최대 데이터율을 지원하는 DAB 수신기용 Viterbi 디코더의 설계

정회원 김 효 원*, 고 우 석**, 류 주 현**, 윤 대 희*

Full Data-rate Viterbi Decoder for DAB Receiver

Hyo-Won Kim*, Woo-Suk Ko**, Joo-Hyun Ryu***, Dae-Hee Youn** *Regular Members*

요 약

DAB 시스템이 요구하는 최대 출력 데이터율을 지원하는 Viterbi 디코더의 효율적인 구조를 제안하고 설계하였다. DAB 수신기에서 Viterbi 디코더는 매우 많은 연산량을 수행하는 부분이며, 이를 위하여 고속으로 동작하는 전용 하드웨어로 설계하는 것이 바람직하다. 본 논문에서는 시스템의 전력소모를 줄이기 위하여 puncturing을 사용하는 Viterbi 디코더에 SST 방식을 적용하였다. 설계면적을 감소시키기 위하여 puncturing vector table을 수정·재배치하여 hardwired logic으로 구현하였으며, 새로운 re-scaling 방식을 제안하여 패스 메트릭을 저장하는데 필요한 워드길이을 최적화시켰다. 제안된 re-scaling 방식은 패스 메트릭을 re-scaling하는데 필요한 연산량을 크게 감소시킨다. 또한 브랜치 메트릭을 계산하는데 필요한 연산량을 줄이기 위하여 미리 계산된 값을 사용하는 방식을 제안하였다. 설계된 Viterbi 디코더는 삼성 0.35μ 표준셀 라이브러리를 이용하여 합성하였으며, 작은 면적을 차지하고 전력 소모가 적음을 확인하였다.

ABSTRACT

The efficient Viterbi decoder that supports full data-rate output of DAB system was proposed. Viterbi decoder consumes lots of computational load and should be designed to be fast specific hardware. In this paper, SST scheme was adopted for Viterbi decoder with pucturing to reduced the power consumption. Puncturing vector tables are modified and re-arranged to be designed by a hardwired logic to save the system area. New re-scaling scheme which uses the fact that the difference of the maximum and minimum of the path metric values is bounded is proposed. The proposed re-scaling scheme optimizes the wordlength of path metric memory and greatly reduces the computational load for re-scaling by controlling MSB of path metric memory. Another saving of computation is done by proposed algorithm for branch metric calculation, which makes use of pre-calculated metric values. The designed Viterbi decoder was synthesized using SAMSUNG 0.35μ standard cell library and occupied small area and showed lower power consumption.

I. 서 론

통신기술의 디지털화 추세는 점차 방송기술로도 확산되어 디지털 방송을 위한 다각적인 연구가 이루어지고 있다. 디지털 오디오 방송(DAB: Digital Audio Broadcasting)은 대용량의 멀티미디어 정보를 무선환경의 이동체에 전송하기 위해 제안된 방식으로서, FM 스테레오 라디오 기술 이후로 가장 근본적인 방송기술의 발전으로 평가받고 있다.

유럽지역을 중심으로 DAB를 실현하기 위해 몇 가지 시스템이 제안되고 실험되었으나, 현재 많은 나라에 보급되고 상용화된 시스템은 유럽의 Eureka-147로 불리우는 디지털 방송 시스템이다[1]. Eureka DAB 시스템은 OFDM (Orthogonal Frequency

* ETRI 부설 국가보안기술연구소 (hwkim@etri.re.kr), ** 연세대학교 전기전자공학과 미디어·통신 신호처리 연구실, *** LG전자
논문번호 : 010273-1009, 접수일자 : 2001년 10월 9일

Division Multiplexing) 기술에 기반하여 최대 64개의 채널 서비스를 제공한다. Eureka-147 시스템은 오류정정을 위한 채널코드로 RCPC (Rate-Compatible Punctured Convolution) 코드를 사용한다. RCPC 코드는 전송되는 신호의 민감도에 따라서 에러정정 코드의 protection 단계를 조절함으로써 전송채널의 사용을 극대화시킨다.

Eureka DAB 시스템이 요구하는 수신기 채널 디코더의 출력 데이터율은 최대 1,824 Mbps이다[1]. 이것은 최대 64개의 채널을 통해서 오디오와 데이터 서비스를 모두 지원할 경우의 값이며, 이를 구현하기 위해서는 방대한 연산량을 고속으로 수행할 하드웨어가 필수적이다. 따라서 DAB 수신기를 위한 Viterbi 디코더는 고속으로 동작하는 전용 hardwired logic으로 설계하는 것이 바람직하다. 또한 많은 연산량을 수행하기 위해 늘어날 수 있는 설계면적을 줄이고, 고속동작으로 인한 소모전력 증가를 최대한 억제하는 것이 시스템의 효율성을 좌우하는 중요한 요인이 된다.

본 논문에서는 DAB 시스템이 지원하는 최대 출력 데이터율을 만족시키고, 전력소모와 설계면적 측면에서 효율적인 Viterbi 디코더의 구조를 제안하고 설계하였다. 전력감소를 위하여 채택한 시스템 구조와 효율적으로 설계면적을 줄이는 방안에 대해서는 2장에서 자세히 기술하였다. 3장은 제안된 구조에 따라 설계된 시스템에 대한 실험적인 검증 결과를 제시하였고, 4장에서는 본 논문의 결론을 간단히 나타내었다.

Ⅱ. 제안된 Viterbi 디코더 구조

여기서는 본 논문에서 설계한 Viterbi 디코더의 target 스펙은 아래과 같다.

- 컨벌루션 코드 : (4,1,7) RCPC 코드
- 서브채널 지원 : 최대 64개의 서브채널
- 최대 출력 데이터율 : 1.824 Mbps
- 디코딩 시간 : 24 ms
- 판정방식 : 4 비트 연판정
- 시스템 동작 주파수 : 65.436 MHz (샘플링 주파수 2.048 MHz의 32배)

위에서 보는 것처럼 DAB 시스템의 full 스펙을 만족시키므로, 이를 처리할 방대한 연산량을 수행하면서 작은 면적을 차지하고 전력 소비를 줄이는 것이 설계의 목적이 된다. 판정방식은 디코더의 성능을 높이기 위해서 경판정보다는 연판정을 선택하였다. 시스템의 동작 주파수는 시스템의 복잡도를 고려해서 DAB 시스템의 샘플링 주파수의 정수배 (32배)로 선택하였다.

Viterbi 디코더를 VLSI로 설계할 때 사용전력을 감소시키기 위한 방안으로서 SST (Scarce State Transition) 방식이 사용되고 있다[2]. SST Viterbi 디코더 구조는 디코딩 과정에서 수행되는 상태천이 횟수를 최소화함으로써 소모전력을 줄이며, 최소 패스 메트릭(Path metric)을 통한 역추적 과정 성능을 향상시킴으로써 에러정정 능력을 높여준다. 본 논문에서 제안된 Viterbi 디코더는 저전력 설계를 위해 SST 방식을 적용하였으며, 전체 시스템의 구성은 그림 1과 같다.

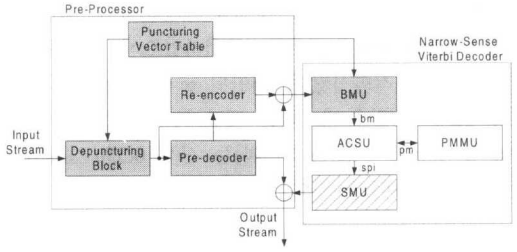


그림 1. 제안된 Viterbi 디코더의 전체 시스템 구성

전체 시스템은 연산의 특성에 따라 전처리단(Pre-Processor)과 협의의 Viterbi 디코더(Narrow-Sense Viterbi Decoder)로 구분된다. 전처리단은 DAB 스펙에서 정의하고 있는 puncturing 과정에 대한 역과정과 전력감소를 위해 채택한 SST 방식을 수행하는 블록들로 이루어져 있다. 협의의 Viterbi 디코더는 전처리단 과정들을 제외하고 순수하게 Viterbi 디코딩 과정만을 수행하는 부분이다.

전처리단을 세부적으로 살펴보면, 우선 depuncturing block이 puncturing vector table을 이용하여 1/4~8/9의 가변 코드율로 전송된 RCPC 코드를 1/4의 일정한 코드율을 갖는 4 비트 코드로 복원한다. 전처리단의 나머지 블록들은 SST 방식을 수행하는 블록들이다. 먼저 간단한 로직으로 구성된 pre-decoder와 re-encoder는 송신단에서 pucnturing을 수행하기 전의 4 비트 mother 코드를 추정한다. 추정된 mother 코드와 depuncturing 블록의 출력 코드는 XOR되어 Viterbi 디코더의 입력 데이터가 된다. 전송된 데이터와 추정된 데이터의 차이(XOR)만을 디코딩함으로써 SST 방식의 Viterbi 디코더는

불필요한 상태의 천이를 제거할 수 있다.

협의의 Viterbi 디코더 블록은 브랜치 메트릭을 계산하는 BMU (Branch Metric Unit), 패스 메트릭을 저장하고 있는 PMMU (Path Metric Memory Unit), 계산된 브랜치 메트릭과 패스 메트릭으로부터 최종적인 메트릭값을 비교하고 생존경로를 저장하는 ACSU (Add-Compare-Select Unit), 그리고 생존자 경로를 찾아내는 SMU (Survivor Memory Unit)으로 구성되어 있다. SMU출력 데이터는 전처리단의 pre-decoder 출력과 XOR되어 최종적인 Viterbi 디코더의 출력 데이터가 된다.

본 논문에서 목표로 삼은 Viterbi 디코더는 최대 출력 데이터율로서 약 1.8Mbps를 만족시켜야 하므로, 방대한 연산량을 고속으로 수행해야 한다. 고속 연산을 위한 방안으로서 전체 시스템을 여러 개의 블록들로 나눠서 파이프라인 방식으로 병렬 연산을 수행하도록 설계하였다. 이때 파이프라인 방식의 효율성을 극대화시키기 위해서는 나뉘어진 블록들이 서로 비슷한 연산량을 갖도록 해야 하며, 이를 고려하여 분할한 3개의 연산모듈들은 다음과 같다.

- 연산모듈 1 : 전처리단 + BMU
- 연산모듈 2 : ACSU + PMMU
- 연산모듈 3 : SMU

위 세 개 모듈들은 그림 1에서 각각 5개의 검게 칠해진 블록, 2개의 흰색 블록, 그리고 1개의 빗금친 블록으로 나타나 있다. 위 세 개의 모듈들은 각각 알고리듬 레벨에서 하드웨어 구현에 적합하도록 최적화시킨 후, 연산의 고속성, 하드웨어 면적, 전력 소모 등을 고려하여 설계하였다. 각 모듈에 대한 세부적인 설계 사항을 살펴보면 다음과 같다.

1. 연산모듈 1 : 전처리단 + BMU

1.1 전처리단

DAB 시스템에서 puncturing에 대한 정보는 protection profile이란 데이터를 이용해서 송수신한다. Protection profile은 protection 적용범위(L)와 사용된 puncturing vector에 대한 인덱스(PI)로 이루어져 있다[1]. 이때 FIC (Fast Information Channel)와 MSC (Main Service Channel) 채널과 적용되는 코드율에 따라 다른 개수는 L과 PI를 갖는 profile이 사용되는데, 본 논문에서는 dummy를 포함하여 각각 5개의 L과 PI를 동일하게 사용하도록 수정하였다.

적용된 puncturing의 형태는 24개의 puncturing vector로 이루어진 테이블을 참조하여 알 수 있다. 이때 각 vector는 4 비트로 이루어진 8개의 그룹으로 나뉘어지며, 각 4 비트는 어떤 비트가 살아남는가를 나타낸다 (표 1-1). 여기서 살아남는 비트는 MSB로부터 1~4개로 구성되므로, 단지 몇 개의 비트가 살아남는가의 정보만이 depuncturing에 필요하다. 이 정보는 2비트만으로 표현할 수 있으며, 이러한 성질을 이용하여 표 1-2의 테이블처럼 크기를 줄일 수 있다. 이때 표 1-2의 1열과 5열, 3열과 7열을 서로 바꾸어서 표 1-3과 같이 수정하면 규칙적인 형태의 테이블로 만들 수 있다.

테이블 정보는 일반적으로 ROM의 형태로 구현하지만, 본 논문에서는 크기가 줄고 규칙성이 증가한 성질을 이용하여 그림 3과 같은 간단한 hardwired logic으로 구현하였다. 그림 3에서 5 비트 PI값은 테이블의 행에 대한 정보를 나타내고, 3 비트 카운터 출력은 테이블의 열에 대한 정보를 나타낸다. Comp 로직과 Vpi Gen 로직은 두 개의 정보를 이용하여 128 비트로 이루어진 최종적인 puncturing vector를 발생시킨다. Hardwired logic으로 구현하면 ROM으로 구현했을 때보다 설계면적을 약 1/2 정도로 감소시킬 수 있다.

표 1-1. Puncturing vector 테이블

PI	V_{PI}
1	1100 1000 1000 1000 1000 1000 1000 1000
2	1100 1000 1000 1000 1100 1000 1000 1000
3	1100 1000 1100 1000 1100 1000 1000 1000
4	1100 1000 1100 1000 1100 1000 1100 1000
5	1100 1100 1100 1000 1100 1100 1000 1000
6	1100 1100 1100 1000 1100 1100 1100 1000
7	1100 1100 1100 1100 1100 1100 1100 1000
8	1100 1100 1100 1100 1100 1100 1100 1100
9	1110 1100 1100 1100 1100 1100 1100 1100
10	1110 1100 1100 1100 1110 1100 1100 1100
11	1110 1100 1110 1100 1110 1100 1100 1100
12	1110 1100 1110 1100 1110 1100 1110 1100
13	1110 1110 1110 1100 1110 1100 1110 1100
14	1110 1110 1110 1100 1110 1110 1110 1100
15	1110 1110 1110 1110 1110 1110 1110 1100
16	1110 1110 1110 1110 1110 1110 1110 1110
17	1111 1110 1110 1110 1110 1110 1110 1110
18	1111 1110 1110 1110 1111 1110 1110 1110
19	1111 1110 1111 1110 1111 1110 1110 1110
20	1111 1110 1111 1110 1111 1111 1110 1110
21	1111 1111 1111 1110 1111 1111 1110 1110
22	1111 1111 1111 1110 1111 1111 1111 1110
23	1111 1111 1111 1111 1111 1111 1111 1110
24	1111 1111 1111 1111 1111 1111 1111 1111

표 1-2. 변형된 puncturing vector 테이블

PI	V_{PI}							
1	1	0	0	0	0	0	0	0
2	1	0	0	0	1	0	0	0
3	1	0	1	0	1	0	0	0
4	1	0	1	0	1	0	1	0
5	1	1	1	0	1	0	1	0
6	1	1	1	0	1	1	1	0
7	1	1	1	1	1	1	1	0
8	1	1	1	1	1	1	1	1
9	2	1	1	1	1	1	1	1
10	2	1	1	1	2	1	1	1
11	2	1	2	1	2	1	1	1
12	2	1	2	1	2	1	2	1
13	2	2	2	1	2	1	2	1
14	2	2	2	1	2	2	2	1
15	2	2	2	2	2	2	2	1
16	2	2	2	2	2	2	2	2
17	3	2	2	2	2	2	2	2
18	3	2	2	2	3	2	2	2
19	3	2	3	2	3	2	2	2
20	3	2	3	2	3	2	3	2
21	3	3	3	2	3	2	3	2
22	3	3	3	2	3	3	3	2
23	3	3	3	3	3	3	3	2
24	3	3	3	3	3	3	3	3

표 1-3. Hardwired 로직 구현에 맞게 재배치한 puncturing 벡터 테이블

PI	V_{PI}							
1	1	0	0	0	0	0	0	0
2	1	1	0	0	0	0	0	0
3	1	1	1	0	0	0	0	0
4	1	1	1	1	0	0	0	0
5	1	1	1	1	1	0	0	0
6	1	1	1	1	1	1	0	0
7	1	1	1	1	1	1	1	0
8	1	1	1	1	1	1	1	1
9	2	1	1	1	1	1	1	1
10	2	2	1	1	1	1	1	1
11	2	2	2	1	1	1	1	1
12	2	2	2	2	1	1	1	1
13	2	2	2	2	2	1	1	1
14	2	2	2	2	2	2	1	1
15	2	2	2	2	2	2	2	1
16	2	2	2	2	2	2	2	2
17	3	2	2	2	2	2	2	2
18	3	3	2	2	2	2	2	2
19	3	3	3	2	2	2	2	2
20	3	3	3	3	2	2	2	2
21	3	3	3	3	3	2	2	2
22	3	3	3	3	3	3	2	2
23	3	3	3	3	3	3	3	2
24	3	3	3	3	3	3	3	3

SST 방식을 위한 pre-decoder는 puncturing 이전의 Mother 코드의 MSB를 복원하는 방식으로 쉽게 구현할 수 있다. 일반적으로 puncturing 과정에는 Mother 코드의 최상위 비트가 포함되지 않으므로 puncturing에 상관없이 복원할 수 있는 Mother 코드의 비트는 MSB이다. 아래 수식 (1)은 컨벌루션 인코더의 역 논리 회로로써 MSB a_i를 구하는 과정을 나타낸다. 이때 i는 시간을 나타내는 인덱스이고, $x_{0,i}$은 수신된 코드의 MSB를 나타낸다.

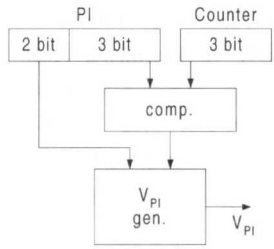


그림 2. Hardwired 로직으로 구현된 puncturing 벡터 테이블

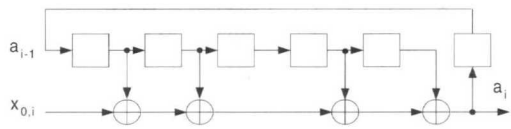


그림 3. 전처리단의 pre-decoder 구조

$$a_i = (a_{i-2} \oplus a_{i-3} \oplus a_{i-5} \oplus a_{i-6}) \oplus x_{0,1}$$
$$= (a_{i-2} \oplus a_{i-3} \oplus a_{i-5} \oplus a_{i-6}) \oplus (a_i \oplus a_{i-2} \oplus a_{i-3} \oplus a_{i-5} \oplus a_{i-6}) \quad (1)$$

위 수식 (1)은 shift 레지스터로 쉽게 구성할 수 있으며, 설계된 pre-decoder의 구조는 그림 3과 같다.

1.2 BMU 블록

연산모듈 1에서 가장 많은 연산량을 차지하는 부분은 BMU 블록이다. BMU 블록은 수신된 컨벌루션 코드와 가장 유사한 Trellis 다이어그램 내의 브랜치를 구하기 위해서 브랜치 메트릭을 계산한다. 이때 상태의 개수가 총 64개이므로 이론적으로 128번의 브랜치 메트릭을 계산해야 한다. 그러나 DAB 시스템의 컨벌루션 코드의 특성상 서로 다른 8개의 코드만이 발생 가능하므로, 실제로 128개의 브랜치

메트릭은 8개의 브랜치 메트릭 중의 하나이다[1].

이러한 성질을 이용하여, 본 논문에서는 8개의 브랜치 메트릭을 미리 계산해 놓은 다음, 128번의 브랜치 메트릭 계산을 실제로 수행하지 않고 미리 계산된 8개의 브랜치 메트릭으로부터 구하는 고속 알고리듬을 제안하였다. 제안된 알고리듬에 의해 브랜치 메트릭을 계산하면 연산량을 정확히 1/16로 감소시킬 수 있다. 이때 8개의 브랜치 메트릭은 입력 컨벌루션 코드에 따라 달라지며 모든 입력 코드에 대한 경우의 브랜치 메트릭을 미리 계산하는 것도 가능하지만, Viterbi 디코더의 성능향상을 위해 채택한 4 비트 연판정(soft decision) 방식을 고려할 때 총 216개의 경우가 존재하므로 너무 많은 메모리 공간을 요구하게 된다.

제안된 브랜치 메트릭 고속 연산 알고리듬에 따라 입력 컨벌루션에 대해서 미리 계산해야 할 8개의 브랜치 메트릭은 다음과 같다. 입력 컨벌루션 코드를 $(x_A x_B x_C x_D)$, 비교해야 할 브랜치 값을 $(ABCD)$ 라고 하면, 브랜치 메트릭 BM_{ABCD}는 다음과 같이 계산할 수 있다.

$$BM_{ABCD} = |x_A - A| + |x_B - B| + |x_C - C| + |x_D - D| \quad (2)$$

이때 BM_{abcd}를 계산하는데 절대값 4번과 덧셈 3번이 필요하고, 8개의 BM을 모두 계산하기 위해서는 절대값 32번과 덧셈 24번을 수행해야 한다. 그러나 식 (2)의 각 절대값 항은 서로 다른 4개의 BM을 계산하는데 반복해서 사용되고, 절대값 2개의 합은 서로 다른 2개의 BM을 계산하는데 반복해서 사용되므로, 이러한 성질을 이용하면 전체 BM을 계산하는데 필요한 연산량을 줄일 수 있다. 그림 4는 불필요한 반복연산을 피하고 효율적으로 8개의 BM을 계산하는 과정을 보여주고 있다.

첫째 단의 d_{ij}에서 i는 입력 코드 $(x_A x_B x_C x_D)$의 각 비트를 나타내는 인덱스이고, j는 브랜치 값 $(ABCD)$ 각 비트가 가질 수 있는 값 '0' 또는 '1'을 가리킨다. 즉 d_{A0}는 x_A와 비트 '0' 사이의 거리 $|x_A - 0|$을 나타낸다. 따라서 첫째 단은 식 (2)에서 가능한 조합의 모든 절대값을 구한 것이다. 둘째 단은 화살표로 이어진 첫째 단 값들의 합을 나타내며, 식 (2)에서 두 개의 절대값을 합한 것과 같다. 마지막 셋째 단은 최종적인 BM_{ABCD} 값들이며, 그림 4에서 보듯이 A=D 이므로 총 8개의 조합이 가능하다.

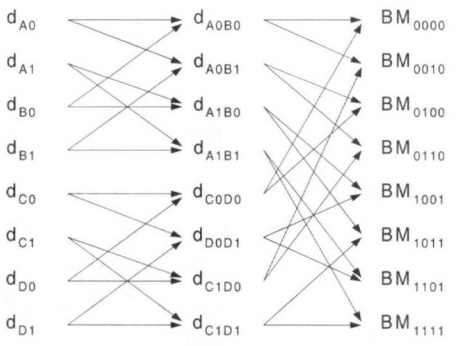


그림 4. Branch Metric을 계산하는 제안된 고속 알고리듬

그림 4의 알고리듬의 사용하여 총 8개의 BM을 계산하는데 필요한 연산은 8번의 절대값과 16번의 덧셈만이 필요하게 된다. 이것은 식 (2)를 그대로 계산할 경우보다 절반이하의 연산량만이 필요함을 알 수 있다.

2. 연산모듈 2 : ACSU + PMMU

2.1 ACSU 블록

ACSU 블록은 총 64개의 상태에 대해서 2가지 경로의 브랜치 메트릭과 패스 메트릭의 합을 비교하여 최종적인 생존 경로를 결정한다. 이 과정은 파이프라인 병렬처리를 위해 분할된 다른 연산 모듈들에 비해 상대적으로 많은 연산량을 차지한다. 이에 본 논문에서는 2개의 ACS 모듈을 할당함으로써 고속연산이 가능하도록 설계하였다.

나비(butterfly) 모양으로 표현한 트렐리스도와 각 상태마다 최종적인 패스 메트릭을 구하고 최종적인 생존경로를 선택하는 ACS 과정을 그림 5와 식 3에 각각 나타내었다.

그림 5에서 $S_{j,t}$는 t 시간의 j 번째 상태를 의미하고, bm은 브렌치 메트릭을 나타낸다. 식 (3)에서 $PM_t(S_j)$는 상태 $S_{j,t}$에서의 패스 메트릭을 가리킨다. 그림 5에서 bm1=bm4, bm2=bm3 이므로 고속연산을 위해서는 그림 5의 나비구조에 따른 패스 메트릭 갱신을 동시에 수행하도록 하는 것이 효율적이다.

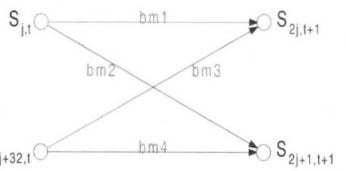


그림 5. 나비 모양의 트렐리스도

$$PM_{t+1}(S_{2j}) = \min\,(\,PM_t(S_j) + bm_1\,,\; PM_t(S_{j+32}) + bm_3\,)$$
$$PM_{t+1}(S_{2j+1}) = \min\,(\,PM_t(S_j) + bm_2\,,\; PM_t(S_{j+32}) + bm_4\,) \quad (3)$$

설계된 ACSU 블록의 구조를 그림 6에 나타내었다. 그림과 같이 2개의 ACS 모듈을 배치하여 식 (3)을 병렬로 연산하도록 하였으며, ACSU 블록의 최종적인 출력은 생존경로(spi)와 각 상태마다의 패스 메트릭(pm) 값들이다. Buffer는 하나의 메모리에 차례로 쓰기 위한 임시 저장장소로 사용된다.

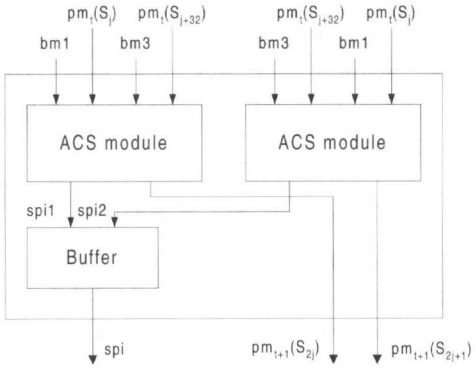


그림 6. ACSU 블록의 구조

2.2 PMMU 블록

PMMU 블록은 최종적으로 갱신된 그림 6의 패스 메트릭(pm) 값을 저장한다. 패스 메트릭을 저장하는 가장 간단한 방법은 ping-pong 방식으로서, 두 개의 메모리를 이용하여 번갈아가며 이전 메트릭 값을 불러오고 새로운 메트릭 값을 저장하는 공간으로 사용한다. 메모리 사이즈를 줄일 수 있는 방법으로서 in-place scheduling 방식이 제안되었으며, 약간의 부가적인 로직을 이용하여 필요한 메모리의 양을 절반으로 줄일 수 있다[3]. 본 논문에서도 시스템의 설계 면적을 줄이기 위해서 in-place scheduling 방식을 채택하였다.

PMMU 블록을 설계할 때 고려해야할 또 하나의 사항은 패스 메트릭의 re-scaling에 관한 것이다. 패스 메트릭의 값은 시간이 지남에 따라 점차 값이 증가하므로, 오버플로우(overflow)가 발생하지 않도록 하려면 적절한 re-scaling이 필요하다. Re-scaling은 또한 패스 메트릭 메모리의 워드길이를 결정하는 요인이 된다. 간단한 re-scaling 방식은 패스 메트릭을 갱신할 때마다 최소 패스 메트릭을 구하여 모든 패스 메트릭으로부터 빼주는 방법이 있다. 그러나 이러한 방법은 연산량을 많이 요구하므로 고속연산을 수행해야 하는 Viterbi 디코더의 설계에는 부적합하다.

본 연구에서는 패스 메트릭의 최대값과 최소값의 차이가 시간에 관계없이 bound 된다는 사실[5]을 이용하여 연산량을 감소시킨 re-scaling 방식을 제안하였다. 패스 메트릭의 최대값과 최소값의 차이 $R_{\max}$는 식 (4)와 같이 bound된다[5]. 이때 N은 상태의 개수이고 $\lambda_{\max}$는 트렐리스에서 브랜치 메트릭의 최대값이다.

$$R_{\max} \le \lambda_{\max} \cdot \log_2 N \quad (4)$$

설계된 Viterbi 디코더는 구속장 7이므로 N=64이고, 코드율 1/4에 4 비트 연판정 방식을 사용하므로 $\lambda_{\max}$ 값은 4×15=60이 된다. 그러나 트렐리스도에서 선택되는 브랜치 메트릭의 최대값은 컨벌루션 코드의 최대 거리 인코딩 성질로 인해 $\lambda_{\max}/2$ = 30이 된다. 따라서 $R_{\max}$ 값은 180에 의해서 bound되며, 이 범위의 값을 표현하는데는 8 비트가 필요하다.

본 논문에서는 패스 메트릭을 표현하는데 9비트를 사용했으며, 모든 패스 메트릭의 MSB가 1이 되었을 때 2^8 값을 빼주는 re-scaling 방식을 제안하였다. 패스 메트릭을 표현하는데 9비트를 사용한 이유는 모든 패스 메트릭의 MSB가 1이 되는 조건에서도 나머지 8 비트에 의해서 패스 메트릭을 표현하는데 오버플로우가 생기지 않게 하기 위함이다.

모든 MSB가 1이 되는 조건은 AND 로직으로 간단하게 구현할 수 있으며, 2^9를 빼주는 연산은 MSB를 단순히 '0'으로 반전시키면 된다. 본 논문에서는 이러한 re-scaling 방식을 적용하기 위해서 패스 메트릭을 저장하는 공간으로서 64개의 레지스터 파일을 사용하였으며, 제안된 re-scaling 방식은 다음과 같은 장점을 가진다.

- 매번 최소 메트릭을 빼주는 방식에 비해서 연산량을 크게 감소시킨다.
- 패스 메트릭을 표현하는데 필요한 워드길이를 최적화시킨다.

3. 연산모듈 3 : SMU

SMU 블록은 매 상태마다 결정된 생존자 경로를 이용하여 최종적인 생존자 경로를 찾아내 전송된 정보를 복원해낸다. 생존자 경로를 저장하는 방식으

로 레지스터 교환(register exchange) 방식과 역추적(trace-back) 방식이 사용되는데, 본 논문에서는 상대적으로 전력소비가 적고 설계 면적이 작은 one-pointer 알고리듬을 이용한 역추적 방식[6]을 채택하였다.

설계된 Viterbi 디코더는 시스템의 동작 주파수를 고려할 때, 1 비트를 출력하기 위해 32 사이클을 사용할 수 있다 (3-2절 참조). 따라서 one-pointer 알고리듬에 의해 32개의 메모리 뱅크가 요구되며, 각 뱅크내 section의 개수인 (역추적 깊이/30)이 정수값이 되도록 역추적 깊이를 60으로 결정하였다[6]. 결정된 역추적 값은 구속장의 5~6배에 해당하므로, Forney에 의하면 모든 생존자 패스가 한곳으로 수렴하는데 충분한 값이다[8].

표 2는 설계된 SMU 블록이 실시간 처리를 위하여 메모리를 제어하는 방식을 나타내고 있다. 메모리에 대한 연산은 역추적 과정 (Trace Back), 쓰기 (Write), 디코딩 (Decoding)으로 이루어져 있으며, t0 ~ t63은 각각 32 사이클로 이루어진 시간간격이다. 표 2에서 $BK_i(S_j)$는 i번째 메모리 뱅크의 j번째 section을 의미하고, 화살표는 가리키는 방향에 따라 차례대로 메모리가 엑세스됨을 나타낸다.

메모리에 대한 동작은 크게 두가지로 분류되며 표 2에서 보듯이 검게 칠해진 부분과 아닌 부분으로 나뉘어 교대로 일어난다. 검게 칠해진 시간에서는 31번의 역추적과 1번의 쓰기 과정이 수행되고, 칠해지지 않은 부분에서는 29번의 역추적과 1번의 쓰기, 2번의 디코딩 과정이 수행된다. 따라서 2개의 시간간격 t_i ~ t_{i+1} 동안을 살펴보면 총 역추적 깊이는 60이 되고, 2개의 생존자 경로가 갱신되며 2비트의 정보가 복원된다. 표 2를 보면 순환방식(circular addressing)에 의해 메모리에 대한 엑세스가 이루어지며, 64번의 주기를 갖고 표 2의 과정들이 반복됨을 알 수 있다.

표 2. one-pointer 알고리듬을 사용한 survivor memory

access \ time	t0	t1	t2
Trace Back	BK31→17 BK16(S1)	BK16(S0) BK15→2	BK0→18 BK17(S1)
Write	BK0(S1)	BK0(S0)	BK1(S1)
Decoding		BK1	

t3	…	t62	t63
BK17(S0) BK16→3	…	BK30→16 BK15(S1)	BK15(S0) BK14→1
BK1(S0)	…	BK31(S1)	BK31(S0)
BK2	…		BK0

설계된 Viterbi 디코더는 전력소모를 줄이기 위해서 SST 방식에 채택하였으므로, 역추적 과정의 시작점으로 가장 작은 패스 메트릭을 갖는 상태를 찾아야하는 MLD (Maximum Likelihood Decision)회로를 사용하지 않아도 성능에 큰 영향을 주지 않는다. 이것은 다른 연산모듈과의 pipeline 처리를 위해서 SMU 블록의 전체 연산량을 크게 감소시키는 역할을 한다.

Ⅲ. 모의 실험 및 결과

1. 실시간 구현 검증

설계된 Viterbi 디코더는 Eureka DAB 시스템이 요구하는 채널 디코더의 최대 출력 데이터율인 1.824 Mbps를 지원한다. DAB 시스템에서 한 개의 전송 프레임 안에 하나의 서브채널이 전송할 수 있는 최대 비트 수는 43,776 비트이다[1]. DAB 프레임의 전송 시간은 24 ms이므로, 65.436 MHz의 동작 주파수를 사용했을 때 Viterbi 디코더 출력 1 비트를 계산하는데 사용 가능한 사이클 수는 약 35.875 사이클 정도이다.

설계된 Viterbi 디코더는 세 개의 연산모듈로 나누어져 있으며, 각 연산모듈은 pipeline 방식의 병렬처리를 하기 때문에, 실시간 동작을 위해서는 모든 연산블록이 35 사이클 안에 필요한 연산을 수행해야 한다. 표 3은 각 연산모듈의 연산을 수행하는데 소요하는 사이클 수를 나타낸다. 표 3에서 보듯이 각 연산모듈은 최대 32 동작 사이클 안에 모든 연산을 수행하므로, 설계된 Viterbi 디코더는 실시간으로 디

표 3. 비트 출력을 위한 Viterbi 디코더 블록별 연산량

연산모듈	세부 연산블록	소요 사이클 수
연산모듈 1	전처리단, BMU 블록	28
연산모듈 2	ACSU 블록, PMMU 블록	32
연산모듈 3	SMU 블록	32

코딩 과정을 수행함을 알 수 있다. 또한 파이프라인의 효율을 높일 수 있도록 전체 연산량이 각 연산모듈에 균등하게 분배되어 있음을 확인할 수 있다.

2. 상태천이 빈도

Viterbi 디코더의 상태천이 빈도는 전력소모와 밀접한 관계가 있다. 즉, 상태천이 빈도가 낮을수록 레지스터의 값이 바뀌는 경우가 적어지며, 결과적으로 소비하는 전력이 감소된다. 그림 7은 본 논문에서 채택한 SST 방식의 Viterbi 디코더와 일반적인 Viterbi 디코더의 채널 오류 확률에 따른 상태천이 빈도를 나타내고 있다. 모의실험은 코드율 1/2로 puncturing된 200000 비트에 $10^{-0.3}$부터 $10^{-1.3}$까지의 채널 오류를 더하여 60의 역추적 길이를 갖는 Viterbi 디코더 모델을 적용하여 수행하였다. 그림 7에서 가로축은 채널 오류확률을, 세로축은 상태천이 빈도를 나타내고, 상태 천이 빈도는 역추적 동작에서 가능한 천이 빈도의 최대 값으로 정규화하여 표현하였다.

그림 7에서 보듯이 SST 방식의 Viterbi 디코더는 채널오류 환경이 양호한 채널 상황에서 일반적인 Viterbi 디코더에 비해 상태의 천이가 크게 줄어드는데, $10^{-1.3}$의 채널 오류에 대해서는 일반적인 Viterbi 디코더에 비해 -40dB 만큼 감소함을 볼 수 있다. 이것은 제안된 Viterbi 디코더가 상대적으로 일반적인 Viterbi 디코더에 비해 전력을 적게 소모함을 의미한다.

3. 물리적 평가

제안된 Viterbi 디코더는 설계면적을 줄이기 위해서 puncturing vector table을 ROM이 아닌 hardwired logic으로 구현하였으며, in-place scheduling 방식을 채택하여 PMMU 모듈의 메모리량을 줄였다. 또한 제안한 re-scaling 방식을 사용하면 패스 메트릭을 저장하는데 필요한 워드길이를 최소화시킬 수 있다. 실제로 Shieh가 제안한 시스템[7]에서 Viterbi 디코더가 사용하는 패스 메트릭 메모리의 워드 길이는 14 비트인데 반해, 설계된 Viterbi 디코더는 9 비트 워드길이의 패스 메트릭 메모리를 사용한다.

설계된 Viterbi 디코더는 VHDL로 기술하여 Synopsys tool을 사용하여 기능적 검증을 수행하였으며, 삼성 0.35 μ standard cell 라이브러리를 사용하여 합성하였다. 표 4는 합성된 Viterbi 디코더의 합성결과를 각 연산모듈 별 게이트 수와 지연시간

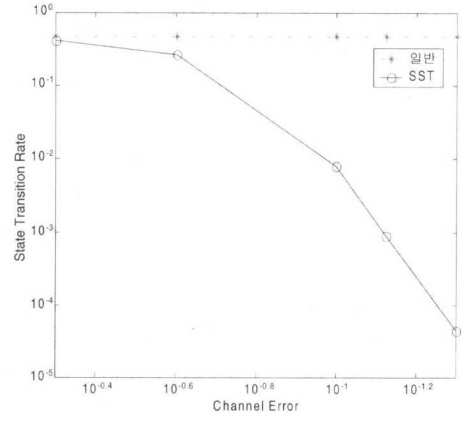


그림 7. 역추적에서의 상태 천이 빈도 비교

으로 나타내고 있다. 표 4에서 보듯이 2개의 ACSU 모듈을 포함하고 있는 연산모듈 2가 가장 많은 면적을 차지하며, 전체 시스템은 약 13,000 게이트 정도를 차지한다. 또한 시스템의 최대 지연시간은 11.02 ns로서 30%의 설계 margin을 고려할 때 디코더의 최대 동작 주파수는 약 69.6 MHz가 된다.

표 4. 설계된 Viterbi 디코더의 합성결과

연산모듈	게이트 수	지연시간 (ns)
연산모듈 1	1,992.25	11.02
연산모듈 2	10,554.75	10.49
연산모듈 3	1,119.50	11.02

Ⅳ. 결 론

Eureka DAB 시스템이 지원하는 최대 출력 데이터율을 만족시키는 Viterbi 디코더를 설계하였다. 최대 1.8Mbps 정도의 출력 데이터율을 만족시키려면 방대한 연산량을 수행해야 하며, 실시간 동작을 고려할 때 많은 하드웨어 자원을 이용하여 고속으로 동작해야 한다. 그러나 DAB 수신기 시스템의 상용성을 고려할 때 설계 면적과 전력 소비를 줄이는 것이 요구된다.

본 논문에서는 설계면적과 전력소모 측면에서 효율적인 Viterbi 디코더 구조를 제안하였다. 고속동작을 위해 전체 시스템을 연산량이 비슷한 3개의 연산모듈로 분할하여 pipeline 방식으로 병렬 연산을

수행하도록 설계하였다. 세 개의 연산모듈은 각각 전처리단과 BMU 블록, ACSU 블록과 PMMU, 그리고 SMU 블록을 수행하도록 분할되었다.

설계면적을 줄이기 위해 전처리단과 BMU 블록이 공통으로 사용하는 puncturing vector 테이블을 수정·재배치하여 간단한 hardwired logic으로 구현하였으며, PMMU 블록이 하나의 패스 메트릭 메모리만을 사용하도록 in-place scheduling 방식을 채택하였다. 또한 패스 메트릭의 최대값과 최소값이 bound 되는 사실을 이용하여 패스 메트릭 메모리의 워드길이를 최적화함으로써 패스 메트릭 메모리의 크기를 줄였다.

설계된 Viterbi 디코더는 SST 방식을 사용하여 불필요한 상태천이를 제거함으로써 소모전력을 감소시켰다. 미리 계산된 브랜치 메트릭을 이용하는 고속 알고리듬은 BMU 블록의 연산량을 크게 줄이며, 패스 메트릭의 모든 MSB가 1이 되는 조건에서 re-scaling을 수행하도록 제안한 re-scaling 방식은 패스 메트릭을 re-scaling 하는데 필요한 연산량을 크게 감소시킨다.

제안된 Viterbi 디코더는 Shieh가 제안한 채널 디코더에 비해 상대적으로 적은 연산량과 소모 전력으로 full data-rate 조건을 만족하도록 설계되었다. 이러한 구조는 앞으로 작은 하드웨어 면적 및 적은 소비전력이 요구되는 DAB 수신기 시스템과 이동통신 시스템에서 유용하게 사용될 것이다.

참 고 문 헌

[1] ETS 300 401, Radio Broadcasting Systems; Digital Audio Broadcasting to mobile portable and fixed receivers, 2nd Ed., *ETSI*, May 1997

[2] S.Kubota, K.Ohtani and S.Kato, "High-speed and high-coding-gain Viterbi decoder with low power consumption employing SST (Scarce State Transition) scheme," *Electron. Lett.*, Vol 22, Apr. 24, 1986

[3] M. Biver, H. Kaeslin, "In-Place Updating of Path Metrics in Viterbi Decoder," *IEEE Journal of Solid-State Circuits*, Vol. 24, No. 4, pp. 1158-1160, Aug. 1989

[4] Andries P. Hekstra, "An Alternative to Metric Rescaling in Viterbi Decoders", *IEEE Trans. Communications*, Vol. 37, No. 11, pp. 1220-1222, Nov. 1989

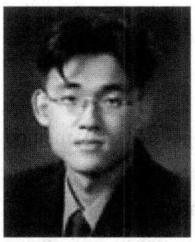
[5] B. Shim and S. Cho, "A new metric rescaling method for pipelined Viterbi decoder," in *ITC-CSCC99*, Vol. 1, pp. 122-125, July 1999

[6] G. Feygin and P. G. Gulak, "Architectural Tradeoffs for Survivor Sequence Memory Management in Viterbi Decoders," *IEEE Trans. Comm.*, Vol. 41, No. 3, pp. 425-429, Mar. 1993

[7] M.D. Shieh, C.M. Wu, H.H. Chou, M.H. Chen and C.L. Liu, "Design and Implementation of a DAB Channel Decoder", *IEEE Trans. Consumer Electronics*, Vol. 45, No. 3, pp. 553-562, Aug. 1999

[8] G. D. Forney, "The Viterbi Algorithm," *Proc. IEEE*, Vol. 61, pp. 268-278, Mar. 1973

김 효 원(Hyo-Won Kim) 준회원



1999년 2월 : 연세대학교 전자공학과 졸업
2001년 2월 : 연세대학교 전기전자공학과 석사
2001년 3월~현재 : ETRI 부설 국가보안기술연구소

<주관심 분야> 통신신호처리, 이동통신, 시스템 구현, 암호학

고 우 석(Woo-Suk Ko) 준회원

한국통신학회 논문지 제26권, 제12C호 참조

류 주 현(Joo-Hyun Ryu) 준회원

1998년 8월 : 연세대학교 전자공학과 졸업
2000년 8월 : 연세대학교 전기·컴퓨터공학과 졸업
2000년 9월~현재 : LG전자

<주관심 분야> 통신신호처리 및 시스템 구현

윤 대 희(Dae-Hee Youn) 정회원

한국통신학회 논문지 제26권, 제12C호 참조