논문 2011-48SD-1-4

메모리 사용을 최적화한 부분 병렬화 구조의 CMMB 표준 지원 LDPC 복호기 설계

(A Memory-efficient Partially Parallel LDPC Decoder for CMMB Standard)

박 주 열*, 이 소 진**, 정 기 석***, 조 성 민*, 하 진 석*, 송 용 호***

(Joo-Yul Park, So-Jin Lee, Ki-Seok Chung, Seong-Min Cho, Jin-Seok Ha, and Yong-Ho Song)

요 약

본 논문에서는 CMMB (China Mobile Multimedia Broadcasting) 표준의 LDPC(Low Density Parity Check) 부호 복호기를 효과적으로 구현하는 방법을 제안한다. 본 논문은 AGU(Address Generation Unit)와 Index 행렬을 이용하여 효율적으로 주소 값을 생성함으로써, 메모리 사용량을 줄이고 복잡도를 감소시켰다. 또한 LDPC 부호 복호기의 throughput을 향상시키기 위해 한 클럭에 여러 메시지를 전달하는 부분 병렬 구조를 사용하였고, 하나의 주소를 사용하여 병렬적으로 동작이 가능하도록 노드 그룹핑을 진행하였다. 제안하는 LDPC 부호 복호기는 Verilog HDL로 구현하였으며, Synopsys사의 Design Compiler를 이용하여 Chartered 0.18μm CMOS cell library 공정으로 합성하였다. 제안된 복호기는 455K(in NAND2)의 크기를 가지며, 185MHz의 클럭에서 1/2 부호는 14.32 Mbps의 throughput을 갖고, 3/4 부호는 26.97Mbps의 throughput을 갖는다. 또한, 기존의 CMMB용 LDPC의 메모리와 비교하여 0.39% 의 메모리만 사용된다.

Abstract

In this paper, we propose a memory efficient multi-rate Low Density Parity Check (LDPC) decoder for China Mobile Multimedia Broadcasting (CMMB). We find the best trade-off between the performance and the circuit area by designing a partially parallel decoder which is capable of passing multiple messages in parallel. By designing an efficient address generation unit (AGU) with an index matrix, we could reduce both the amount of memory requirement and the complexity of computation. The proposed regular LDPC decoder was designed in Verilog HDL and was synthesized by Synopsys' Design Compiler using Chartered 0.18μm CMOS cell library. The synthesized design has the gate size of 455K (in NAND2). For the two code rates supported by CMMB, the rate-1/2 decoder has a throughput of 14.32 Mbps, and the rate-3/4 decoder has a throughput of 26.97 Mbps. Compared with a conventional LDPC for CMMB, our proposed design requires only 0.39% of the memory.

Keywords: CMMB, Low Density Parity Check(LDPC) codes, Multi-rate support, Partially parallel decoder

* 학생회원, 한양대학교 전자컴퓨터통신학과
(Department of Electronics Computer Engineering, Hanyang University)
** 정회원, 현대자동차
(Hyundai Motors)
*** 정회원, 한양대학교 융합전자공학부
(Department of Electronic Engineering, Hanyang University)
※ 이 논문(저서)은 2006년도 정부재원(교육인적자원부 학술연구조성사업비)으로 한국학술진흥재단의 지원을 받아 연구되었음. (KRF-2006-331-D00469)
접수일자: 2010년8월19일, 수정완료일: 2010년12월28일

Ⅰ. 서 론

현재 이동통신 시스템과 차세대 방송 표준에서 오류 정정 부호로 주목받고 있는 LDPC (Low Density Parity Check) 부호는 1962년에 Gallager에 의해 제안된 block code의 일종이다[1]. 하지만, 당시의 기술력으로는 구현이 불가능하여 잊혀졌다. 이후, 1993년 터보 부호의 발견과 더불어 반복 복호 방식에 대한 많은 연구가 이루어 졌으며, 1995년에 Mackay와 Neal에 의해

구현 가능한 방법과 함께 재조명 되었다[2].

LDPC 코드는 DVB-S2 (Digital Video Broadcasting - Satellite 2nd Generation) 표준이나 802.16e 표준에서 적용되어 이를 효율적으로 구현하기 위한 많은 연구들이 진행 되고 있다.[3~5]

본 논문은 이러한 흐름에 맞춰, CMMB(China Multimedia Mobile Broadcasting) 표준에서 사용되는 LDPC 부호 복호기를 하드웨어로 구현하는 방법에 관한 것이다. DVB-S2 표준이나 802.16e 표준의 패리티 검사 행렬 H는 같은 QC(Quasi-Cyclic)-LDPC 구조로 되어 있기 때문에 하드웨어 구현 방법이 비슷하다[5]. 그러나 CMMB 표준의 경우 패리티 검사 행렬 H는 HS(High-Structured)-LDPC 구조를 갖고 있기 때문에 LDPC 부호 복호기 구현에 많은 메모리양이 요구된다. 따라서 이러한 메모리 요구량을 줄일 수 최적화 방법에 관해 서술한다.

본 논문의 구성은 다음과 같다. Ⅱ장에서 CMMB 표준의 LDPC 부호에 대하여 기존의 표준과의 차이점을 설명하고 CMMB 부호의 특성을 설명한다. Ⅲ장에서는 하드웨어 구현을 위한 구조와 주소생성 알고리즘에 대하여 제안한다. Ⅳ장에서는 구현결과를 통해 성능을 측정하였다.

Ⅱ. CMMB 표준의 LDPC 부호

CMMB는 4세대 이동 방송 표준의 하나로, 패리티 검사 행렬 H가 높은 에러 정정 능력을 갖고 있는 것으로 알려졌다[6]. CMMB 표준에서 에러 정정 부호로 LDPC 부호와 RS (Reed Solomon) 부호를 사용하며, LDPC 부호는 부호화율 1/2 과 3/4 를 선택적으로 적용하여 사용한다[7]. CMMB 표준에서 사용하는 부호의 특성은 아래의 표 1과 같다.

DVB-S2 표준에서 사용하는 패리티 검사 행렬 H는 QC-LDPC 구조로[8] Hbase 행렬, Hbase 행렬과 같은 크기의 서브 행렬 그리고 p x p 크기의 단일 행렬로 구성된다. Hbase 행렬은 p x p 단일 행렬을 cyclically shift 할 것인지 아닌지를 나타낸다. 그리고 Hbase 행렬과 같은 크기를 갖는 서브행렬은 Hbase에서 cyclically shift를 한다고 하였을 때, 몇 번의 shift를 시킬 것인지를 나타내는 행렬이다. 따라서 패리티 검사 행렬 H는 Hbase 행렬의 크기가 m x n이라고 하였을 때, mp x np의 크기를 갖게 된다. 이렇듯 QC-LDPC의 경우는 Hbase 행렬, Hbase 행렬과 같은 크기의 서브 행렬 그리고 p x p 크기의 단일 행렬을 가지고 있으면 된다. 그러나 CMMB 표준에서 사용하는 패리티 검사 행렬 H는 HS-LDPC 구조의 일종으로 QC-LDPC 구조와는 다르다. 따라서 CMMB 표준에서 사용하는 패리티 검사 행렬 H는 패턴에 따라 기본 패턴인 부분 행렬만을 저장해두고, AGU를 이용하여 나머지 주소를 생성하는 접근법이 필요하다. 이러한 내용은 아래 1/2 부호와 3/4 부호로 나누어 설명한다.

표 1. CMMB 표준의 LDPC 부호 특성
Table 1. LDPC Coding Configuration.

부호율 R	정보 블록 K	부호어 N	행의 무게 d_c	열의 무게 d_b
1/2	4,608bits	9,216bits	6	3
3/4	6,912bits	9,216bits	12	3

이에 앞서 근본적으로 패리티 검사 행렬을 저장하는 공간을 줄이기 위해 본 논문에서는 패리티 검사 행렬을 그대로 저장하는 것이 아니라 그림 1과 같이 열에서 1의 위치를 저장하는 방법을 사용한다. 하지만, 하드웨어 구현상 메모리를 통해 메시지를 주고받기 때문에, 0번째와 2번째 체크노드가 0번째 비트노드와 연결되어 있

$$H=\begin{pmatrix}1&1&1&0&0&0&0&0\\0&0&0&1&1&1&0&0\\1&0&0&1&0&0&1&0\\0&1&0&0&1&0&0&1\end{pmatrix}$$

0	1	2
3	4	5
0	3	6
1	4	7

그림 1. 패리티 검사 행렬 H와 저장되는 패리티 검사 행렬 H
Fig. 1. Parity Check H-Matrix and H-Matrix saved the position of 1's.

0	2	4
5	7	9
1	6	10
3	8	11

그림 2. 실제 저장되는 H Matrix 주소
Fig. 2. H-matrix saved the real address.

다고 해서 같은 주소인 0을 가질 수 없다. 따라서 그림 2과 같이 두 번째 등장하는 0은 1의 값을 가지게 되고 나머지 주소들도 이에 맞춰 주소 값이 변경되어야 한다. 이러한 실제 주소 값을 생성해내는 역할을 AGU에서 해주고, AGU에서 생성해낸 주소를 통해 메모리를 접근하게 된다. 본 논문의 3절에서는 이러한 역할을 하는 AGU를 어떠한 방식으로 효율적으로 설계하였는지를 다룬다.

열에서 1의 위치를 저장하는 방법을 이용하면 패리티 검사 행렬 H를 저장하는데 필요한 메모리의 크기가 M x N보다 훨씬 줄어든 M x dc(열 무게)를 저장하게 된다. 물론 저장하는 비트수가 1bit에서 log2N만큼 늘어나겠지만, N을 dc만큼 줄인 효과가 더 크기 때문에 이와 같은 방식을 사용하는 것이 효율적이다. 비록 그림 1의 경우는 패리티 검사 행렬 H의 크기가 작기 때문에 오히려 저장하는 크기가 커졌으나(4x8x1bits 에서 4x3x4bits), CMMB 표준의 1/2부호의 경우 이와 같은 방식을 이용하면 4608x9216x1bits에서 4608x6x14bits로 줄어든다(약 1%로 감소). 하지만 줄어든 패리티 검사 행렬 H의 크기 역시 아직은 매우 크기 때문에, 앞서 언급한대로 기본 패턴만을 저장하고 나머지 주소는 AGU를 이용하여 생성하는 접근법이 필요하다.

1. 균일 1/2 부호

CMMB표준에서 사용하는 균일 1/2부호의 패리티 검사 행렬 H는 4608x9216의 크기를 가지며, 열에서 1의 위치를 저장하는 방법을 사용하면 패리티 검사 행렬 H를 저장하기 위한 공간으로 4608x6x14bits를 사용하게 된다. 그러나 패리티 검사 행렬 H는 그림 3와 같이 18행마다 36bits씩 shift된 형태를 갖는다[9].

$$H_{m,n} = (H'_{(m \bmod 18),n} + 36 \times qu(m/18)) \bmod N$$

$$H'_{i,j(i=0,1,\cdots,17,\ j=0,1,\cdots,d_c-1)}$$
$$m = 0,1,\cdots,k-1 \qquad (1)$$
$$n = 0,1,\cdots,d_c-1$$

패리티 검사 행렬의 패턴에 따라서 식 (1)을 사용하면 부분행렬만을 저장하여 나머지 패리티 검사 행렬을 만들 수 있다. 식 (1)에서 qu()는 나눈 몫을 결과 값으로 내보내는 함수이다. 즉, m에 18미만의 값이 들어오면 0의 결과 값이 나오고 m에 18이상부터 36미만의 값이 들어오면 1의 결과 값이 나오는 함수이다. 행렬 H'

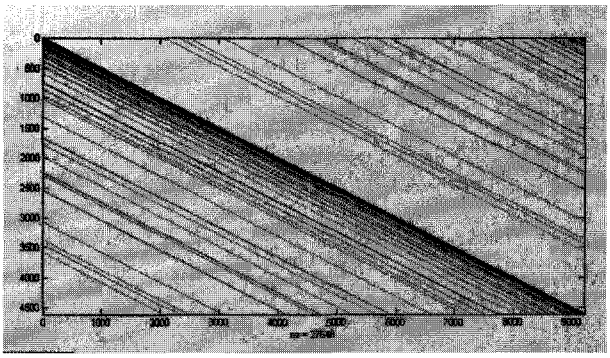


그림 3. 1/2 부호 패리티 검사 행렬 H의 패턴
Fig. 3. Rate-1/2 Parity Check H-Matrix pattern.

$$H = \begin{Bmatrix} H' \\ H'+36 \\ \vdots \\ H'+9180 \end{Bmatrix} = \begin{Bmatrix} H'_{0,0} & \cdots & H'_{0,5} \\ \vdots & & \\ H'_{17,0} & & \\ H'_{0,0}+36 & & \\ & \ddots & \vdots \\ \vdots & & \\ & & \\ H'_{17,0}+9180 & \cdots & H'_{17,5}+9180 \end{Bmatrix}$$

그림 4. 행렬 H'를 이용하여 구성된 패리티 검사 행렬 H
Fig. 4. H matrix is constructed by H'-matrix.

는 패리티 검사 행렬 H의 부분 행렬로 18 x 6의 크기를 갖는다. 식 (1)을 이용하면, 1/2 부호 복호기를 구현하기 위해서는 오직 18 x 6의 부분 행렬만을 저장함으로써, AGU를 이용하여 그림 4와 같이 나머지 패리티 검사 행렬을 생성해 낼 수 있다.

2. 균일 3/4 부호

CMMB표준에서 사용하는 균일 3/4부호의 패리티 검사 행렬 H는 2304x9216의 크기를 가지며, 열에서 1의 위치를 저장하는 방법을 사용하면 패리티 검사 행렬 H를 저장하기 위한 공간으로 2304x12x14bits를 사용하게 된다. 그러나 패리티 검사 행렬 H는 그림 5과 같이 1/2 부호와 유사하게 9행마다 36bits씩 shift된 형태를 갖는다[9].

$$H_{m,n} = (H'_{(m \bmod 9),n} + 36 \times qu(m/9)) \bmod N \qquad (2)$$

패리티 검사 행렬의 패턴에 따라서 식 (1)에서 몇 개의 파라미터만 바꾼 식을 사용하면 부분행렬을 저장하여 나머지 패리티 검사 행렬을 만들 수 있다. 1/2 부호의 경우 18개의 행마다 패턴이 반복되지만, 3/4부호의

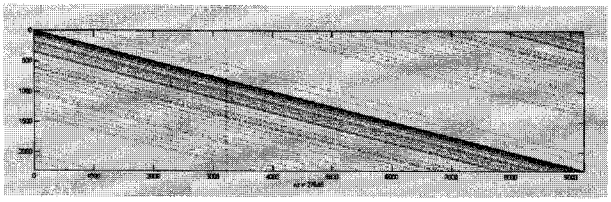


그림 5. 3/4 부호 패리티 검사 행렬 H의 패턴
Fig. 5. Rate-3/4 Parity Check H-Matrix pattern.

표 2. 패리티 검사 행렬을 저장하기 위해 요구되는 메모리
Table 2. Memory Requirement for H'-Matrix.

부호율	기존 패리티 검사 행렬 H	제안한 부분 행렬 H'	비율
1/2	387,072bits	1,512bits	0.39%
3/4	387,072bits	1,512bits	0.39%

경우는 9개의 행마다 패턴이 반복된다. 따라서 식 (1)은 식 (2)와 같이 바뀌게 된다. 9x12의 부분 행렬만을 저장함으로써, AGU를 이용하여 나머지 패리티 검사 행렬을 생성해 낼 수 있다.

이렇게 패리티 검사 행렬 H의 패턴을 통해 훨씬 적은 공간을 사용하는 부분 행렬 H'만을 저장하고, AGU를 이용하여 나머지 주소 값을 생성해줌으로써 저장하는 메모리 공간을 줄일 수 있다. 표 2는 기존의 패리티 검사 행렬 H를 열에서 1의 위치를 저장하는 방법으로 저장하였을 때와 현재 논문에서 제안한 부분 행렬 H'을 열에서 1의 위치를 저장하는 방법으로 저장하였을 때를 비교한 것이다. 1/2 부호에서 기존의 패리티 검사 행렬 H를 저장하는 방법은 앞서 언급했듯이 4608x 6x14bits를 저장하는 것으로 4608은 정보 블록의 길이, 6은 열 무게 그리고 14는 1의 위치를 표현하기 위해 필요한 비트를 나타낸다.

Ⅲ. 제안하는 LDPC 부호 복호기

본 논문에서 제안한 LDPC 부호의 복호기는 부분 병렬 구조로 16개의 BNU와 16개의 CNU을 사용하였고, 16개의 공유 메모리를 이용하여 메시지를 전달한다. 공유메모리는 비트노드에 맞춰 저장되는 방법을 사용하기 때문에, CNU는 AGU에서 주소 값을 받아 메모리에서 데이터를 읽어 들여 연산을 진행한다. 연산이 끝나면 전달할 메시지를 동일한 주소에 쓰는 동작을 한다. 따라서 AGU가 패리티 검사 행렬 H에 맞게 메시지 충돌 없이, 메모리 주소를 정확하게 생성해줄 필요가 있다. 본 논문에서는 주소 값을 생성하는 오버헤드를 줄이고, 메시지 충돌 없이 16개의 공유 메모리를 이용하는 그룹핑 방법을 제시한다. 마지막으로 그림 2에서 설명했던 것과 같이 동일한 메모리 주소에 덮어 쓰는 일이 없게 하기 위해, AGU에서 메모리 주소를 생성할 때 인덱스 행렬 I를 사용하여 주소를 생성하는 방법을 제안한다.

1. 부분 병렬 구조

LDPC 부호 복호기를 하드웨어로 구현하는데 있어서 구조로 보아 크게 직렬, 병렬 그리고 부분 병렬 구조로 구분된다. 본 논문에서 제안한 부분 병렬 구조는 각각 16개의 BNU와 CNU를 사용하여 그림 6과 같이 구성되어 있다. 그림 6에서 알 수 있듯이, CNU에서 연산을 진행할 때 패리티 검사를 동시에 진행하도록 되어 있다. 이와 같이 실행을 하면 패리티 검사를 통과하는 마지막 복호 때 패리티 검사 때 필요 없는 체크노드 연산을 한 번 더 진행하는 단점이 있다. 그러나 별도의 클럭을 소비하지 않고도 패리티 검사를 진행할 수 있다는 장점이 있고 이로 인해 높은 throughput을 가질 수 있으므로 본 논문에서는 체크노드 연산과 패리티 검사를 동시에 하는 방법을 채택하였다.

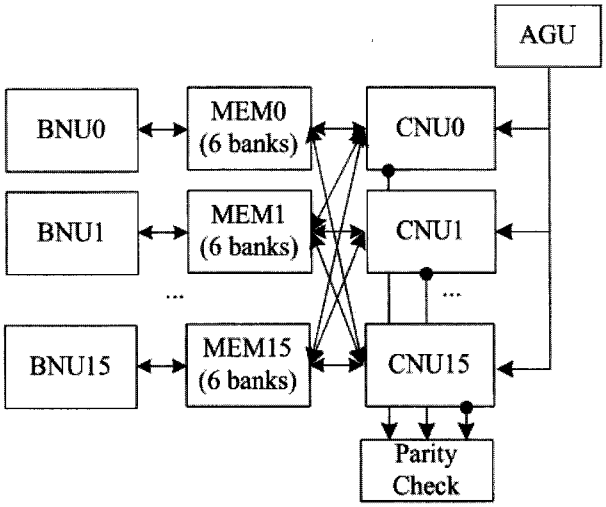


그림 6. 부분 병렬 구조의 LDPC 부호 복호기
Fig. 6. Partially parallel architecture of LDPC decoder.

2. 노드 그룹핑

CMMB 표준에서 사용하는 패리티 검사 행렬 H는

앞서 언급했듯이 1/2 부호는 총 18행의 1의 위치를, 3/4 부호에서는 총 9행의 1의 위치를 저장해둠으로써 나머지 행에서의 주소 값을 생성할 수 있다. 본 절에서는 노드를 어떻게 그룹핑하면 동시에 병렬적으로 동작하게 할 수 있는지를 다룬다. 이를 위하여 식 (1)을 체크노드 관점으로 표현할 수 있고, 이는 식 (3)과 같다.

$$c_i = (H'_{(i \bmod 18),n} + 36 \times qu(i/18)) \bmod N \quad i = 0,1,...,k-1 \tag{3}$$

식 (3)을 통해서 우리는 i를 18로 나눈 나머지 값이 같으면 행렬 H′의 같은 위치에서 값을 읽어온다는 것을 알 수 있다. 이렇게 행렬 H′에서 읽어온 값에 i를 18로 나눈 몫을 더함으로써 실제 읽어올 주소 값을 생성하게 된다. 따라서 18개씩 체크노드를 묶어서 체크노드 그룹으로 나눌 수 있고, 이렇게 18개씩 노드를 묶는 방법은 CMMB 표준의 LDPC 부호 복호기를 부분 병렬구조로 설계할 때 병렬성을 최대로 높이는 방법이다. 18개씩 체크노드를 묶어서 체크노드그룹을 나눈다면 총 256개의 노드그룹이 생기게 되고, i가 0, 18, …, 4590인 체크노드들이 동시에 동작하게 된다. 그러나 이와 같이 18개씩 노드를 묶는 경우 각각 256개의 CNU, BNU 그리고 공유 메모리가 필요하기 때문에 로직의 크기가 매우 커지고 컨트롤 로직을 구성하기 매우 어렵다.

우리는 식 (3)을 통해, 18개씩 체크노드를 묶는 것도 가능하지만 18의 배수가 되는 노드의 수로 노드그룹을 묶는 경우도 병렬적으로 동작이 가능하다는 것을 알 수 있다. 체크노드 36개씩 묶어서 128개의 노드그룹, 체크노드 72개씩 묶어서 64개의 노드그룹, 32개의 노드그룹, 16개의 노드그룹, 8개의 노드그룹 그리고 4개의 노드그룹으로 나눠 부분 병렬구조로 설계할 수 있는 것이다. 4개나 8개의 노드그룹으로 나누는 경우 CMMB 표준에서 제시한 throughput을 만족시킬 수 없고, 32개 이상의 노드그룹은 throughput을 만족하나 로직의 크기가 매우 커진다. 따라서 본 논문에서는 throughput을 만족시키면서 로직의 크기가 상대적으로 작은 16개의 노드그룹을 선택하여 부분 병렬구조로 설계하였으며, 이러한 그룹핑의 모습은 그림 7과 같다. 18의 배수인 288개의 연속적인 체크노드를 그룹핑하여 16개의 그룹으로 나누면, 그림 7과 같이 16개의 CNU는 모듈로 연산의 결과 값이 갖게 되어서 병렬적으로 진행이 가능하다. 제안된 구조는 16개의 공유메모리를 갖고, 공유메모리는 BNU가 연산을 진행할 때에 순차적으로 읽어갈 수 있도록 설계하였다. 따라서 576개의 연속적인 비트노드

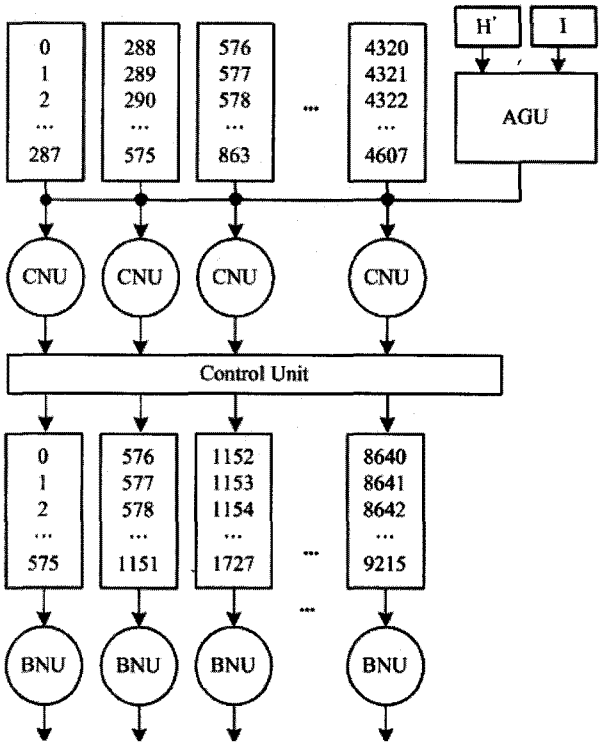


그림 7. 부분 병렬 구조의 LDPC 부호 복호기그룹핑을 통한 병렬 처리(1/2 부호)
Fig. 7. Parallel processing with grouping (rate-1/2).

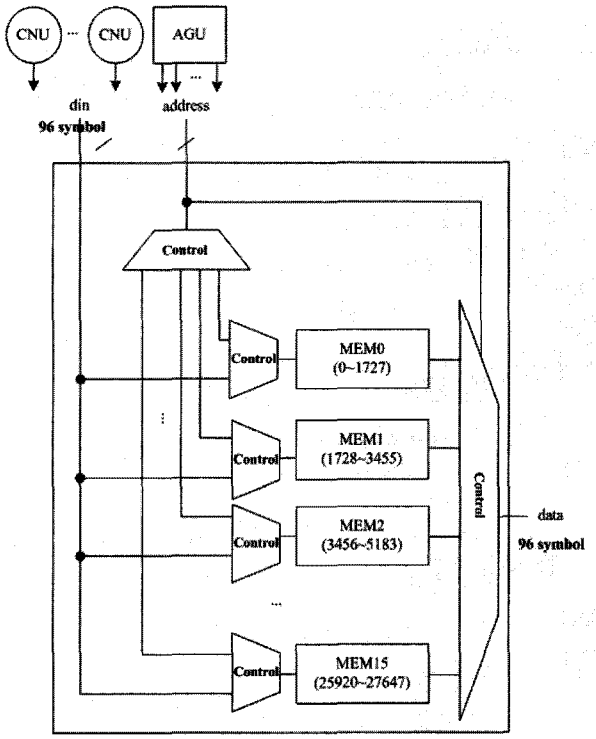


그림 8. CNU에서 메모리를 선택하여 읽고 쓰는 동작
Fig. 8. Read/Write operation at CNU.

를 그룹핑하여 16개의 그룹으로 나누고, 그룹은 16개의 공유메모리에 각각 맵핑된다.

CNU의 경우 그림 8를 통해 알 수 있듯이, 처음에 체크노드c_0, c_{288}, c_{576}, …, c_{4320}이 병렬적으로 진행된다. AGU에서는 식 (3)에 따라, 행렬 H′에서 0번째 행이 선택되고(0, 288, 576, …, 4320 모두 18로 나눈 나머지는 0이기 때문에), 0번째 행에 6개의 주소를 바탕으로 16개의 공유 메모리에서 주소에 맞는 메모리를 선택하여 데이터 값을 읽어 들인다. 읽어 들인 데이터로 CNU연산을 마치면 같은 주소에 쓰는 작업을 한다.

앞서 언급했듯이, BNU의 경우는 CNU가 AGU에서 주소를 받아 열의 위치에 맞춰 나눠진 공유 메모리에 저장해두었기 때문에 각각 자신에게 연결된 메모리에서 순서대로 3개의 데이터를 읽어 들여 계산하면 된다. BNU의 읽거나 쓰는 동작은 모두 자신에게 맵핑된 메모리에서만 진행되기 때문에, 역시 16개의 그룹이 동시에 진행되며 AGU 없이 주소 값을 순차적으로 증가시키면 되기 때문에 동작이 간단한다.

3. 인덱스 행렬

공유 메모리는 열의 위치에 맞춰 데이터를 저장하고 있다. 그림 9는 1/2 부호에서 저장하고 있는 행렬 H′의 18행 중 일부와 이를 이용하여 생성한 주소 값으로 실제 메모리에 맵핑되는 모습을 나타낸다. 그림 9의 오른쪽 행렬에 숫자는 체크노드가 어떤 비트노드와 연결되어 있는지 비트노드 번호를 나타내는 것으로, 예를 들어 c_0는 b_0, b_6, b_{12}, b_{18}, b_{25}, b_{30}과 연결되어 메시지를 전달받는 것을 나타낸다.

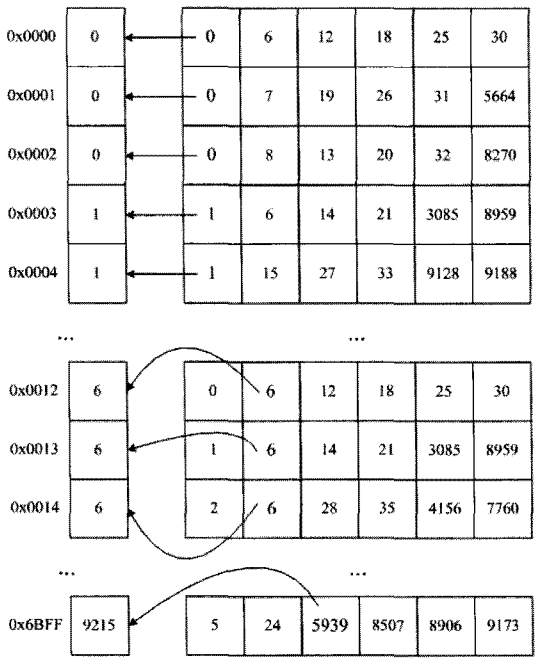


그림 9. 행렬 H′와 메모리 맵핑
Fig. 9. Address generation by H′-matrix.

그림 2을 통해 언급했듯이, 같은 비트노드 번호가 나왔을 때 데이터를 덮어쓰는 일이 없게 하기 위해 다른 주소를 할당하여야 한다. 1/2 부호나 3/4 부호 모두 열 무게가 3이기 때문에 하나의 비트노드에는 총 3개의 체크노드가 연결되어 있고, 따라서 CNU연산을 할 때 같은 비트노드가 3번 나타난다. 3번 나오는 같은 비트노드 번호를 같은 주소로 할당하면 데이터가 덮어 씌워지기 때문에 한 비트노드 번호 당 3개의 저장 공간이 필요하게 된다. 그림 9와 같이 c_0와 연결된 b_0는 0의 주소 값을 갖게 되지만 c_1과 연결된 b_0는 1의 주소 값을 가져야 한다. 즉, c_0와 연결된 b_6은 18의 주소 값을, b_{12}는 36의 주소 값을 갖게 된다.

그러나 이와 같이 주소 값을 생성하기 위해서는 같은 번호의 비트노드가 몇 번 나왔는지를 세어서 저장하는 레지스터가 필요하다. 비트노드마다 인덱스를 저장하고 있는 2bits 레지스터를 할당하고 주소 값을 생성할 때마다, 인덱스 값을 더하여 주소 값을 생성하고 인덱스 값을 업데이트 해줘야한다. 이러한 방법을 이용하면 매번 주소를 생성할 때에 계산해야 하기 때문에 오버헤드가 크다.

다른 방법으로 각 체크노드에 연결된 비트노드의 인덱스를 미리 생성해서 저장해두는 방법으로, 4608x6x2bits의 추가적인 메모리 공간이 필요하게 된다. 본 논문에서는 이러한 인덱스의 패턴을 파악하여, 18x6의 인덱스 행렬 I를 저장하는 방법을 제안한다. 18x6x2bits 크기의 인덱스 행렬을 저장하여 실제 주소를 생성한다는 것은 처음 등장하는 비트노드가 인덱스를 무조건 0을 가져야 하는 것이 아니라 1이나 2를 가질 수 있음을 뜻한다. 이와 같은 것이 가능한 이유는 AGU가 항상 같은 행렬 H′와 행렬 I를 이용하여 주소를 생성하기 때문이다.

그림 10은 행렬 H′와 행렬 I를 이용하여 주소를 생성하는 것을 나타낸다. 이해를 돕기 위하여 오른쪽에는 실제 패리티 검사 행렬 H의 행 번호 i(체크노드 ci)와 행렬 H′의 행 번호, 그리고 행렬 I의 행 번호를 표시하였다. 그림 10를 통해 알 수 있듯이, 행렬 H′와 행렬 I는 같은 행을 읽어 들여 주소 값을 생성한다. 따라서 각

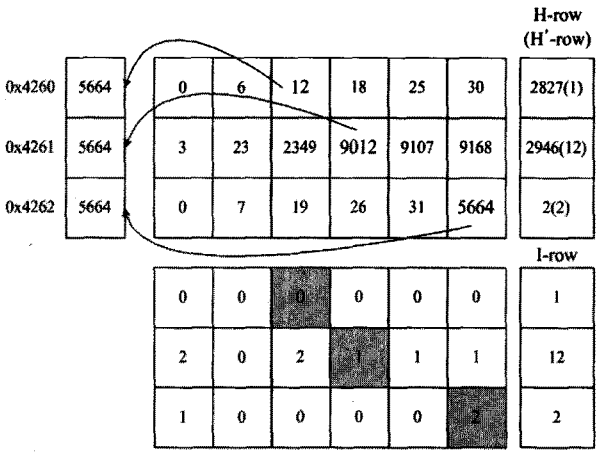


그림 10. 행렬 I를 이용한 주소 생성
Fig. 10. Address generation by I-matrix.

행에서의 1이 위치하는 열 번호를 생성하는 식 (3)에 연산을 추가하여 실제 생성되는 메모리 주소 값은 식 (4)에 의해 생성된다.

$$H_{m,n} = 3\times(H'_{(m \bmod 18),n} + 36\times qu(m/18)) \bmod N + I_{(m \bmod 18),n} \quad (4)$$

그림 10는 5664번째 비트노드와 연결된 체크노드가 메모리 0x4260번지에서 0x4262번지에 저장되는 모습을 나타낸다. c_{2327}, c_{2946} 그리고 c_2가 b_{5664}와 연결되어 있음을 알 수 있고, c_{2946}일 때에 주소 값은 아래 식 (5)에 의해 생성된다.

$$c_{2946} = 3\times(H'_{12,n} + 36\times 163) \bmod 9216 + I_{12,n} \quad (5)$$

$$3\times(3+36\times163)+2 = 17615 = 44CF_{16}, \quad n=1$$

$$3\times(23+36\times163)+0 = 17673 = 4509_{16}, \quad n=2$$

$$3\times(2349+36\times163)+2 = 24653 = 604D_{16}, \quad n=3$$

$$3\times(9012+36\times163)\bmod 9216+1 = 3\times5664+1 = 16993 = 4261_{16}, \quad n=4$$

식 (5)에 각 n을 대입시켜서 식을 풀어보면 위와 같고, 따라서 n=4일 때 0x4261의 주소 값을 생성하는 것을 확인할 수 있다. 본 논문에서는 이와 같은 방법을 통해 인덱스를 저장하는 공간을 줄였을 뿐만 아니라, 쉽게 인덱스를 읽어 들이고 식에 적용시켜 더해주기만 하면 되기 때문에 오버헤드를 감소 시켰다. 표 3은 모든 인덱스를 저장할 때와 논문에서 제안한 부분 행렬 I를 저장하였을 때를 비교한 것이다.

표 3. 행렬 I를 저장하기 위해 요구되는 메모리
Table 3. Memory Requirement for I-Matrix.

부호율	모든 인덱스 저장	제안한 행렬 I	비율
1/2	55,296bits	216bits	0.39%
3/4	55,296bits	216bits	0.39%

Ⅳ. 실험 결과

본 논문에서 제안한 LDPC 부호 복호기는 CMMB 표준을 지원하는 부분 병렬 구조의 복호기이다. CMMB 표준은 9216bits의 부호어 길이를 가지며 두 가지 부호율을 지원한다. (1) 1/2 부호로 (9216, 4608)균일 LDPC 부호, (2) 3/4 부호로 (9216, 2304) 균일 LDPC 부호. 제안하는 구조는 재구성 가능한 구조로 특성이 다른 두 가지 부호율을 모두 지원해준다. 표 4은 제안하는 재구성 가능한 구조로 두 가지 부호율을 지원하는 하나의 복호기와 한 가지 부호율을 지원하는 복호기의 크기와 throughput을 비교한 것이다. 각각의 복호기와 크기를 비교하였을 때는 당연히 본 논문에서 제안하는 재구성 가능한 복호기가 크고 느리지만, 두 가지 부호율을 지원하기 위해 두 개의 복호기를 써야 하는 상황이라면 본 논문에서 제안하는 복호기가 훨씬 면적 측면에서 이득을 봄을 알 수 있다. 본 논문에서 측정한 합성 데이터들은 Synopsys사의 Design Compiler를 사용해 Chartered 0.18μm CMOS cell library공정으로 얻은 결과이다.

제안한 복호기의 구조는 표 4에서 확인할 수 있듯이, 455K의 크기를 갖고 공유 메모리로 221,184bits 크기를 필요로 한다. 1/2 부호 복호기로 동작할 때는 185MHz의 클럭에서 14.32 Mbps의 throughput을 갖고, 3/4 부호 복호기로 동작할 때는 185MHz의 클럭에서 26.97Mbps의 throughput을 갖는다. CMMB 표준에서 요구하는 throughput은 1/2 부호일 때 10.852Mbps, 3/4 부호일 때 16.243Mbps이기 때문에 본 논문에서 제안한 복호기의 throughput은 두 가지 표준 모두 throughput 요구사항을 만족시킨다. 앞서 언급했던 것처럼, CMMB 표준을 지원하는 LDPC 부호 복호기를 설계한 논문은 발표된 바가 없다. 따라서 본 논문이 CMMB 표준을 지원하는 LDPC 부호 복호기의 첫 논문이기 때문에, 복호

표 4. 재구성 가능한 복호기의 성능 평가
Table 4. Evaluation of Reconfiguarable Decoder.

부호율	1/2 부호 복호기	3/4 부호 복호기	제안하는 재구성 가능한 복호기	
Throughput (20 iteration)	16.24	27.68	1/2 부호	3/4 부호
			14.32	26.97
MEM(bits)	221,184	221,184	221,184	
Gate Size (in NAND2)	257K	424K	455K	
Operating Freq. (MHz)	209	189	185	

표 5. AGU를 사용하지 않는 복호기와 AGU를 사용하는 복호기의 비교
Table 5. Comparison between Decoder with AGU and Decoder without AGU.

부호율		AGU를 사용하지 않는 복호기	AGU를 사용한 복호기	비율
Gate Size (in NAND2)	AGU	-	75K	-
	MEM	294K	7K	2.4%
	Total	294K	82K	28%
Throughput (20 iter)	1/2 부호	19.4Mbps	14.32Mbps	-
	3/4 부호	28.9Mbps	26.97Mbps	-

기의 성능을 비교할 참고논문이 없다. 관련 연구로 소개한 [5]의 경우는 DVB-S2 표준의 LDPC 부호 복호기를 설계한 것이기 때문에 패리티 검사 행렬 H의 구조가 다르고, 따라서 비교 대상이 될 수 없다.

본 논문에서 제안한 구조는 표 2와 표 3에서 알 수 있듯이, 패리티 검사 행렬 H나 주소를 생성하기에 필요한 행렬 I를 위한 메모리 요구량을 일반적인 방식으로 했을 때에 비해 0.39%로 매우 효과적으로 줄였다. 표 2와 표 3은 필요한 메모리 비트를 이용하여 비교한 것이고 표 5는 메모리 컴파일러를 이용하여 메모리를 생성한 후, 생성된 메모리의 크기와 추가된 로직인 AGU의 크기를 합한 전체적인 크기를 비교한 데이터이다. 또한 표 5에는 AGU로 인해 생긴 오버헤드를 반영하여 AGU를 사용하였을 때와 그렇지 않았을 때의 throughput을 비교해 놓았다. AGU를 사용하지 않는 복호기에 필요한 메모리를 메모리 컴파일러를 사용하여 합성하면 294K(in NAND2) 크기가 나온다. AGU를 사용한 복호기에 필요한 메모리는 메모리 컴파일러를 사용하여 합성하면 7K(in NAND2) 크기이고, AGU로 인하여 75K(in NAND2)의 추가 로직이 필요하기 때문에 총 82K(in NAND2) 크기라고 볼 수 있다. 이것은 AGU를 사용하지 않았을 때와 비교하였을 때 약 72%를 감소시켰다. 그러나 AGU를 사용함으로써, AGU에서 주소를 생성하는데 1cycle을 소모한다. AGU를 사용하는 부분이 critical path는 아니기 때문에 동작 속도는 변함이 없으나 노드의 연산을 진행할 때마다 1cycle의 소모로 throughput의 차이를 보인다.

참 고 문 헌

[1] R. G. Gallager, "Low density parity check codes", IRE Trans. Inform. Theory, Vol. IT-8, Jan 1962, pp. 21-28.

[2] D. J. C. Mackay and R. M. Neal, "Neal Shannon Limit Performance of Low Density Parity check codes", Electron. Lett, Vol. 32, Aug 1996, pp. 1645-1646.

[3] J. G. Park and C. H. Lee, "Architecture of an LDPC Decoder for DVB-S2 using reuse Technique of processing units and Memory relocation", Journal of the Institute of Electronics Engineers of Korea, Vol.43 SD, No.9, Sept 2006, pp. 31~37.

[4] L. Yang, M. Shen, H. Liu and C. J. R. Shi, "An FPGA Implementation of Low-Density Parity-Check Code Decoder with Multi-Rate Capability", ASP-DAC, Volume 2, Jan 2005, pp. 760-763.

[5] Z. Wang, "Low-complexity high-speed decoder design for quasi-cyclic LDPC codes," IEE Trans. VLSI systems, vol.15, no.1, 2007.

[6] Interfax China (2006-10-25). China releases mobile TV industrial standard. Retrieved on 2007-04-14

[7] Mobile Multimedia Broadcasting (P. R. China) Part 1: Framing Structure, Channel Coding and Modulation for Broadcasting Channel

[8] F. Kienle, T. Brack, and N. Wehn, "A synthesizable IP core for DVB-S2 LDPC code decoding," in Proc. DATE, Munich, Germany, Vol.3, Mar. 2005, pp. 100-105.

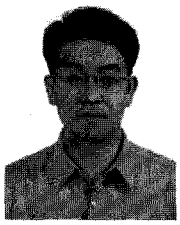
[9] Wang Peng, Chen Youg-en, "Low-complexity Real-time LDPC Encoder Design for CMMB", International Conference on Intelligent

Information Hiding and Multimedia Signal Processing, Aug. 2008, pp. 1209~1212.

저 자 소 개

박 주 열(학생회원)

2004년 아주대학교 전자공학부 학사 졸업.
2005년 LG전자 이동통신기술연구소 연구원.
2009년 한양대학교 전자컴퓨터통신공학과 석사 졸업.
2009년~한양대학교 전자컴퓨터통신공학과 박사 재학중.
<관심분야: 임베디드 하드웨어, LDPC, DSP 및 SoC 설계>



정 기 석(정회원)

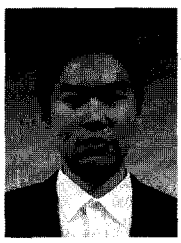
1989년 서울대학교 컴퓨터공학과 학사 졸업.
1997년 University of Illinois at Urbana-Champaign 전산학 박사 졸업.
1998년 University of Illinois at Urbana-Champaign 강의전담 교수.
2000년 Synopsys, Inc. Sr. R&D Engineer.
2001년 Intel Corp. Staff Engineer.
2001년 홍익대학교 컴퓨터공학과 조교수.
2004년~현재 한양대학교 융합전자공학부 부교수.
<주관심분야 : 멀티코어, 임베디드시스템, SoC, 저전력 시스템 설계, VLSI/CAD>

하 진 석(학생회원)

2001년 인하대학교 재료공학부 학사 졸업.
2003년 인하대학교 정보통신학부 석사 졸업.
2004년~한양대학교 전자컴퓨터통신공학과 박사 재학중.
<주관심분야: 임베디드시스템, SoC>

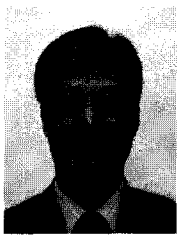
이 소 진(정회원)

2008년 한양대학교 컴퓨터학과 학사 졸업.
2010년 한양대학교 전자컴퓨터통신공학과 석사 졸업.
2010년~현대자동차 연구원.
<주관심분야 : LDPC, 고성능 멀티코어 시스템, SoC>



조 성 민(학생회원)

2006년 한양대학교 정보통신학부 학사 졸업.
2008년 한양대학교 전자컴퓨터통신공학과 석사 졸업.
2008년~한양대학교 전자컴퓨터통신공학과 박사 재학중.
<주관심분야: NoC, SoC, 멀티코어시스템>



송 용 호(정회원)

1989년 서울대학교 컴퓨터공학과 학사 졸업.
1991년 서울대학교 컴퓨터공학과 석사 졸업.
2002년 University of Southern California, Electrical Engineering, 박사 졸업.
2004년~현재 한양대학교 융합전자공학부 부교수.
<주관심분야: 임베디드시스템, SoC, NoC>