IEEE 802.11a 무선 랜에 적용할 Low Latency 인터리버 설계

신 보 영*, 이 종 훈, 박 준, 원 동 윤, 송 상 섭
전북대학교 전자공학과
e-mail : young@codelab.chonbuk.ac.kr sindelella@hotmail.com

An interleaver design of low latency for IEEE 802.11a Wireless LAN

Bo young Shin*, Jong hoon Lee, June Park, Dong youn Won, Sang seob Song
Department of Electronics Engineering, Chonbuk National University

Abstract

By minimizing the burst error of data and correcting the error, we can define the convolution coding and interleaving in IEEE 802.11a wireless lan system. Two step block interleaver was decided by coded bits per OFDM symbol and due to this it comes to the delay time in IEEE 802.11a. This is the point of the question which we must consider. We try to decrease the delay time by all 48-clock from interleavings, and we have proposed a way carried out the interleaving outputs per symbol. So in comparison with the existing interleaver, we can decrease the delay time in reading and writing data, as well as reduce the delay time of bit re-ordering per symbol. Also this scheme is apply in all x-QAM cases.

I. 서 론

최근 공중망 무선 랜 사업 서비스의 확대는 초고속 무선 랜 서비스 방식을 5-GHz 주파수 대역에서 54Mbps로 데이터 전송이 가능한 IEEE 802.11a 방식으로 추진되고 있다. 이제 무선 랜의 이용은 단순 사무용이 아닌 초고속 통신을 위한 하나의 매개체이며, 다양한 서비스 애플리케이션이 개발되고 있다. 이러한 이유는 무선 랜이 가지는 경제적인 측면과 이용의 효율성에 있다.

또한 802.11b의 최대 처리율이 11Mbps 인데 반해, 802.11a의 최대 처리율은 54Mbps이다. 뿐만 아니라 802.11a 기반 제품은 보다 안전하고 설치 및 관리가 용이하다는 장점을 갖는다. 그리고 보다 향상된 무선 네트워크의 성능과 특성을 보이며, 802.11b 기반 제품보다 약 5배 정도 빠르다. 이와 같은 처리율(throughput) 향상은 사용자의 생산성 향상은 물론, 무선 랜을 비디오나 스트림 데이터 처리 응용에 사용하는 것을 가능하게 한다. 이러한 이유들로 현재 세계 무선 랜의 시장은 많은 잠재력을 가지고 성장하고 있다. [3]

전송에 따라 데이터에 오류가 발생하면 복구가 불가능하게 되어 데이터를 이용할 수 없는 경우가 발생하는데 이를 최소화하기 위해 오류 수정 기술을 사용하며, IEEE 802.11a 무선 랜에서는 이러한 기술로 컨볼루션 코딩과 인터리버를 정의하고 있다. 컨볼루션 인코더를 통해 코딩된 데이터는 전송 속도에 따라서 심볼 단위로 매핑 되어 IFFT 과정을 수행하는데, 인터리버는 그 과정에서 한 곳에 연속적으로 존재하는 오류 데이터의 위치를 일정한 규칙에 따라 비연속적인 위치로 바꾸어 놓음으로써 오류 수정 능력을 높인다.

아울러 무선 랜의 처리량에 영향을 미치는 요인으로는 이용자 수, 범위와 멀티패스 등의 채널 요인, 이용하고 있는 무선 랜 시스템의 형태, 그리고 랜 연결 지점에서의 지연과 병목 등이 있는데, 본 논문에서는 이러한 인터리빙 과정에서 생기는 지연 시간을 줄이는 방법을 제시한다. 즉, 인터리빙에 사용되는 메모리를 몇 개의 작은 단위로 나누어 사용하며, IFFT 처리를 위해 직접 심볼 단위로 읽어 들이는 방법을 제안한다.

II. IEEE 802.11a 기술

IEEE 802.11a는 OFDM(Orthogonal Frequency Division Multiplexing) 방식을 사용하는 것을 가장 큰 특징으로 한다. OFDM은 넓은 대역의 단일 반송파 대신 상호 중첩된 좁은 대역의 여러 부반송파를 병렬로 보내는 다중 반송파 변조 방식으로, 매우 큰 ISI (Inter-Symbol-Interference)를 갖는 주파수 선택적 페이딩 채널에서도 좁은 대역의 각 부채널은 flat-fading 특성을 갖는다는 사실에 기반한 방식이다. OFDM 모뎀부는 컨볼루셔널 부호기/비터비 복호기, 인터리버/디인터리버, 변/복조부, IFFT/FFT부, 동기부 등으로 구성된다. 일반적으로 컨볼루셔널 부호화기, 인터리버/디인터리버, 변복조부 등의 블록은 전송 속도에 따라 각각 다른 파라미터로 설정되고, IFFT/FFT부와 동기부의 성능은 전송 속도에 관계없이 일정한 파라미터로

설정된다.

다음 [표 1]은 가변 전송율 제공을 위하여 각 모드에서 사용되는 변조 방식과 부호율 등의 파라미터를 정리한 것이다. [1]

전송 속도 (Mbits/s)	변 조	Coding rate(R)	N_{BPSC}	N_{CBPS}	N_{DBPS}
6	BPSK	1/2	1	48	24
9	BPSK	3/4	1	48	36
12	QPSK	1/2	2	96	48
18	QPSK	3/4	2	96	72
24	16-QAM	1/2	4	192	96
36	16-QAM	3/4	4	192	144
48	64-QAM	2/3	6	288	192
54	64-QAM	3/4	6	288	216

- N_{BPSC} : Coded bits per subcarrier
- N_{CBPS} : Coded bits per OFDM symbol
- N_{DBPS} : Data bits per OFDM symbol

[표 1] Rate-dependent parameters

IEEE 802.11a 무선 랜에서 사용되는 인터리버의 규격은 N_{CBPS}와 N_{BPSC}의 서로 다른 4쌍에 의해서 총 4가지로 분류된다. 구체적으로 이러한 인터리버는 첫 번째 서로 인접한 코딩된 데이터들을 인접하지 않은 부반송파에 매핑하고, 두 번째 심볼 매핑 성상도에서의 비트 위치를 바꾸는 두 단계의 순열 치환을 이용하여 데이터의 순서를 바꾸어 놓는다.

다음 식들은 위에서 설명한 두 단계 과정을 나타낸다. [1]

• First permutation

$$i = (N_{CBPS}/16)(k \bmod 16) + floor(k/16)$$
$$k = 0, 1, \ldots N_{CBPS} - 1$$

• Second permutation

$$j = s \times floor(i/s) + (i + N_{CBPS} - floor(16 \times i/N_{CBPS})) mod\ s$$
$$i = 0, 1, \ldots N_{CBPS} - 1 \qquad s = max(N_{BPSC}/2, 1)$$

한편, 디인터리버는 주소를 생성하는 과정에서 메모리 write 시퀀스와 read 시퀀스가 인터리버와 반대가 될 뿐 동일한 구조를 갖는다.

III. 인터리버의 메모리 주소 생성

1. 메모리 주소 발생

인터리버에 사용되는 메모리는 가로 열의 크기는 고정되어 있고, 세로 행의 크기가 N_{CBPS}에 의해서 변화한다. 따라서 메모리에 데이터를 쓰는 과정에 필요한 주소는 mod-16 카운터와 mod-3, mod-6, mod-12, mod-18 카운터의 조합으로 발생시킬 수 있다.

가로 열의 주소 발생을 위한 mod-16 카운터는 4-비트로 구성할 수 있고, 세로 행의 주소 발생을 위한 카운터들은 5-비트($2^4 < 18 < 2^5$)로 구성할 수 있다. [4]

2. First permutation 과정

블록 인터리버에서 First permutation을 위한 주소 발생은 매우 단순하다. First permutation의 과정을 살펴보면, 가로 열이 16으로 고정된 메모리의 데이터를 세로 방향으로 읽어오는 과정에 불과하다. 물론 메모리의 세로 행의 크기는 앞서 말한 N_{CBPS}에 따라서 4가지 종류(3,6,12,18)가 존재한다.

그 중에서 $N_{CBPS} = 48$의 경우를 생각해 보자. First permutation 과정을 통해서, 입력되는 k-번째 데이터는 i-번째 데이터로 그 위치를 옮기게 된다.

여기에서 k는 메모리의 번지수이자 입력되는 데이터의 순서이며 i는 메모리의 번지수라는 점에 착안하여, k-번째 데이터는 일정한 규칙에 따라 메모리의 i-번 주소로 그 위치가 변화하는 것을 알 수 있다. 이처럼 순차적으로 입력되는 연속된 비트들은 $N_{CBPS}/16$의 크기만큼 서로 떨어진 위치로 분산된다. 결국, first permutation은 단순히 데이터를 메모리의 가로방향으로 순차적으로 쓰고, 세로방향으로 읽어나가는 방법으로 구현 가능하다. 그리고 $N_{BPSC} = 1\,(s = 1)$ 또는 $N_{BPSC} = 2\,(s = 1)$의 경우는 first permutation만으로 인터리빙이 모두 끝나게 된다.

간단히 $N_{CBPS} = 48$의 경우를 예로 살펴보자.

$$i = (N_{CBPS}/16)(k \bmod 16) + floor(k/16)$$
$$k = 0, 1, \ldots N_{CBPS} - 1$$

$N_{CBPS}/16$	3	3	3	...	3	3	3	3	...	3	3	3	3	...	3	3
$k \bmod 16$	0	1	2	...	14	15	0	1	...	14	15	0	1	...	14	15
$floor(k/16)$	0	0	0	...	0	0	1	1	...	1	1	2	2	...	2	2
k	0	1	2	...	14	15	16	17	...	30	31	32	33	...	46	47
i	0	3	6	...	42	45	1	4	...	43	46	2	5	...	44	47

[표 2] First permutation ($N_{CBPS} = 48$)

위 [표 2]에서 구해진 값에 따라, k-번째 들어오는 데이터는 i-번째에 읽혀지게 된다. 결국, $N_{CBPS} = 48$의 경우 다음과 같은 3×16의 메모리를 생각해 본다면, 다음 [표 3_a]와 같은 write 시퀀스에 따라, [표 3_b]의 과정을 통해 [표 3_c]와 같은 read 시퀀스를 얻어낼 수 있다. 아울러 [표 3_a]의 세로 행 방향으로 데이터를 읽어나간 결과가 [표 3_c]와 같다는 것을 통해, 블록 인터리빙의 형태를 확인할 수 있다.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47

[표 3_a] $N_{CBPS} = 48$ 의 write 시퀀스

0			1			2			3			4			5
		6			7			8			9			10	
	11			12			13			14			15		

[표 3_b] $N_{CBPS} = 48$ 의 first permutation 과정

0	16	32	1	17	32	2	18	34	3	19	35	4	20	36	5
21	37	6	22	38	7	23	39	8	24	40	9	25	41	10	26
42	11	27	43	12	28	44	13	29	45	14	30	46	15	31	47

[표 3_c] $N_{CBPS} = 48$ 의 first permutation 결과

3. Second permutation 과정

Second permutation의 과정은 k로부터 생성된 i주소를 이용해서 다시 주소 값 j를 생성하는 과정이라고

할 수 있다. $N_{BPSC}=1(s=1)$ 또는 $N_{BPSC}=2(s=1)$의 경우에는 Second permutation이 데이터의 순서에 전혀 영향을 미치지 않으나, $N_{BPSC}=4(s=2)$ 이거나 $N_{BPSC}=6(s=3)$인 경우에는 second permutation의 다음 과정을 통해 first permutation에서 생성한 주소 값에 약간의 변형을 가하게 된다.

$$j=s\times floor(i/s)+(i+N_{CBPS}-floor(16\times i/N_{CBPS}))mod\ s$$
$$i=0,1,\ldots N_{CBPS}-1 \qquad s=max(N_{BPSC}/2,1)$$

i	0	12	24	36	...	1	13	25	37	...	10	22	34	..	11
$floor\ (i/s)$	0	6	12	18	...	0	6	12	18	...	5	11	17	...	5
$s\times floor\ (i/s)$	0	12	24	36	...	0	12	24	36	...	10	22	34	...	10
$floor(16\times i/N_{CBPS})$	0	1	2	3	...	0	1	2	3	...	0	1	2	...	0
$(i+N_{CBPS}-floor(16\times i/N_{CBPS})$	192	203	212	225	...	193	204	215	226	...	202	213	224	...	203
mod 2	0	1	0	1	...	1	0	1	0	...	0	1	0	...	1
j	0	13	24	37	...	1	12	25	36	...	10	23	34	...	11

[표 4] Second permutation ($N_{BPSC}=4(s=2)$)

위 [표 4]에서 보이는 것처럼, second permutation의 과정을 거치게 되면, first permutation에서 발생한 주소 값에 대해서 주기 16을 기준으로 [0, +1] 또는 [0, -1]의 변화가 반복되는 결과를 얻는다. 또 이와 마찬가지로 $N_{BPSC}=6(s=3)$인 경우는 [0, +2, +1], [0, -1, +1] 또는 [0, -1, -2]의 변화를 반복하게 된다.

이 절의 내용을 정리해 보면, IEEE 802.11에서 규정한 인터리버는 두 단계의 순열 치환을 이용해 데이터의 순서를 섞어놓는 블록 인터리버이다. 또한 그 내부는 k → i → j로의 계산 과정을 보이는데, 이는 데이터 자체의 값이 아닌 주소 값의 변화이므로 각각의 주소에 따른 데이터들의 재 정렬이 곧 인터리빙 결과이다. 아래 [표 5]는 k-번째에 쓴 데이터가 최종 j-번째에 읽혀지며, k-번째에 읽혀지는 데이터는 각각의 표 끝 부분의 데이터 값이 됨을 보이고 있다.

BPSK		
k	i=j	data
0	0	0
1	3	16
2	6	32
3	9	1
4	12	17
5	15	33
6	18	2
7	21	18
8	24	34
9	27	3
10	30	19
. . .		

QPSK		
k	i=j	data
0	0	0
1	6	16
2	12	32
3	18	48
4	24	64
5	30	80
6	36	1
7	42	17
8	48	33
9	54	49
10	60	65
. . .		

16-QAM			
k	i	j	data
0	0	0	0
1	12	13	16
2	24	24	32
3	36	37	48
4	48	48	64
. . .			

64-QAM			
k	i	j	data
0	0	0	0
1	18	50	16
2	36	37	32
. . .			
18	37	38	17
19	55	55	33

12	144	144	17
13	156	157	1
14	168	168	49
15	180	181	33
16	1	1	81
. . .			

20	73	72	1
. . .			
36	74	73	34
37	92	90	2
38	110	110	18
. . .			

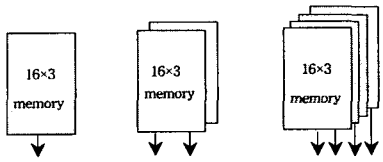
[표 5] x-QAM의 k → i→ j 매핑

Ⅳ. 메모리 분할에 의한 인터리버 설계

지금까지의 인터리빙 방법을 이용하면 각 데이터 전송 속도에 따라 심볼을 구성할 때 많은 지연이 발생하게 된다. 즉, n개의 비트로 한 개의 심볼을 구성할 경우 단위 심볼을 읽어오기 위해서 n번의 클럭이 필요하다. 그러나 클럭은 통신에서 매우 중요한 요소이고 이를 최소화하기 위한 방법이 요구되며, 다음에 설명하는 방법과 같이 분할된 메모리를 사용할 경우 한 개의 심볼은 한 개의 클럭으로 읽기가 가능할 수 있다.

예를 들어 16-QAM의 경우 16×3 메모리 4개를 써서 병렬 처리하면, 모든 전송 속도에 대하여 같은 속도로 동작시킬 수 있다. 16-QAM의 경우는 4-비트가 1-심볼로 구성되며 BPSK는 1-비트가 1-심볼, QPSK는 2-비트가 1-심볼 마지막으로 64-QAM의 경우는 6-비트가 1-심볼로 구성된다. 각 경우에 대하여 1-심볼에 해당하는 비트 수 만큼의 메모리를 사용함으로써 x-QAM 모두 48-클럭에 인터리빙을 마칠 수 있다.

16-QAM의 경우 4개의 메모리를 사용함으로써 4비트를 동시에 처리한다. (행)×(열)×(메모리 개수)=16×3×4=192 비트로 N_{CBPS}를 만족하는 값을 얻어낸다. 192-비트는 4-비트씩 메모리에 저장된다. 아래 [그림 1]은 메모리를 분할하여 사용하는 경우를 도식화 한 것이다. 16×3으로 분할된 각 메모리는 동시에 읽혀지고, 3-클럭 후에는 3개의 심볼을 구성하는 데이터가 출력 되어진다.



[그림 1] 분할된 메모리의 사용

세로 방향(↓)으로 읽은 것은 first permutation의 결과이고, 각각의 메모리에서 읽혀진 4개의 데이터 간의 비트 재 정렬을 통해 second permutation의 결과를 얻어낸다. 4-비트가 1-심볼로 구성되므로 3-클럭 동안 모두 12-비트를 읽을 수 있다. 아래의 [표 6]은 16-QAM의 경우 각각의 메모리에 저장되는 데이터를 보여준다.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143

	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159

33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
161	162	163	164	165	166	167	168	169	170	171	172	173	174	175

49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
177	178	179	180	181	182	183	184	185	186	187	188	189	190	191

[표 6] 16-QAM의 메모리 데이터

위의 [표 6]에서 각 메모리에 음영으로 표시된 부분이 처음 3-클럭 동안 읽어오는 데이터이다. 블록 좌-상단부터 3-클럭동안 읽어오는 데이터는 (0,16,32,48), (64,80.96,112), (128,144,160,176)이 된다. 그리고 다음 4-6클럭 째에 읽어오는 데이터는 (1,17,33,49), (65,81, 97,113), (129,145,161,177)이 되는데, 여기에서 second permutation 과정의 비트 재 정렬이 요구되어진다. 즉, 각 단위 심볼로 읽어 들인 4개의 데이터 중 앞뒤의 2개 데이터 간의 위치를 바꾸어 출력하면 IEEE 802.11a 표준에서 제시하는 인터리빙의 결과가 유도되며, 아래 [표 7]은 비트 재 정렬 내용을 보여준다.

1 clk	2 clk	3 clk
0	64	128
16	80	144
32	96	160
48	112	176

4 clk	5 clk	6 clk
1	65	129
17	81	145
33	97	161
49	113	177

[표 7_a] 16-QAM의 비트 재 정렬

1 clk	2 clk	3 clk
0	96	192
16	112	208
32	128	224
48	144	240
64	160	256
80	176	272

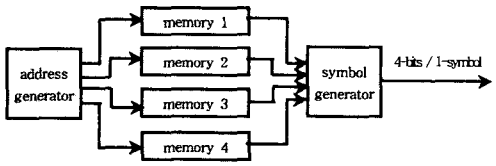
4 clk	5 clk	6 clk
1	97	193
17	113	209
33	129	225
49	145	241
65	161	257
81	177	273

7 clk	8 clk	9 clk
2	98	194
18	114	210
34	130	226
50	146	242
66	162	258
82	178	274

[표 7_b] 64-QAM의 비트 재 정렬

위와 같은 비트 재 정렬의 과정은 매 클럭마다 주기적으로 같은 흐름을 반복하게 된다.

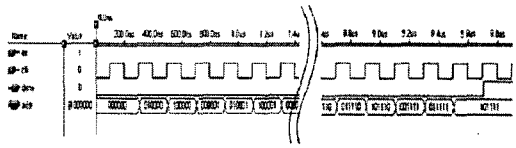
아래 [그림 2]는 지금까지 설명된 인터리빙의 과정을 도식화 한 것이다.



[그림 2] 16-QAM 인터리버 블록

위 [그림 2]처럼 address generator에서 write/read가 수행되며, 이 때의 write와 read는 select 신호 제어에 의해 각각의 결정 메모리로의 작업이 수행되고, 그 결과 read의 경우 symbol generator가 symbol 단위의 데이터를 읽어 들이면서 비트 재 정렬 과정을 마쳐 최종 4-비트/심볼의 데이터가 출력된다.

이러한 내용을 구현하는 데에는 가로 열(4-비트)과 세로 행(2-비트), 메모리(2-비트)의 번지 지정 스킴을 적용하며, 그 결과 아래 [그림 3]과 같이 예상했던 카운터들의 동작 타이밍도를 얻어 낼 수 있다. 즉, 분할된 메모리를 사용하면 x-QAM에서 x-값에 관계없이 동일 읽기 방식으로 48-클럭 내에 모든 읽기 작업이 완료된다.



[그림 3] x-QAM에서의 인터리빙 read 타이밍도

V. 결 론

802.11a의 인터리버와 디인터리버 블록 설계 OFDM 무선 모뎀에서는 두 단계 블록 인터리빙을 수행하며, 다양한 속도를 지원하기 위해 속도에 따라 다른 크기의 블록 인터리버를 사용한다. 그러나 각 속도에 따라 다른 크기의 완전한 메모리 1-plane은 데이터를 쓸 때와 읽어 올 때 N_{CBPS}에 포함된 데이터 개수만큼의 지연이 발생하게 된다.

그 점에 착안하여 본 논문은 두 단계의 인터리빙을 수행할 경우 분할된 메모리의 사용으로 지연 발생을 최소화 시키고자 데이터를 나누어서 쓴 다음, 읽어오는 과정에서 심볼 단위로 읽고 곧바로 IFFT에 전달하여 지연을 줄이는 방법을 사용하였다. 그리하여 기존 인터리버와 비교해서 데이터를 쓰고 읽는데 발생하는 지연과 또 심볼 단위 재 정렬에 추가적으로 발생되는 지연을 줄일 수 있었다. 아울러 이는 N_{CBPS} 값과 관계없이 x-QAM 모두 심볼의 개수인 48-클럭에 IFFT 단에 전달하는 방법이 된다.

더 나아가 두 단계 인터리버를 하나의 입,출력 관계식으로 유도하거나, write 하는 데에도 48-클럭만을 소요하는 방법이 제안될 경우, 더 적은 메모리 소자와 적은 클럭 수로 인터리빙이 가능할 것이다.

참 고 문 헌

[1] IEEE Std 802.11a : Wireless LAN Medium Access Control(MAC) and Physical Layer(specifications, 1999

[2] J. Hokfelt, O, Edfors and T. Maseng, Interleaver Design for Turbo Codes Based on the Performance of Iterative Decoding, submitted to ICC' 99, Vancouver, Canada.

[3] 정 찬형, 이 태진 저 "5GHz 대역 초고속 무선 기술", 2001

[4] 이 종훈, 김 우정, 송 상섭, "IEEE 802.11a 무선 랜을 위한 효율적인 인터리버 설계" JCCI, 2003