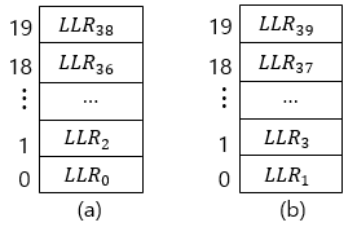

그림 11. 복호에 필요한 LLR 메모리 내용($K=40$)
Fig. 11. Contents of LLR memory for decoding.

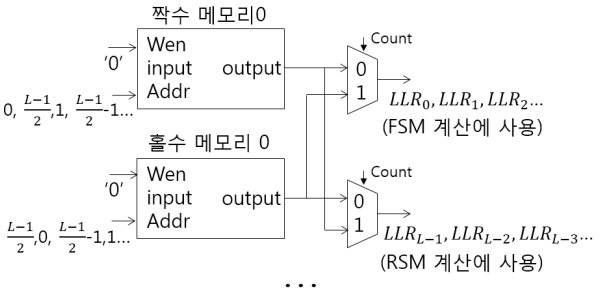

그림 12. 메모리 읽기 위한 제어 회로
Fig. 12. Control circuit for memory reading.

이러한 순서로 메모리를 읽기 위한 제어 회로를 그림 12에 보인다. 쓰기 가능 신호는 '0'으로 오프(off)시키고, 각각의 메모리에 적절한 주소 값(addr)을 입력하여 원

하는 LLR을 가져온다. 짝수 메모리와 홀수 메모리에 제공해야 할 오프셋 값을 표 2에 보인다. 표에서 밑줄 친 오프셋은 FSM 계산을 위한 것이고, 나머지는 RSM 계산을 위한 메모리 오프셋이다.

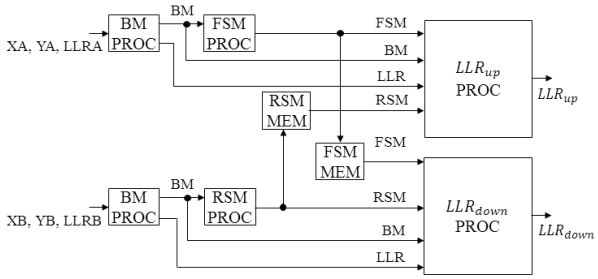

그림 13. Double flow 기법의 MAP 복호 모듈
Fig. 13. Double flow based MAP decoding module.

5. MAP 복호 모듈 설계

Double flow 기법을 적용한 복호 모듈 구조를 그림 13에 보인다. MAP 복호 모듈 외부에서 한 쌍의 정보 비트(XA, XB), 패리티 비트(YA, YB), 사전 정보 값(LLRA, LLRB)을 입력받아 두 개의 BM 프로세서로 가지 값을 계산하고 이들을 FSM 프로세서와 RSM 프로세서, 그리고 LLR 프로세서들로 보낸다. 그리고 LLRA는 LLRup프로세서로 보내져 다음 LLR 값인 LLR_{up} 계산에 사용되고, LLRB는 LLRdown 프로세서에서 LLR_{down} 계산에 사용된다.

FSM 프로세서의 출력은 처음 $L/2$ 클록동안 FSM 메모리에 저장되는데 이들은 $L/2$ 클록 후에 RSM 프로세서의 출력과 함께 LLR_{down} 값들의 계산에 사용된다. 마찬가지로 RSM 프로세서의 출력은 처음 $L/2$ 클록동안 RSM 메모리에 저장되고, 이들은 $L/2$ 클록 후에 FSM 프로세서의 출력과 함께 LLR_{up} 값들의 계산에 사용된다. FSM 메모리와 RSM 메모리는 모두 싱글 포트 메모리로 가능하며 그 크기는 각각 $L/2$이다.

Ⅳ. 구현 및 실험 결과

1. 복호기 전체의 구조와 동작

그림 14에 구현한 병렬 터보 복호기의 전체 구성을 보인다. 복호기는 총 32개의 병렬 프로세서로 구성되어 있으며, 입력 블록(IB), MAP 복호 블록(MDB), 인터리브/디인터리브 주소 생성 블록(IDAGB), 인터리브/디인터리브 블록(IDB), 짝·홀 교차 사상 블록(EOCMB), 출력 블록(OB) 등의 여섯 부분으로 구성되어 있다. 본 논문에서 사용한 K, P, L 값과 QPP 인터리버의 f_1, f_2 값은 참고문헌 [8]을 참고하여 결정하였다.

패킷이 입력되면 IB에 정보 비트와 패리티 비트가 저장됨과 동시에 IDAGB에서 주소를 생성하여 IDB의 매퍼 메모리에 저장한다. 복호는 패킷 입력을 마치면 시작된다. 입력 블록에서 정해진 순서대로 MDB에 인터리브를 위한 정보 비트와 패리티 비트, 꼬리 비트를 전달하면 MDB는 복호를 수행하여 IDB로 LLR 값을 전달한다. IDB는 인터리브를 수행하여 EOCMB로 재배열된 LLR 값을 보내고, EOCMB에서 경합 없이 메모리에 사상시켜 다음 반복을 위한 LLR 값을 순서에 맞게 IB로 보내준다.

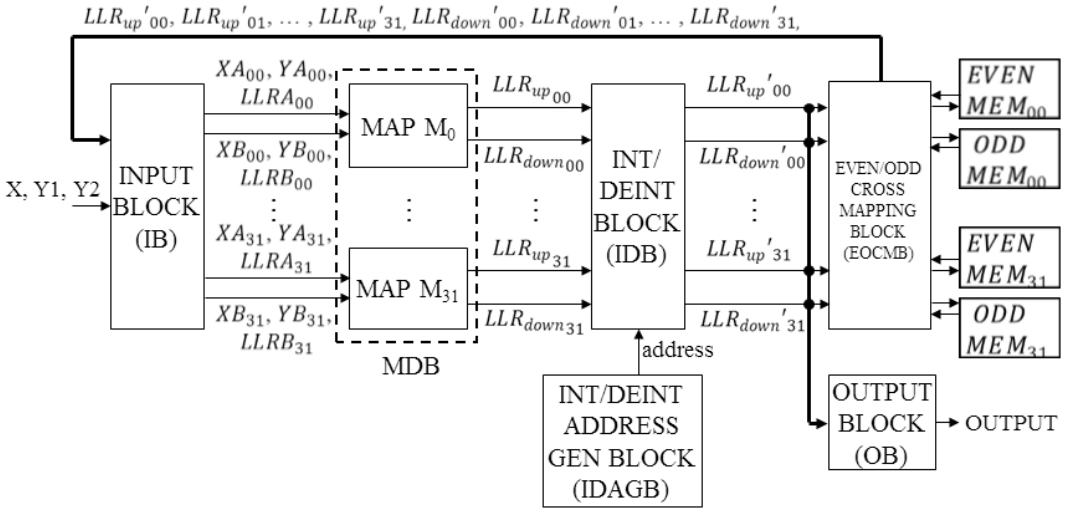

그림 14. 구현한 병렬 터보 복호기의 구성도
Fig. 14. Block diagram of our parallel turbo decoder.

그리고 다시 IB는 디인터리브를 위한 정보 비트와 패리티 비트, 꼬리 비트를 MDB에 전달한다. MDB에서는 앞과 같은 과정을 수행하고 IDB에서는 디인터리브를 수행하여 IB의 LLR 메모리에 값을 저장한다.

이러한 과정을 반복적으로 수행하며 주어진 반복 횟수만큼 복호를 마치면 IDB는 OB로 LLR 값을 전달한다. OB에서는 LLR 값을 경판정하여 출력 메모리에 저장하고 저장이 끝나면 복호 결과를 출력한다.

2. 성능 분석

구현한 복호기의 성능을 조사하고 이의 정확한 동작을 확인하기 위하여 먼저 C언어를 사용하여 시뮬레이터를 작성하였다. C 시뮬레이터는 실수(floating point) 연산과 복호기 구현에 사용한 유한 비트의 fixed point 연산을 선택적으로 사용하여 복호를 수행할 수 있다.

구현한 복호기는 double flow 기법을 적용하되 메모리 경합을 피하기 위하여 본 논문에서 제시한 짝·홀 교차 사상 기법을 인터리버 설계에 적용한 것이다. 한편, 기존의 MDF 방법은 LLR 계산 순서를 double flow의 그것과 다르게 하여 경합을 피한 것으로, 우리의 복호기에 비해 복호 시간만 늘어난다. 따라서 두 복호기의 복호 기능은 동일하며, BER(bit error rate) 역시 동일하다.

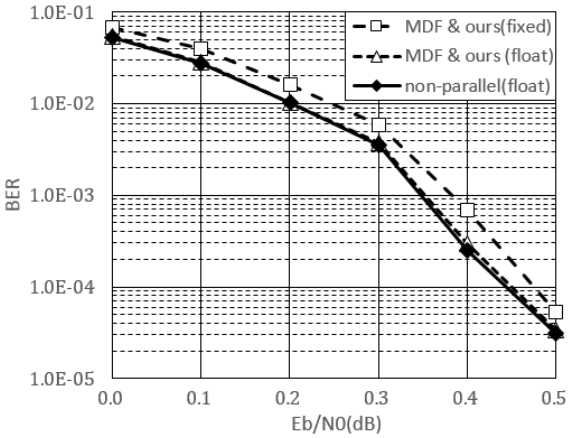

그림 15. BER 성능 비교($K=4096$).
Fig. 15. BER performance comparison($K=4096$).

그림 15에 구현한 복호기의 BER 성능을 보인다. 그림에서 마름모 표식의 실선은 비병렬 실수 연산으로 구한 BER로 하드웨어 복호기의 성능을 가늠하는데 기준이 된다. 즉, 동일한 부호기와 인터리빙 방식 그리고 반복 복호 횟수를 사용할 경우 이보다 우수한 성능의 복호기 구현은 대단히 어렵다.

우리의 복호기를 실수 연산으로 시뮬레이션 하였을 경우(삼각형 표식의 점선) 그 성능은 비병렬 실수 연산에 의한 복호 성능과 거의 유사하여 병렬 처리에 문제가 없음을 알 수 있다. 한편, 실제 구현한 복호기는 실수 연산에 비해 약 0.05 dB 이하의 성능 저하를 보이는데(네모 표식의 점선), 이는 fixed point 연산으로 인한 피할 수 없는 작은 성능 저하로 간주된다.

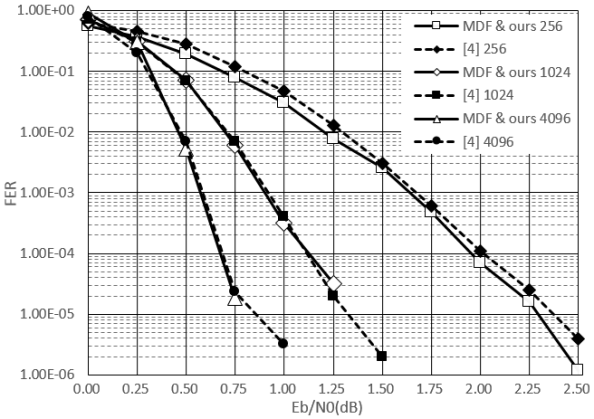

그림 16. 참고문헌 [4] 결과와의 FER 성능 비교
Fig. 16. FER performance comparison with ref[4].

구현한 복호기의 성능을 확인하기 위하여 참고문헌 [4]의 결과와 비교한 결과를 그림 16에 보인다[4~5]. 참고문헌 [4]에서는 BER 대신 FER(frame error rate)를 사용하였기 때문에 그림 16은 FER 비교 결과를 보인 것이다. 그림에서 보인 것과 같이 여러 크기의 패킷 길이에 대하여 구현한 복호기는 참고 문헌 [4]의 결과에 비하여 유사하거나 더 좋은 FER 결과를 얻는다.

다음, 복호에 필요한 클록 수를 확인하였다. 복호기는 VHDL로 구현하여 Modelsim을 통해 시뮬레이션을 수행하였다. 표 3에 여러 크기의 패킷에 대해 MDF기법과 double flow 기법으로 각각 복호하는데 필요한 클록 수를 보인다. 반복 횟수는 안정적인 BER 값을 얻는 8회로 고정하였다. MDF 기법에 비하여 약 32%의 복호 시간 감소율을 보이는데 이론적 복호 시간 감소율인 33%에 미치지 못한다. 이는 작동 주파수 향상을 위한 파이프 라인 삽입으로 고정적인 지연시간이 생겼기 때문이다. 패킷 길이가 40일 때 약 28%의 가장 작은 감소율을 보이는데, 이는 패킷 크기가 작을 경우 실제 복호시간 대비 고정적인 지연시간이 상대적으로 크기 때문이다.

3. 합성 결과

VHDL로 작성된 복호기를 SMIC-0.18um 기술의

표 3. MDF 기법과 Double flow 기법의 복호 시간 비교
Table3. Decoding time comparison with MDF method and double flow method.

패킷 길이		40	256	264	512	528	1024	1056	2048	2112	4096	4160	6144
프로세서 개수		1	1	2	2	4	4	8	8	16	16	32	32
부분 패킷 길이		40	256	132	256	132	256	132	256	132	256	130	192
총 복호 시간 (클록 수)	MDF 기법	1120	6304	3328	6304	3328	6304	3328	6304	3328	6304	3280	4768
	Double flow 기법	800	4256	2272	4256	2272	4256	2272	4256	2272	4256	2240	3232
	감소율 (%)	28.6	32.5	31.7	32.5	31.7	32.5	31.7	32.5	31.7	32.5	31.7	32.2

표 4. Standard cell 합성 결과
Table4. Results of standard cell synthesis.

사용 회로	MDF 기법	짝·홀 교차 사상을 이용한 double flow 기법(ours)	듀얼 포트 메모리를 이용한 double flow 기법
	gate count	gate count (MDF 대비 8.4% 증가)	gate count (MDF 대비 21.4% 증가)
Logic Area	968,959	1,361,266	1,171,337
Macro Area(Memory)	3,248,563	3,212,953	3,952,653
Total area	4,217,521	4,574,219	5,123,990

Arm-Artisan 표준셀 라이브러리로 합성을 수행하였다. 사용한 도구는 Synopsys의 Design Compiler이다. MDF 기법, 듀얼 포트 메모리를 사용한 double flow 기법, 본 논문에서 제시한 짝·홀 교차 사상을 적용한 double flow 기법 등으로 설계한 복호기를 모두 합성하였다. 공정한 비교를 위하여 파이프라인 단수는 모두 같게 맞추었다.

표 4에 합성 결과를 보인다. 표에서 보인 것과 같이 병렬 복호기의 경우 메모리 면적이 논리회로 면적보다 두 배 이상 크다. 이는 입력 메모리, LLR 메모리, 각 복호 모듈에서의 RSM 또는 FSM 저장용 메모리 등 많은 부분에 메모리 블록이 필요하기 때문이다.

우리의 복호기를 살펴보면 논리회로의 면적이 MDF 기법에 의한 면적보다 매우 큼을 알 수 있다. 이는 MDF에서는 각 복호 모듈에서 LLR 모듈이 하나만 필요하지만, double flow의 경우 두 개의 LLR 모듈이 필요하고, 추가로, 짝·홀 교차 사상에 의한 인터리버 구현에 논리회로가 더 필요하기 때문이다.

한편, MDF 기법에 사용한 메모리 비트수는 double flow를 사용한 복호기에 비하여 75%에 불과하다(표 5). 이는 MDF기법에서는 RSM을 저장할 메모리만 필요한데 비해, double flow기법에서는 FSM을 저장할 메모리가 추가로 필요하기 때문이다. 그러나 MDF 기법에 필요한 RSM 메모리는 듀얼 포트 메모리이어야 하는데 반하여, double flow 기법에서는 FSM, RSM 모두 싱글 포트 메모리를 사용할 수 있어 표 4에 보인 것처럼 전체 메모리 면적은 MDF 기법과 비교할 때 거의 유사하다. 결론적으로 본 논문에서 제안한 복호기의 전체 면적은 MDF에 의한 복호기의 면적에 비하여 약 8% 증가하였다.

LLR 메모리로 듀얼 포트 메모리를 사용하여 double flow 기법을 구현한 경우, 복호 모듈에 추가로 장착되는 LLR 모듈로 인한 논리회로 뿐만 아니라, 약 590K bits의 듀얼 포트 메모리 사용으로(표 5) 인하여 MDF에 비해 전체 면적이 약 21% 증가한다(표 4). 그러나 본 논문에서 구현한 복호기의 경우, MDF에 비해 약 8% 정도의 추가 면적을 사용하여 약 32%의 복호 속도 개선을 이룰 수 있어 고속 복호기로서의 가치가 더 크다고 볼 수 있다.

마지막으로 각 설계의 동작 가능 클록 주파수는 모두 100MHz 또는 그 이상으로 예측되었다.

Ⅳ. 결 론

터보 코드의 빠른 복호를 위한 double flow 기법은 메모리 경합 가능성이 있는데, 이를 피하기 위하여 기존에 제안된 MDF 기법의 복호 시간은 최적이 아니다. 본 논문에서는 이러한 문제의 해결 방안으로 QPP 인터리버의 주소 출력 상의 특징을 이용한 짝·홀 교차 사상을 제안하였다. 그리고 double flow와 제안한 기법을 적용하여 구현한 병렬 터보 복호기의 동작과 구조를 아울

러 설명하였다. 짝·홀 교차 사상은 메모리 경합 없이 double flow기법을 싱글 포트 메모리만으로 구현할 수 있게 하여 MDF 대비 약 8%의 면적 증가로 복호 시간을 약 32% 단축할 수 있게 하여준다. 또한, MDF와 동일한 BER 성능을 가지기 때문에 본 논문에서 기술한 복호 방법은 고속 터보 복호가 필요한 응용에 유용하게 사용할 수 있다.

감사의 글

바쁘신 업무에도 불구하고 표준 셀 합성을 기꺼이 도와주신 ㈜파인스의 김원영 사장님외 연구원 여러분께 심심한 감사를 드립니다.

REFERENCES

[1] C. E. Shannon, “A mathematical theory of communication,” Bell System Technical Journal, vol. 27, pp. 379-423, Jul. 1948.
[2] Jae-Ming Hsu and Chin-Liang Wang, “A Parallel Decoding Scheme for Turbo Codes,” IEEE Inl. Symp. on Circuits and Systenrs, vol. 4, pp. 445-448, June 1998.
[3] C. Schurgers, F. Catthoor, M. Engels, “Optimized MAP Turbo Decoder”, SiPS 2000, Oct. 2000, pp. 245-254.
[4] O. Y. Takeshita, “On Maximum Contention-Free Interleavers and Permutation Polynomials over Integer Rings”, IEEE Transactions on Information Theory, Vol. 52, No. 3, Mar. 2005, pp. 1249-1253.
[5] Jae-Hun Chung, Heemin Park, Chong S., “Design of a Contention-Free Parallel Double-Flow MAP Decoder”, IEEJ, Volume 10, 2015, pp.70-74.
[6] “Technical Specification Group Radio Access Network; Multiplexing and channel coding (FDD)(Release 6)”, 3GPP TS 25.212 V6.7.0, 2005-12.
[7] S. S. Pietrobon, A. S. Barbulescu, “A Simplification of the Modified Bahl Decoding Algorithm for Systematic Convolutional Codes”, International Symposium on Information Theory and its Applications, pp. 1073-1077, Sydney, Australia, Nov. 1994.
[8] Ericsson, “Quadratic Permutation Polynomial Interleaver Designs for LTE Turbo Coding”, 3GPP TSG-RAN WG1 #47 bis, R1-070462.
[9] Z. He, P. Fortier, S. Roy, “Highly-Parallel Decoding Architectures for Convolutional Turbo Codes”, IEEE Transactions on Very Large Scale Integration(VLSI) Systems, Vol. 14, No. 10, pp. 1147-1151, Oct. 2006.
[10] H. Bajwa, An area-efficient, high-performance, low-power multi-port cache memory architecture, Ph.D. Thesis, Department of Electrical Engineering, City University of New York., pp. 7-16, 2007.
[11] J.-H. Chung and C. S. Rim, Design of Contention Free Parallel MAP Decode Module, J. of IEEK, Vol. 48(SD), No. 1, pp. 39~49, 2011.

저 자 소 개

좌 유 철(학생회원)
2015년 서강대학교 컴퓨터공학과 학사 졸업.
2017년 서강대학교 컴퓨터공학과 석사 졸업.
<주관심분야: ASIC 설계, SoC 설계, 부호 이론, 암호학>

임 종 석(평생회원)
대한전자공학회 논문지
제45권 SD편 제12호 참조