논문 2011-48SD-1-6

메모리 경합이 없는 병렬 MAP 복호 모듈 설계

(Design of Contention Free Parallel MAP Decode Module)

정 재 헌*, 임 종 석**

(Jae-Hun Chung and Chong-Suck Rim)

요 약

터보 코드는 반복 복호를 하기 때문에 긴 복호 시간을 필요로 한다. 고속 통신을 하기 위해서는 복호 시간을 줄여야 하며 이는 병렬 처리를 통해 해결할 수 있다. 하지만 병렬 처리 시 메모리 경합이 발생할 수 있는데 이는 복호기의 성능을 저하시킨다. 이러한 메모리 경합을 피하기 위해 2006년 QPP 인터리버가 제안되었다. 본 논문에서는 QPP 인터리버에 적합하며 비교적 적은 지연 시간을 갖고 회로의 크기도 줄인 MDF 기법을 제안한다. 그리고 MDF 기법을 사용한 MAP 복호 모듈의 설계를 보인다. 구현한 복호기는 Xilinx 사의 FPGA에 타겟팅하였으며 최대 80Mbps의 처리율을 보인다.

Abstract

Turbo code needs long decoding time because of iterative decoding. To communicate with high speed, we have to shorten decoding time and it is possible with parallel process. But memory contention can cause from parallel process, and it reduces performance of decoder. To avoid memory contention, QPP interleaver is proposed in 2006. In this paper, we propose MDF method which is fit to QPP interleaver, and has relatively short decoding time and reduced logic. And introduce the design of MAP decode module using MDF method. Designed decoder is targetted to FPGA of Xilinx, and its throughput is 80Mbps maximum.

Keywords: MAP, Turbo decoder, QPP interleaver, Parallel process, Contention free

Ⅰ. 서 론

터보 코드[1]는 섀넌의 제한에 근접한 복호 방식으로 ITU(international telecommunications union)에서 IMT-2000의 동기식 및 비동기식 통신 시스템의 고속 데이터 전송용의 채널 코드로 채택된 상태이다[2~3].

터보 코드는 복호 방법으로 반복 복호(iterative decoding) 방법을 이용하기 때문에 좋은 성능을 얻기 위해서는 긴 복호 시간이 요구된다. 그리고 터보 코드 복호 알고리즘으로 주로 이용되는 MAP(maximum a posteriori) 알고리즘은 전방 상태 값(FSM : forward state metric)과 후방 상태 값(RSM : reverse state metric)을 이용하여 양방향으로 복호를 진행하며 전통적인 처리 기법[4]으로 구현할 경우 입력된 패킷(packet)에 대해 그 두 배의 길이만큼의 복호 시간을 필요로 한다. 이러한 복호 시간을 줄이기 위하여 병렬 처리 기법에 대한 연구가 진행되었다.

병렬 처리 기법은 입력된 패킷을 적절히 나누어 한 개의 복호 모듈이 나누어진 패킷의 한 부분을 복호하여 출력하고, 각 복호 모듈의 출력을 연결하여 최종 출력을 내는 방법이다. 병렬 처리를 할 경우 나누어진 패킷의 양 끝 부분에 학습 구간을 두어야 좋은 BER 성능을 유지할 수 있다. 하지만 이러한 학습 구간에 의한 복호 지연 시간이 발생하게 된다. 본 논문에서는 이러한 지연 시간을 줄이기 위하여 He가 제안한 학습 구간이 없는 병렬 구조(warm-up-free parallel architecture)[5]를 사용

* 정회원, 삼성전자 DMC연구소
(DMC R&D Center, Samsung Electronics)
** 평생회원, 서강대학교 컴퓨터공학과
(Dept. of CS&E, Sogang University)
※ 본 연구는 2009년 텔에이스(주)의 지원에 의한 결과임
접수일자: 2010년8월13일, 수정완료일: 2010년12월23일

하여 학습 구간으로 인한 복호 지연 시간을 줄였다.

병렬 처리를 할 경우 병렬 프로세서의 출력이 인터리브(interleave) 또는 디인터리브(de-interleave) 시 두 개 이상의 복호 모듈 출력이 하나의 메모리로 사상되는 메모리 경합(contention)이 발생할 수 있다. 메모리 경합이 발생하면 이를 해결하기 위하여 추가적으로 회로를 더 사용하여야 하며 또한 복호 지연 시간도 늘어나게 되어 비효율적이다. 메모리 경합을 없애기 위하여 본 논문에서는 QPP(quadratic polynomial permutation) 인터리버를 사용하였다[6]. QPP 인터리버는 터보 코드의 병렬 복호에 적합한 인터리버로 패킷의 길이에 따라 병렬 처리하는 프로세서의 개수가 다르다[7].

MAP 알고리즘의 처리 기법 중 하나인 double flow 기법[8~9]은 전방 상태 값과 후방 상태 값을 동시에 계산하기 시작하여 패킷 입력이 L일 경우 L 만큼의 복호 시간을 가진다. 하지만 double flow 기법은 LLR (log-likelihood ratio) 값 출력이 한 클록에 두 개씩 나오며, 출력 순서가 QPP 인터리버를 사용할 경우 메모리 경합이 발생하여 병렬 처리에 적합하지 않다.

따라서 본 논문에서는 double flow 기법을 수정하여 병렬 처리를 하여도 메모리 경합이 없는 MDF(modified double flow) 기법을 제안한다. MDF 기법은 1.5L의 복호 시간을 갖고 double flow 기법에 비해 회로의 크기와 메모리의 개수를 줄였다. 그리고 MDF 기법을 적용한 MAP 복호 모듈의 설계와 제어 방법을 소개한다.

본 논문에서 구현한 복호기는 QPP 인터리버를 사용하였고, MDF 기법을 적용한 MAP 복호 모듈 32개를 사용하여 병렬 처리를 하였다. 복호기는 VHDL로 구현되었고 Synplify Pro 8.5 및 Xilinx ISE 8.2i에서 합성되었다. 구현된 복호기는 xc4vlx200-11ff1513 장치에서 최대 80Mbps로 동작한다.

본 논문의 구성은 다음과 같다. 제 Ⅱ장에서 터보 코드 병렬 처리와 관련된 이론을 소개한다. Ⅲ장에서는 double flow 기법의 문제점과 이를 해결한 MDF 기법, 그리고 이의 설계를 보인다. Ⅳ장에서는 구현된 전체 복호기의 구조와 동작, 그리고 실험 결과를 보인다. 마지막으로 Ⅴ장에서는 본 논문의 결론을 내린다.

Ⅱ. 터보 코드 병렬 처리 기법

1. 터보 코드 부호기와 복호기

터보 부호기는 2개의 8-상태 구성 부호기(8-state constituent encoder)와 내부 인터리버를 갖는 병렬 연접 길쌈 부호기(parallel concatenated convolutional encoder)이다. 터보 부호기의 부호 율(code rate)은 1/3이며 그림 1에 터보 코드 부호기의 블록 다이어그램을 보인다[11]. 그림 1에서 부호기로 정보 비트 d_k가 입력되면 d_k는 그대로 출력하고, 첫 번째 구성 부호기는 입력된 d_k를 부호하여 첫 번째 패리티(parity) 비트 Y_k를 출력한다. 두 번째 구성 부호기는 d_k를 인터리버를 통해 재배열하고 이를 부호하여 두 번째 패리티 비트 Y'_k를 출력한다.

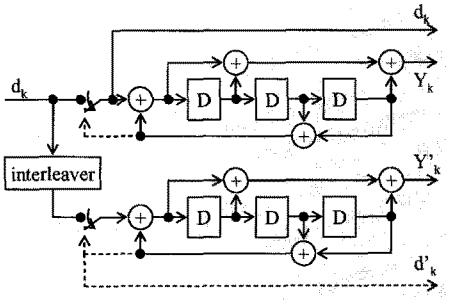


그림 1. 터보 코드 부호기[10]
Fig. 1. Turbo code encoder[10].

본 논문에서는 AWGN 채널(additive white Gaussian noise channel) 상에서 부호 율 1/3로 정보 비트와 패리티 비트가 BPSK(binary phase shift keying) 변조되어 전송되는 것을 기준으로 한다. 길이 K인 패킷은 부호기 출력 $(d_0, Y_0, Y'_0),\cdots,(d_{K-1}, Y_{K-1}, Y'_{K-1})$에 대해 이들이 갖는 0, 1 값이 -1, +1로 각각 바뀌어 전송되고 전송 도중 평균이 0이고 분산이 σ^2인 정규 분포를 갖고 서로 독립적인 잡음 p_k, q_k, r_k가 각각 더해진다고 가정하면 수신 신호 (x_k, y_k, y'_k)은 수식 (1)과 같이 정의된다.

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

수신된 신호는 양자화를 거쳐 L_c를 사전에 곱한 후 입력된다고 가정한다. 여기서, L_c는 잡음의 강도에 따

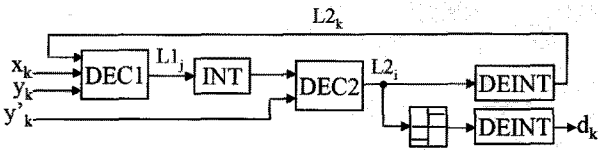


그림 2. 터보 코드 복호기[11]
Fig. 2. Turbo code decoder[11]

라 사전에 곱해주어야 할 값으로 d_k, Y_k, Y'_k이 각각 -1 또는 +1로 바꾸어 설정하였을 경우 $\frac{2}{\sigma^2}$를 갖는다.

터보 코드 복호기는 기본적으로 그림 2와 같은 구조를 가지고 있다[12]. x_k와 y_k, 그리고 사전 정보 값(a priori information) $L2_k$를 첫 번째 복호 모듈로 입력되어 첫 번째 복호를 수행한다. 사전 정보 값의 초기 값은 0으로 한다. 그리고 첫 번째 복호 모듈의 출력 $L1_j$를 인터리버의 입력으로 하여 인터리브를 수행한다.

인터리버의 출력과 y'_k을 두 번째 복호 모듈로 입력하여 두 번째 복호를 수행한다. 두 번째 복호 모듈의 출력 $L2_i$는 디인터리버(de-interleaver)로 입력되어 디인터리브(de-interleave)된 후 다음 반복 복호를 위한 사전 정보 값으로 다시 첫 번째 복호 모듈의 입력으로 들어간다.

이의 과정을 반복적으로 수행하다 주어진 반복 횟수만큼 복호를 하면 두 번째 복호 모듈의 출력 $L2_i$를 경판정(hard decision)한 후 디인터리버를 거친 후 출력된다.

2. Log-MAP 알고리즘

복호 모듈에서 사용하는 log-MAP 알고리즘의 배경이론 및 자세한 유도 과정은 참고문헌 [12]에 기술되어 있으므로 이에 대한 설명을 생략하고 터보 코드 복호를 위한 결론적인 수식만을 기술하기로 한다.

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 값은 초기 계산일 경우 0이고, 반복 계산일 경우 이전 단계에서의 LLR 값이다.

계산한 b_k 값을 사용하여 FSM f_k와 RSM r_k는 그림 3의 8-상태 구성 부호기의 트렐리스 다이어그램(Trellis diagram)에 근거하여 반복 계산할 수 있다. FSM, RSM은 8개의 상태를 가지므로 각각 8개의 값을 가진다.

f_k들은 초기 값으로 수식 (4)와 같이 설정한다.

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

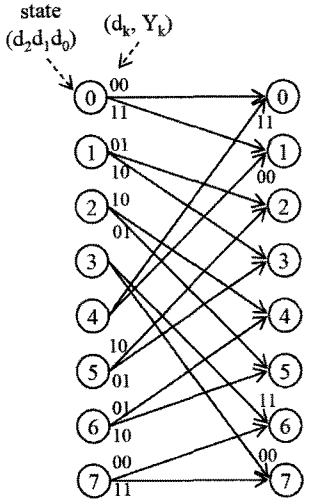


그림 3. 부호기의 트렐리스 다이어그램
Fig. 3. Trellis diagram of encoder.

그리고 함수 $E(x,y)$를 수식 (3)과 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.

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

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

이와 유사하게 r_k들은 초기 값으로 수식 (6)과 같이 설정한다.

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

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

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

$S_f^i(m)$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.

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

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

$$L^i(d_k) = E_{m=0}^{7}(f_k(m) + r_k(S_f^i(m)) + b_k(i, Y_k(m))) \quad (9)$$

그러면 수식 (10)에 의하여 d_k 값을 예측할 수 있다.

$$L(d_k) = L^1(d_k) - L^0(d_k) \quad (10)$$

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[12].

3. 병렬 처리 기법

터보 코드는 반복 복호를 수행하고, 복호 시간이 입력된 패킷의 길이에 비례하기 때문에 긴 복호 시간을 가진다. 하지만 입력된 패킷을 적당한 크기로 나누어 여러 개의 MAP 복호 모듈이 동시에 복호를 진행한다면 처리율은 향상될 것이다.

일반적으로 log-MAP 복호를 수행하기 위해서는 패킷의 시작과 끝 상태 값을 알고 있어야 한다. 하지만 병렬 처리를 위해 패킷을 나눌 경우 전체 패킷의 시작과 끝을 제외한 나누어진 패킷의 시작과 끝 상태 값은 정의가 되어 있지 않다. 하지만 정의되어 있지 않은 패킷의 시작과 끝 상태 값을 임의의 값으로 놓고 복호를 하게 되면 BER 성능은 떨어지게 된다. 이러한 BER 성능 저하를 막기 위하여 나누어진 패킷의 시작과 끝 상태 값을 정의하기 위하여 학습 구간을 둔다.

하지만 학습 구간은 반복 복호를 할 경우 긴 지연 시간을 발생시킨다. 이를 개선시키기 위하여 He가 제안한 학습 구간이 없는 병렬 구조[5]를 사용하였다. 학습 구간이 없는 병렬 구조는 학습 구간이 없는 대신 나누어진 패킷의 시작과 끝 상태 값을 이전 반복에서 저장한 상태 값으로 사용하는 방법이다. 첫 반복에서는 임의의 상태 값으로 나누어진 패킷의 시작과 끝 상태 값을 정의한다. 학습 구간이 없는 병렬 구조를 사용할 경우 반복 횟수가 5회 이상이 되면 학습 구간을 둔 병렬 처리 기법과 성능 차이가 거의 없다[5].

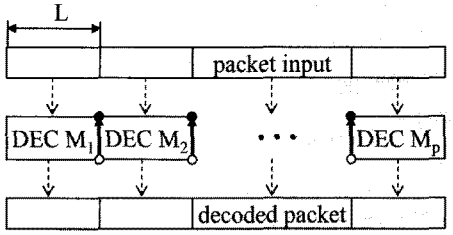


그림 4. 학습 구간이 없는 병렬 구조
Fig. 4. Warm-up free parallel architecture.

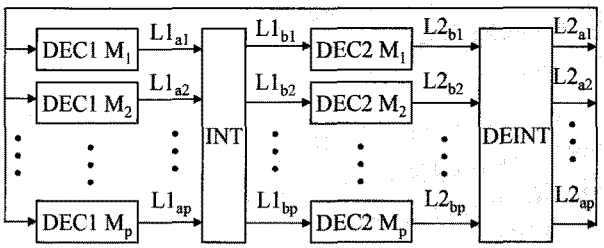


그림 5. 병렬 처리 구조
Fig. 5. Parallel process architecture.

그림 4에 학습 구간이 없는 병렬 처리 기법을 보인다. 그림 4에서 각 복호 모듈의 위와 아래에 있는 검은 원은 상태 값의 읽음을, 하얀 원은 상태 값의 저장을 의미하며, 굵은 화살표는 이전 반복에서 저장된 상태 값을 읽음을 의미한다. 각 복호 모듈의 시작과 끝의 상태 값은 복호가 끝날 때 저장된 후 다음 복호의 시작 때 읽혀진다. 학습 구간이 없는 병렬 처리는 하나의 복호 모듈이 L 만큼을 입력으로 하여 복호를 수행하고 그 출력을 이어 전체 패킷의 복호결과가 된다.

그림 5에 병렬 복호 처리를 위한 프로세서 구조를 보인다. 그림 5에서는 사전 정보 값이 흘러가는 데이터 경로(data path)만을 보인다. 각각의 복호 모듈은 나누어진 패킷 L 만큼을 복호한다. 이 때 각각의 복호 모듈을 윈도우(window)라고 정의하고, 나누어진 패킷 길이 L을 윈도우 길이라 정의한다. 그리고 각 윈도우가 처리하는 패킷 색인을 새롭게 0부터 순서대로 시작하여 오프셋(offset)이라 정의하며 오프셋은 패킷 색인을 윈도우 길이로 나눈 나머지로 구할 수 있다.

첫 번째 복호 모듈들에서 나온 사전 정보 값 $(L1_{a1}, L1_{a2}, \cdots, L1_{ap})$은 인터리버를 거쳐 재배열되어 $(L1_{b1}, L1_{b2}, \cdots, L1_{bp})$가 된다. 사전 정보 값의 각 아래 첨자들은 오프셋이 아닌 패킷 색인을 의미한다. 인터리버를 거쳐 재배열된 사전 정보 값을 입력으로 하여 두

번째 복호를 하고, 두 번째 복호 모듈들에서 나온 사전 정보 값 $(L2_{b1}, L2_{b2}, \cdots, L2_{bp})$는 디인터리버를 거쳐 $(L2_{a1}, L2_{a2}, \cdots, L2_{ap})$로 재배열된 후 반복 복호를 위하여 첫 번째 복호 모듈로 보내어진다.

4. 메모리 경합과 QPP 인터리버

원활한 병렬 처리를 하기 위해서는 인터리브와 디인터리브를 할 경우 메모리 경합이 없어야 한다. 메모리 경합이란 그림 6의 (a)와 같이 인터리브 시 두 개의 복호 모듈(첫 번째 복호 모듈과 세 번째 복호 모듈)의 출력이 인터리브 후 한 개의 복호 모듈(두 번째 복호 모듈)로 사상되는 경우이다. 인터리브 결과는 메모리에 저장된 후 다음 복호 모듈로 결과를 출력하는데, 메모리는 한 클록에 한 개의 주소에 한 개의 데이터만 쓸 수 있기 때문에 이를 처리할 수 없으므로 메모리 경합이라 부른다.

메모리 경합이 발생하면 이러한 충돌을 피하기 위해서 버퍼(buffer)를 두고 경합이 일어날 때마다 버퍼에 저장한 후 FIFO(first in first out) 방식으로 사상을 하여 버퍼를 모두 비우면 인터리브가 끝나게 된다. 하지만 버퍼는 레지스터의 집합이므로 회로가 커질 뿐만 아니라 버퍼를 통과하는 지연 시간이 발생하여 복호 시간도 늘어난다. 따라서 원활한 병렬 처리를 위해서는 그림 6의 (b)와 같이 메모리 경합이 없는 인터리버를 사용하여야 한다.

본 논문에서 사용한 QPP 인터리버는 2006년 Takeshita에 의해 처음 제안되었으며 메모리 경합이 없고 3GPP(3rd generation partnership project) 표준에 잘 부합되는 인터리버이다[6].

길이가 K인 패킷이 주어졌을 때, x번째 주소에 대한 인터리브된 주소는 수식 (11)과 같이 정의된다.

$$\Pi(x) = f_1 x + f_2 x^2 \bmod K \quad where\ 0 \le x, f_1, f_2 < K \tag{11}$$

수식 (11)에서 사용하는 계수 f_1, f_2를 정하는 방법은 참고문헌 [6]에 자세하게 설명되어 있으므로 본 논문에서는 생략하기로 한다. QPP 인터리버는 윈도우 크기 L, 패킷의 길이 K에 대해 P=K/L개의 프로세서를 사용할 경우 메모리 경합이 발생하지 않는다.

5. 전통적인 처리 기법과 double flow 기법

가. 전통적인 처리 기법

전통적인 처리 기법[4]은 먼저 FSM을 계산하고, 그 값을 메모리에 저장한 후 RSM을 계산함과 동시에 메모리에 저장된 FSM을 읽어 LLR 값을 계산하는 방법이다. 하나의 프로세서가 길이 L의 패킷을 처리할 경우 FSM 계산에 L 클록, 그리고 RSM 계산과 LLR 값 계산에 L 클록의 시간이 걸려 총 2L 클록의 복호 시간이 소요된다.

그림 7에 전통적인 처리 기법을 그래프 형태로 보인다. 그래프에서 X 축은 클록 단위로 나타내는 시간을 의미하고, Y 축은 처리할 패킷의 색인(index)을 나타낸다. 오른쪽 위를 향하는 화살표는 FSM 계산을 나타내고, 오른쪽 아래를 향하는 화살표는 RSM계산을 나타낸다. 굵은 화살표는 LLR 값 계산을 나타낸다.

전통적인 처리 기법은 먼저 L 클록 동안 패킷 색인 0부터 L-1까지 증가하는 순서대로 FSM을 계산하며 그 결과를 FSM 메모리에 저장한다. 그리고 L+1 클록부터 2L 클록까지 패킷 색인 L-1부터 0까지 감소하는 순서대로 RSM을 계산한다. 동시에 계산된 RSM과 메모리에 저장된 FSM을 읽어 LLR 값을 계산한다.

전통적인 처리 기법은 한 개의 FSM 프로세서와 한 개의 RSM 프로세서, 한 개의 LLR 프로세서를 필요로

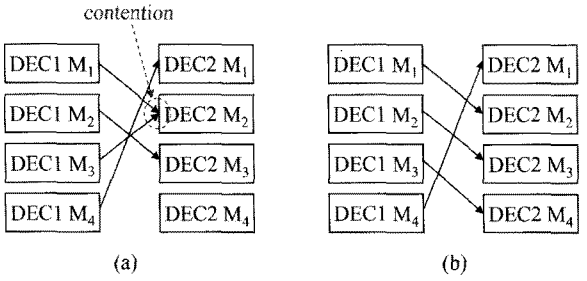


그림 6. 메모리 경합
Fig. 6. Memory contention.

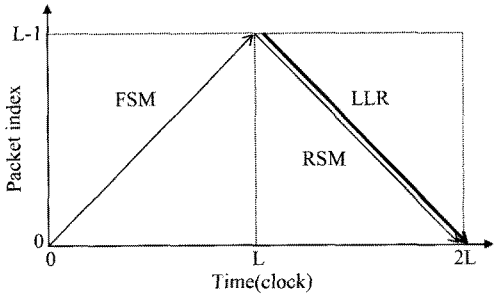


그림 7. 전통적인 처리 기법
Fig. 7. Conventional method.

한다. 그리고 깊이 L의 FSM 메모리 한 개를 필요로 한다.

나. Double flow 기법

Double flow 기법[7~8]은 FSM과 RSM을 동시에 계산하는 방법이다. FSM과 RSM 계산을 동시에 시작하며 L/2 클록 후에 LLR 값이 한 클록에 두 개씩 나온다. 총 L 클록의 복호 시간이 소요된다.

그림 8에 double flow 기법을 그래프 형태로 보인다. 그림 8과 같은 동작을 하기 위해서는 패킷 색인 0부터 L/2부터 L-1까지의 RSM을 저장하는 메모리가 필요하다.

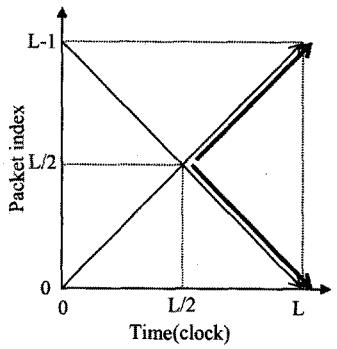


그림 8. Double flow 기법
Fig. 8. Double flow method.

처음 L/2 클록 동안 계산된 FSM과 RSM을 메모리에 저장한 후 다음 L/2 클록 동안 저장된 FSM과 RSM을 메모리에서 읽어 LLR 값을 계산하면 된다.

Double flow 기법은 한 개의 FSM 프로세서, 한 개의 RSM 프로세서, 그리고 두 개의 LLR 프로세서를 필요로 한다. 그리고 깊이 L/2의 FSM 메모리, 깊이 L/2의 RSM 메모리를 한 개씩 필요로 한다.

Ⅲ. MDF 기법

1. Double flow 기법의 문제점

Double flow 기법은 한 클록에 LLR 값을 두 개씩 출력한다. 그리고 LLR 값의 패킷 색인은 한 개는 L/2에서 증가하는 순서로, 나머지 한 개는 L/2-1에서 감소하는 순서로 나오게 된다.

QPP 인터리버가 메모리 경합이 없기 위해서는 각 병렬 복호 모듈의 LLR 값 출력이 한 클록에 같은 오프셋(offset) 값을 가져야 한다. 즉 하나의 LLR 값이 패킷 색인이 증가하는 순서로 출력되면 나머지 LLR 값들도 패킷 색인이 증가하는 순서로 출력되어야 한다. 하지만 double flow 기법에서는 한 프로세서 내의 LLR 출력이 서로 다른 순서로 출력되므로 QPP 인터리버를 사용하더라도 메모리 경합이 발생하게 된다.

따라서 double flow 기법을 QPP 인터리버에 적용하기 위해서는 LLR 값의 출력 순서를 조정하여야 한다. 다음 절에 QPP 인터리버에 적용 가능하도록 double flow 기법을 수정한 방법을 보인다.

2. MDF 기법

그림 9에 double flow 기법을 수정한 MDF(modified double flow) 기법을 그래프 형태로 보인다. 그림 9와 같은 동작을 하기 위해서는 RSM을 저장하는 깊이 L/2의 쓰기 포트와 읽기 포트를 가진 듀얼 포트(dual port) 메모리가 필요하다.

처음 L/2 클록 동안 RSM을 계산하여 메모리에 저장한다. 그리고 L/2 클록 후에는 메모리에 저장된 RSM과 계산 중인 FSM을 이용하여 패킷 색인 L/2부터 L-1까지의 LLR 값을 계산하여 출력하고 패킷 색인 L/2-1부터 0까지의 RSM을 메모리에 저장한다. 다음 L/2 클록 동안 메모리에 저장된 RSM과 계산 중인 FSM을 이용하여 패킷 색인 0부터 L/2-1까지의 LLR 값을 계산하여 출력한다. 한 클록에 한 개의 LLR 값이 출력되므로 경합이 발생하지 않지만 복호 시간이 1.5L 클록으로 늘어난다.

표 1. 복호 기법들의 비교
Table 1. Comparison of decoding methods.

복호 방식	FSM proc.	RSM proc.	LLR proc.	메모리 구성	복호 시간
전통적인 기법	1	1	1	L	2L
Double flow	1	1	2	2(L/2)	L
MDF 기법	1	1	1	L/2	1.5L

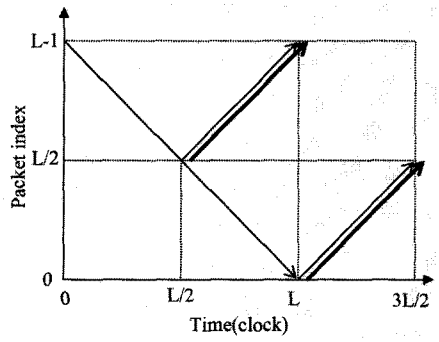


그림 9. MDF 기법
Fig. 9. MDF method.

MDF 기법은 FSM 프로세서, RSM 프로세서, 그리고 LLR 프로세서를 각각 한 개씩 필요로 한다. 그리고 깊이 L/2의 RSM 메모리를 필요로 한다. Double flow 기법에 비해 LLR 프로세서와 메모리 개수가 반으로 줄었다.

표 1에 지금까지 소개한 처리 기법들의 프로세서 및 메모리 구성과 복호 시간을 보인다. 전통적인 처리 기법은 각각 한 개의 FSM, RSM, LLR 프로세서를 사용하지만 깊이 L의 메모리를 사용하고 복호 시간이 2L이 걸려 비효율적이다. Double flow 기법은 LLR 프로세서를 두 개 사용하며 메모리 또한 총 깊이 L을 사용하지만 복호 시간은 L로 가장 짧다. 하지만 double flow 기법은 QPP 인터리버를 사용할 경우 메모리 경합이 발생하여 적용하기에 적절하지 못하다. MDF 기법은 각각 한 개의 FSM, RSM, LLR 프로세서를 사용하고 메모리 또한 깊이 L/2를 사용하여 최소의 하드웨어 구성을 보인다. 그리고 복호 시간은 1.5L이 걸리지만 QPP 인터리버를 사용할 경우 메모리 경합이 발생하지 않는다.

그림 10에 MDF 기법을 적용하여 병렬 처리하는 방식을 그래프 형태로 보인다. 길이가 K=ML인 패킷에 대해 M개의 복호 모듈이 각각 패킷 길이 L 만큼을 담당하여 복호를 하여 1.5L 클록 동안 복호를 한다. M개의 LLR 값들은 QPP 인터리버로 전달되어 메모리 경합

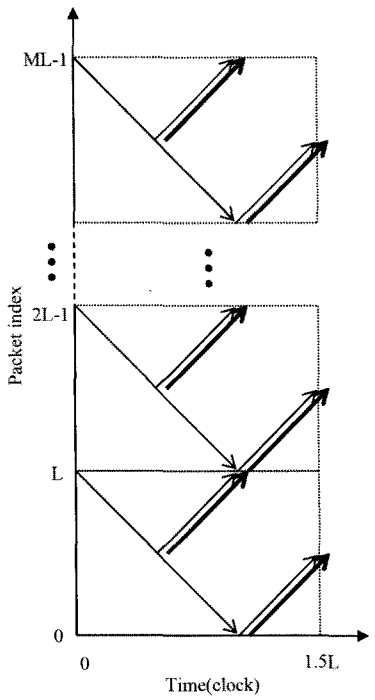


그림 10. MDF 기법의 병렬 처리
Fig. 10. Parallel processing of MDF method.

없이 인터리브 또는 디인터리브를 수행한다.

한 번 반복의 복호 시간은 1.5L 클록이므로 패킷 길이 K=ML에 대한 처리율은 수식 (12)와 같다.

$$Throughput = \frac{M}{eration \times 3} \times 100\% \qquad (12)$$

Iteration은 반복 횟수이고 한 번의 반복은 인터리브와 디인터리브를 거치기 때문에 총 3L 클록이 걸리게 된다. 입력은 총 복호 시간 iteration*3L 클록마다 K=ML을 받을 수 있으므로 ML을 iteration*3L로 나눈 값에 100을 곱하여 처리율을 구할 수 있다. 단, 처리율은 100%를 넘을 수 없다.

3. MDF 기법을 이용한 MAP 복호 모듈 설계

그림 11에 각 연산 프로세서의 동작을 보인다. 그림 11의 직사각형들은 연산 및 동작을 의미한다. 각 연산 및 동작 뒤의 화살표는 패킷 색인의 순서를 의미하는데 오른쪽 위로 향하는 화살표는 패킷 색인이 증가하는 순서를 의미하고, 오른쪽 아래로 향하는 화살표는 패킷 색인이 감소하는 순서를 의미한다. 굵은 화살표는 연산 결과의 흐름을 의미한다. FBM은 FSM 연산을 위한 가지 값을, RBM은 RSM 연산을 위한 가지 값을 의미한다.

RSM 연산 동작을 살펴보면 먼저 RBM 프로세서에서 패킷 색인이 L-1부터 0까지 감소하는 순서대로 RBM을 계산한다. RSM 프로세서에서는 계산된 RBM을 이용하여 패킷 색인 L-1부터 0까지 감소하는 순서대로 RSM을 계산한다. 계산된 RSM은 패킷 색인이 감소하는 순서대로 RSM 메모리에 써진다. 저장된 RSM은 L/2 클록 후 패킷 색인이 증가하는 순서대로 읽혀진다.

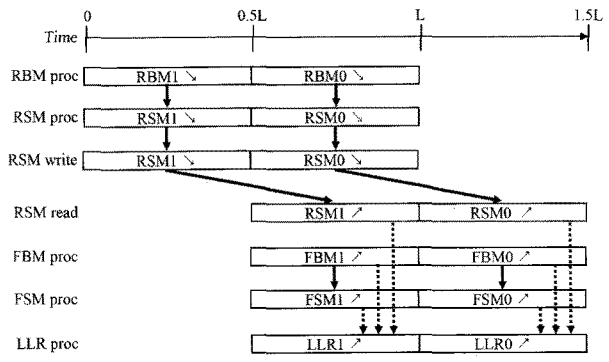


그림 11. MDF 기법의 동작
Fig. 11. Working of MDF method.

FSM 연산 동작을 살펴보면 먼저 FBM 프로세서에서 패킷 색인 L/2부터 L-1까지 증가하는 순서대로 FSM을 계산한다. 그 후 패킷 색인 0부터 L/2-1까지의 FBM과 FSM을 다시 계산한다.

마지막으로 LLR 값 연산 동작을 살펴보면 L/2+1 클록부터 L 클록까지 FSM, FBM, RSM의 연산 결과 및 메모리 출력이 모두 패킷 색인 L/2-1부터 L-1까지 증가하는 순서대로 나온다. 따라서 LLR 프로세서에서는 FSM, FBM, RSM을 이용하여 패킷 색인 L/2-1부터 L-1까지 증가하는 순서대로 LLR 값을 계산하여 출력한다. 그리고 L+1 클록부터 1.5L 클록까지 FSM, FBM, RSM의 연산 결과 및 메모리 출력이 모두 패킷 색인 0부터 L/2-1까지 증가하는 순서대로 나온다. 따라서 LLR 프로세서에서는 FSM, FBM, RSM을 이용하여 패킷 색인 0부터 L/2-1까지 증가하는 순서대로 LLR 값을 계산하여 출력한다.

그림 11의 연산 흐름을 살펴보면 하드웨어의 구조를 예측할 수 있다. 우선 FBM 프로세서의 결과가 FSM 프로세서로 입력되고, FBM 프로세서의 결과와 FSM 프로세서의 결과는 LLR 프로세서로 이어짐을 알 수 있다. 그리고 RBM 프로세서의 결과가 RSM 프로세서로 입력되고, RSM 프로세서의 결과는 RSM 메모리로, 그리고 RSM 메모리의 출력은 LLR 프로세서로 이어짐을 알 수 있다.

FBM 프로세서와 RBM 프로세서는 동시에 연산을 시작하여야 하므로 MAP 복호 모듈의 입력은 정보 비트와 패리티 비트, 사전 정보 값이 한 쌍으로 입력되어야 한다. 그림 11의 동작을 근거로 그림 12에 복호 모듈의 구조를 보인다.

MAP 복호 모듈 외부에서 한 쌍의 정보 비트(XA, XB), 패리티 비트(YA, YB), 사전 정보 값(LLRA, LLRB)가 입력되면 이는 각각 FBM 프로세서와 RBM 프로세서로 입력되어 가지 값을 계산한다. 계산된 가지 값은 각각 FSM 프로세서와 RSM 프로세서, 그리고 LLR 프로세서로 보내어진다. 그리고 LLRA 값은 LLR 프로세서로 보내어져 LLR 프로세서의 연산 결과에 LLRA 값을 뺀 값이 LLR 프로세서의 최종 출력이 된다. 그리고 RSM 프로세서의 출력은 RSM 메모리에 저장된 후 L/2 클록 후에 순서가 바뀌어 LLR 프로세서로 보내어지고, LLR 프로세서에서는 각 프로세서의 출력을 받아 최종 LLR 값을 계산한다.

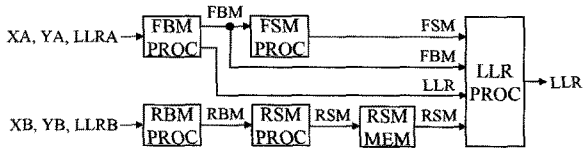


그림 12. MDF 기법을 적용한 복호 모듈 구조
Fig. 12. Architecture of decoding module using MDF method.

Ⅳ. 실험 결과

1. 복호기 전체 구조와 동작

그림 13에 병렬 터보 복호기의 전체 구성을 보인다. 프로세서는 총 32개의 병렬 프로세서로 구성되어 있으며, 입력 블록(IB), MAP 복호 블록(MDB), 인터리브/디인터리브 주소 생성 블록(IDAGB), 인터리브/디인터리브 블록(IDB), 출력 블록(OB)의 다섯 부분으로 구성되어 있다.

외부에서 패킷이 들어오면 IB에 정보 비트와 패리티 비트, 그리고 꼬리 비트가 저장됨과 동시에 IDAGB에서 주소를 생성하여 IDB의 매퍼 메모리에 저장한다. 패킷 입력이 끝나면 복호를 시작한다. 입력 블록에서 정해진 순서대로 MDB에 인터리브를 위한 정보 비트와 패리티 비트, 꼬리 비트를 전달하면 MDB는 복호를 수행하여 IDB로 LLR 값을 전달한다. IDB는 인터리브를 수행하여 IB의 LLR 메모리에 값을 저장한다.

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 출력 값을 출력한다.

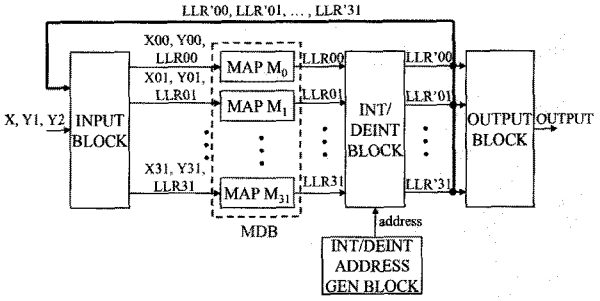


그림 13. 병렬 터보 복호기의 구성
Fig. 13. Parallel turbo decoder.

2. 시뮬레이션 결과

구현한 프로세서의 검증을 위해 단계별로 시뮬레이션을 하였다. 시뮬레이션은 C 언어로 작성하였으며 Visual C++ 6.0에서 실행하였다. 본 프로세서의 정보 비트와 패리티 비트, 그리고 꼬리 비트는 9비트를 입력으로 하였다. 내부적으로 LLR 값 9비트, 가지 값 9비트, 상태 값 13비트를 사용하였으며, FSM과 RSM, LLR 값 계산 시 13비트 모듈로 연산을 하였다[13].

그림 14는 병렬 처리를 하지 않은 log-MAP 복호 방식, MDF 기법, 비트 고정 MDF 기법의 SNR 대비 BER 성능을 보인다. 비트 고정 MDF 기법은 실제 하드웨어 구현을 고려한 실험 결과로 구현된 복호기와 동일한 결과를 가진다. K=4096이며 반복 횟수는 8이다.

비트 고정 MDF 기법은 최악의 경우에도 병렬 처리를 하지 않은 결과와 0.1dB 이하의 BER 성능 저하를 보이고 있으며 0.7dB에서 BER 10^{-7}을 만족하였다.

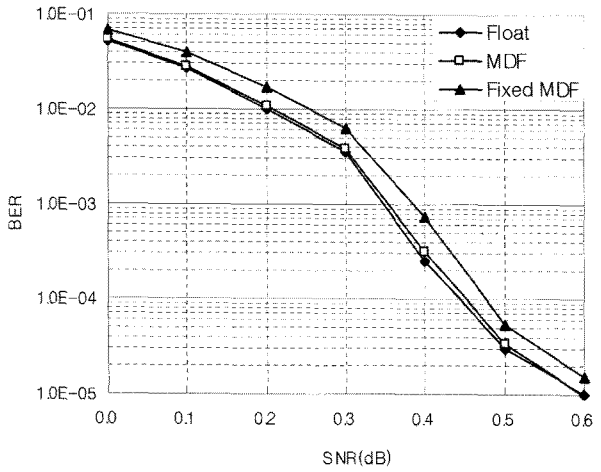


그림 14. BER 성능(K=4096, iteration=8)
Fig. 14. BER performance(K=4096, iteration=8).

3. 구현 결과

본 설계는 VHDL로 구현하였고, Synplify Pro 8.5에서 합성(synthesize)하였으며 Xilinx ISE 8.2i에서 번역(translate), 맵(map), 배치 및 배선(place and route)되었다. 장치는 Xilinx 사의 FPGA(Field Programmable Gate Array)인 Virtex4 xc4vlx200-11ff1513에 타겟팅(targeting)하였다. 클록 주파수 80MHz의 제한(constraint)을, 핀 인/핀 아웃(pin in/pin out) 지연 제한은 12.5ns를 주어 합성, 맵, 배치 및 배선을 수행하였다. 표 2에 맵을 수행한 결과를 보인다. slice 플립플롭은 58,847개를 사용하여 33%를, 4 입력 LUT(Look Up Table)는 116,426개를 사용하여 65%를 사용하였다. 이를 slice로 환산하면 77,008개를 사용하여 총 86%의 slice를 사용하고 있다. 사용한 블록 램(block RAM)의 개수는 294개를 사용하여 87%를 사용하고 있다.

표 2. 맵 수행 결과
Table 2. Result of map.

사용한 회로	사용 회로 갯수	사용률(%)
Slice flip-flop	58,847	33
4 input LUT	116,426	65
Slice	77,008	86
Block RAM	294	87

표 3. 배치 및 배선 수행 결과
Table 3. Result of placement and route.

	작동 주파수 및 클록
Clock frequency	80MHz
Critical path	12,513ns
Pin in delay	11.392ns
Pin out delay	10.328ns

표 3에 배치 및 배선을 수행한 결과를 보인다. 클록 주파수는 80MHz에서 작동함을 확인하였고, 임계 경로(critical path)의 지연 시간은 12.513ns였다. 핀 인/핀 아웃 지연 시간은 11.392ns와 10.328ns로 모두 제한을 만족하였다.

구현한 복호기가 길이 K인 패킷을 복호하는 시간은 수식 (13)과 같다.

$$iteration \times (26 + 3K/P) \tag{13}$$

한 번의 복호를 하는데 $13+1.5K/P$ 클록의 시간이 필요한데 이는 다음과 같은 지연 시간의 합으로 이루어진다. IB에서 시작 신호를 받고 MDB로 정보를 보내는데 3 클록, MDB에서 삽입된 파이프라인 레지스터에서 8 클록, IDB에서 LLR 메모리나 출력 메모리로 정보를 보내는데 2 클록이 소요되어 총 13 클록이 항상 지연된다. 그리고 II장에서 설명한 지연 시간 $1.5K/P$가 지연되어 총 $13+1.5K/P$ 클록이 지연된다. 그리고 한 번 반복은 인터리브와 디인터리브를 수행하여 내부적으로 두 번의 복호를 수행하기 때문에 이에 2를 곱하여 $26+3K/P$ 클록이 지연되고 여기에 반복 횟수를 곱하면 총 복호 시간을 구한다.

표 4에 반복 횟수 8일 경우에 패킷 크기에 따른 처리 속도를 보인다. 수식 (13)에 의해 패킷 당 복호 시간을 계산하고, K 값을 필요 클록 수로 나누어 효율을 구

표 4. 디코더의 처리율
Table 4. Throughput of decoder.

K	복호 시간/패킷 (클록)	효율 (K/복호 시간, %)	처리율 (Mbps)
40	1168	3.4	2.72
512	6352	8.1	6.48
1024	6352	16.1	12.88
2048	6352	32.2	25.76
4096	6352	64.5	51.60
6144	4816	100	80.00

한다. 만약 복호 시간이 패킷을 입력받는 시간보다 짧다면 입력은 연속적으로 들어갈 수 있으므로 효율은 100%이다. 구해진 효율에 동작 클록 주파수를 곱하여 처리율을 계산한다. K 값이 작으면 처리율이 매우 낮으며, K 값이 클수록 처리율은 좋아진다.

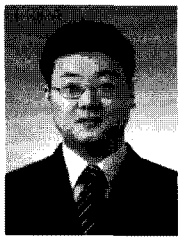
V. 결 론

본 논문에서는 MDF 기법을 제안하고, MDF 기법을 적용한 MAP 복호 모듈의 설계와 제어를 설명하였다. MDF 기법은 QPP 인터리버를 사용한 병렬 처리를 할 경우 인터리브, 디인터리브 시 메모리 경합이 발생하여 효율이 떨어지는 double flow 기법을 수정하였다. MDF 기법을 적용한 MAP 복호 모듈은 병렬 처리를 하여도 인터리브, 디인터리브 시 메모리 경합이 발생하지 않는다. 또한 LLR 프로세서와 메모리의 개수를 반으로 줄여 회로의 크기를 줄였다. 그리고 유동 소수점 연산을 한 시뮬레이션 결과와 최대 0.1dB 이하의 성능 감소를 보여 BER 성능이 좋으면서도, 최대 80Mbps의 처리율을 갖는 고속 동작을 한다.

참 고 문 헌

[1] C. Berrou, A. Glavieux, P. Thitimajshima, "Near Shannon Limit Error-correcting Coding and Decoding : Turbo-codes", *ICC 1993*, pp. 1064 - 1070, Geneva, Switzerland, May 1993.
[2] "Multiplexing and Channel coding (TDD)", 3G TS 25.222 V3.1.1 (1999-12), pp. 12 - 16.
[3] "Physical Layer Standard for cdma2000 Spread Spectrum Systems (IS-2000-2)", TIA, pp. 3.73 - 3.79.
[4] J. Kwak, S. M. Park, S. S. Yoon, K. Lee, "Implementation of a Parallel Turbo Decoder with Dividable Interleaver", *ISCAS 2003*, pp. II-65 - II-68, May 2003.
[5] 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.
[6] O. Y. Takeshita, "On Maximum Contention-Free Interleavers and Permutation Polynomials over Integer Rings", *IEEE Transactions on Information Theory*, Vol. 52, No. 3, pp. 1249 - 1253, Mar. 2006.
[7] Broadcom, "Flexible and Formulaic Collision-free Memory Accessing for Parallel Turbo Decoding with Quadratic polynomial permutation (QPP) Interleave", 3GPP TSG RAN WG1 #47 BIZ, R1-070618.
[8] C. Schurgers, F. Catthoor, M. Engels, "Optimized MAP Turbo Decoder", *SiPS 2000*, pp. 245 - 254, Oct. 2000.
[9] I. Al-Mohandes, M. Elmasry, "A Low-Power 5Mb/s Turbo Decoder for Third-Generation Wireless Terminals", *CCECE 2004*, pp. 2387 - 2390, Niagara Falls, May 2004.
[10] "Technical Specification Group Radio Access Network; Multiplexing and channel coding (FDD)(Release 6)", 3GPP TS 25.212 V6.7.0, 2005-12.
[11] 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.
[12] Ericsson, "Contention-Free Interleavers vs. Contention-Avoidance Decoding Solutions", 3GPP TSG-RAN WG1 #47 bis, R1-070463.
[13] B. Shim, H. G. Myung, "A Novel Metric Representation for Low-Complexity Log-MAP Decoder", *ISCAS 2005*, pp. 5830 - 5833, May 2005.

저 자 소 개



정 재 헌(정회원)
2006년 서강대학교 기계공학과 학사 졸업.
2008년 서강대학교 컴퓨터공학과 석사 졸업.
2008년 현재 삼성전자 DMC연구소 선임연구원.
<주관심분야 : 통신, ASIC 설계, High Level Synthesis, Electric System Level 구조 설계>

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