초고속 기가비트 IP 라우터를 위한 새로운 Lookup 장치와 VHDL 구현

정회원 강 승 민*, 송 재 원**

A Novel IP Lookup Scheme and VHDL Implementation for Fast Gigabit IP Routers

Seung-min Kang*, Jae-won Song** *Regular Members*

요 약

본 연구는 하드웨어로 쉽게 구현할 수 있는 초고속의 유니캐스트용 라우터의 Lookup 장치를 제안하고, VHDL로 구현하여 그 성능을 확인한 것이다. 장치 내부의 포워딩 표용 메모리는 매우 작은 크기이므로 저렴한 비용으로 고속의 SRAM을 사용할 수 있다. 예를들어, IPMA 사이트에서 구한 40,000개의 라우팅 정보는 대략 300~400KB의 포워딩 표로 축약될 수 있다. 대부분의 라우팅 Lookup은 1~2회의 메모리 접근을 통해 가능하고, 3회를 넘지않는다. ALTERA EPM7192시리즈에 100MHz 클럭을 이용하여 VHDL 모델링한 결과 10ns 접근속도를 가진 SRAM 기준으로 1회의 메모리 접근만으로 Lookup하는 경우 27ns 이내의 접근시간이 소요되며, 이는 대략 50 x 10^6 개/라우팅 Lookup/초)을 지원한다. 2회 이내의 메모리 접속으로 "다음홉"을 구할 수 있는 비율은 최대 95%까지이며, 소요 시간은 83ns 이내이다.

ABSTRACT

This study presents proposal and an VHDL implementation of a fast unicast route-lookup algorithm that be implemented by a hardware. The forwarding table is small enough to fit into a faster SRAM with low cost by the proposed scheme. For example, a large routing table with 40,000 routing entries obtained by IPMA website can be campacted into a forwarding table of 300~400KB with low cost. Most route lookups need only two memory access; no lookup needs more than three memory accesses. When we implemened by VHDL based on ALTERA EPM7192 device and a 100MHz clock under condition of one memory access with 10ns SRAM, this system needs 27ns memory access time and furnishes approximately 50 x 10^6 routing lookups/s. The beyond 95% lookups of incoming IP needs only the two memory access and interface processing time is below 83ns.

I. 서 론

인터넷(Internet)이 지난 수년간에 걸쳐 일반화되면서 네트웍 상호간의 트래픽은 급속히 증가되어 왔다. 앞으로 다양한 멀티미디어 정보에 대한 인터넷 서비스 요구를 감안할 때 인터넷 트래픽의 증가는 지속될 것이다. 이러한 사회적 요구를 충족시키고, 인터넷 서비스의 질을 향상시키기 위해선 인터넷상에 존재하는 IP 라우터(Router)의 발전이 수반되어야 하는데, 그 발전의 중요한 항목으로 1) 링크(Link)의 속도, 2) 라우터의 처리용량, 3) 패킷의 포워딩(Forwarding) 속도(율)등이 있다. 이중에 링크의

* 포항 선린대학 정보통신과 전임강사
** 경북대학교 전자공학부 교수
논문번호 : 99435-1028, 접수일자 : 1999년 10월 28일

속도 문제는 광통신의 비약적인 발전으로 해결의 과정중에 있고, 라우터 처리량 문제도 라우팅 인터페이스 기법의 발전과 스위칭의 계층별 분산 기법으로 해결되고 있다. 본 논문은 3번째 문제에 대한 연구의 결과이다.

포워딩(Forwarding)이란 IP 패킷의 헤더(Header)부에 있는 원격지 주소를 분석하여 적절히 출력 인터페이스 링크로 해당 패킷을 전송하는 것을 지칭한다. 특히 다음 홉(NextHop)으로의 패킷전달을 위해 최적의 메모리 구성과 접근을 포괄적으로 Lookup이라 한다. 그러나, 링크 속도의 비약적인 증가에 비해 라우터 내의 포워딩 속도(혹은 Lookup 속도)는 그리 개선되지 못하고 있다.

본 연구에서는 최대 프리픽스(Prefix, 유효 네트워크 주소) 정합형 (Longest - prefix matching) 기술을 근간으로 저비용, 초고속의 새로운 Lookup 알고리즘을 제안한다. 라우터의 속도는 Lookup 속도에 의해 결정된다. Lookup의 속도를 개선하기 위해선 Lookup 장치를 하드웨어로 구현해야 한다. 그러기 위해선 포워딩표의 소요 메모리 크기가 작아야 되고, 알고리즘이 단순해야 한다. 그래서 얼마전 논문에 제안된 캐쉬(Cashe) 메모리를 이용한 소프트웨어 기법의 Lookup[1]은 초대규모 네트워크에는 더 이상 적당하지 않다.

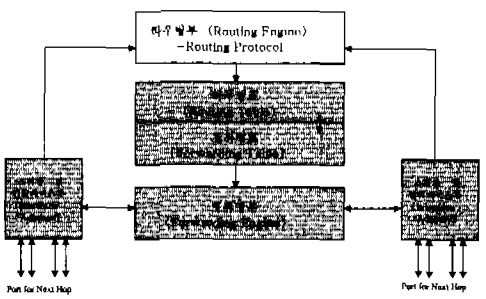

그림 1. IP 라우터 구조.

메모리 용량과 알고리즘의 단순성과 더불어 Lookup 과정에서 메모리의 접속수는 아주 중요한 문제로 대두된다. 당연히, 가능한 내부 메모리의 접속수가 적을수록 좋다. 결국 라우터의 개발과정에서 고려해야할 중요 요소는 다음과 같이 정리될 수 있다. Lookup 알고리즘이 단순해서 하드웨어로 구현이 용이해야 하고, 내부 소요 메모리가 매우 작아 초고속의 SRAM을 사용할 수 있어야 한다. 그리고, Lookup 과정에서 메모리 접속회수가 작아 고속성이 유지되어야 한다. 본 논문은 이러한 요소들을 충실히 수용하는 Lookup 장치를 제안하고, 하드웨어 구현의 바로 전 단계인 VHDL로 모델링하고 성능을 검증해 보았다.

Ⅱ. 본 론

2.1 초고속 IP 라우터의 구조

일반적으로 IP 라우터는 크게 스위칭부(Switching Fabric), 포워딩부(Forwarding Engine), 라우팅부(Routing Engine)로 구분된다. 그림 1은 이러한 라우터 구조를 도식적으로 나타낸 것이다. 라우팅부는

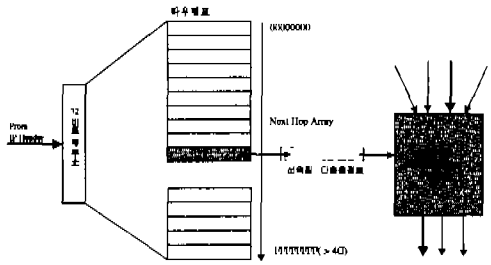

그림 2. 직접방식의 Lookup 기법

라우팅 프로토콜(RIP, OSPF와 같은)이 실행되어 패킷이 최적의 경로를 찾는데 필요한 정보를 저장하는 기능을 한다. 부가적으로 이들은 라우팅 정보를 포워딩부가 Lookup 알고리즘을 수행하기 위해 적절한 형태로 자료를 재가공한다. 이러한 과정은 몇분에 한번씩 동적으로 수행되며 수행시간은 내부 CPU와 관련 프로토콜에 따라 결정된다.

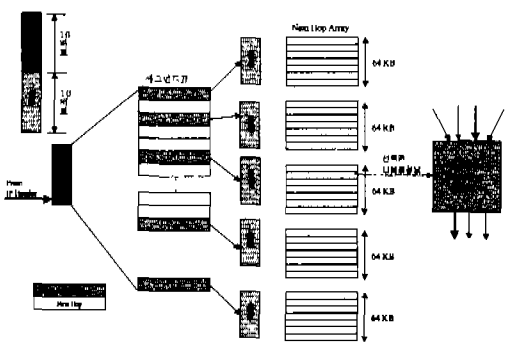

그림 3. 간접방식에 의한 Lookup 기법

포워딩부는 패킷이 입력되면 헤더부를 분석하고 이들의 전송 경로를 포워딩표(Forwarding Table)을

이용하여 신속히 결정하고 해당 출력포트로 경로를 열어주는 역할을 한다. 이는 라우터의 속도를 결정하는 부분이다.

2.2 기존의 프리픽스 정합형 IP Lookup 알고리즘

IP 라우터의 Lookup을 하드웨어로 구현하는 방법에는 여러 가지가 있으나, 근래 최대 프리픽스(Prefix, 유효 네트웍 IP주소)를 정합하여 Lookup하는 기법에 관심이 집중되고 있다. 이들 방법 중에는 그림 2와 같이 32비트 IP주소 전체를 포워딩표(일종의 메모리)의 주소로 사용하는 "직접방식"이 있다. 이는 한번의 메모리 접근으로 원하는 정보를 얻을 수 있으므로 속도 측면에서 아주 이상적이라 할 수 있다. 그러나 이러한 "직접방식"은 매우 큰 메모리를 (2^{32} = 4Gbyte) 요구하므로 현실성이 없다.

따라서, 메모리의 크기를 줄이기 위해 일반적으로 "간접방식"을 사용한다. 그림 3처럼 각 IP의 주소를 두부분으로 나눈다. 1) 세그먼트(Segmemt : 16 bit)와 2) 옵셋(Offset : 16 bit). 세그먼트표의 크기는 64KB가 되고 세그먼트표의 내용은 출력포트의 직접적인 정보가 되거나 다음홉배열(Next Hop Array : NHA)을 담고 있는 메모리의 초기주소를 지정하는 포인터(Pointer)가 된다. NHA의 주소로 IP주소의 옵셋부를 사용한다. NHA의 저장 자료는 다음홉의 출력포트정보가 된다. 즉, 원격지 IP 주소가 a.b.y.z 인 경우 a.b는 세그먼트표의 주소가 되고 y.z는 NHA의 인덱스(Index)로 사용된다. 세그먼트 a.b에 대해 프리픽스의 길이가 16비트 이하이면 세그먼트의 내용이 직접적인 다음홉의 출력포트 정보가 되고, NHA가 불필요하게 된다. 이 경우 한번의 메모리 접근으로 패킷의 출력정보를 얻을 수 있다. Class B의 IP 패킷이 이에 해당한다. 그러나, 프리픽스 길이가 16비트 이상이면, 옵셋이 필요하고 하위 16비트 모두를 옵셋 주소로 사용하므로 각 세그먼트별로 64KB의 NHA가 요구된다. 이 경우 IP Lookup 과정에서 최대 2번의 메모리 접근이 필요하다. 이러한 간접 Lookup 기법 중에 세그먼트부 주소를 24비트로 배당한 경우도 있다. 이 경우 세그먼트용 메모리로 16MB가 필요하다.

이러한 방식은 Lookup 과정에서 메모리의 접근회수가 최대 2회 이내이므로 Lookup 속도를 개선하고 동시에 메모리의 크기를 어느 정도 줄일수 있다(직접방식에 비해). 그러나 IP의 유효 주소(프리픽스 길이)가 가변적이라는 사실을 이용하지 못함으로

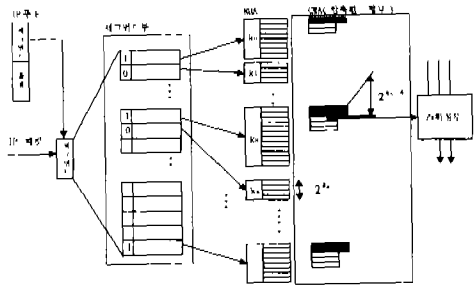

그림 4. 제안된 Lookup 기법의 개념도(가변 옵셋과 압축기법의 적용).

프리픽스가 16보다 긴 모든 경우에 대해 64KB의 NHA를 배당하게 되어 메모리 크기가 여전히 SRAM과 같은 고가의 메모리를 사용할 정도로 감소된 것은 아니다.

2.3 제안된 IP Lookup 알고리즘

현실적으로 동일한 세그먼트부 주소를 가진 패킷들은 세그먼트부는 같더라도 프리픽스의 길이는 다를수 있다. 이때, 동일한 세그먼트를 가진 일련의 집합에서 가장 긴 프리픽스의 길이에 16을 뺀 수치를 "옵셋길이(k_n)"로 하고, IP 주소부 중에 하위 16비트부터 16-k_n비트까지를 NHA의 주소로 사용하면, NHA의 크기를 줄일 수 있다. 그 결과 하나의 세그먼트에 대해 NHA의 메모리 크기는 2^{kn}이 된다. 이전의 방식에서 NHA의 길이는 모두 2^{16}인 것에 비해 메모리 크기가 더욱 작아진다. 이를 일종의 가변 오셋방식(Variable Offset)이라 한다. k_n이 큰 경우 여전히 NHA의 크기가 꽤 크므로 이경우는 압축알고리즘을 이용하여 NHA의 크기를 더욱 줄일 수 있다. 그림 4는 본 Lookup 장치를 개략적으로 나타내 것이다.

이처럼 본 알고리즘은 기존의 최대 프리픽스 정합형(Longest-Prefix Matching)형을 기반으로 하면서 "가변 옵셋" 기법에 "압축 알고리즘"을 적용한 복합적인 알고리즘이다. 이러한 복합적인 알고리즘은 기존의 논문에서도 발표된 예가 있으나[2], 본 논문에서는 메모리 구조의 크기를 최소화하고, 특히 메모리 접근회수를 극소화하는 기법을 제안한다. 이는 본 논문의 지향점이 Lookup 장치의 "하드웨어 구현성 및 경제성"과 더불어 "초고속성"이기 때문이다.

(1) 세그먼트 표의 구조

그림 5에 나타낸 것처럼 세그먼트는 크게 3부분

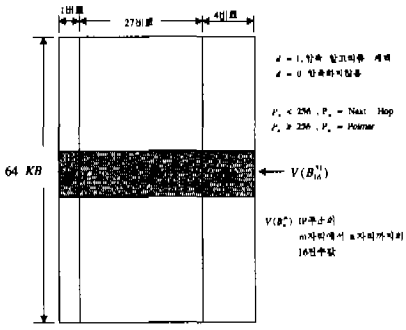

그림 5. 세그먼트 표의 구조

으로 나눈다. 최상위 1비트는 현재의 IP에 대해 압축알고리즘 채택 여부(d=1이면, 압축함)를 지시하는 비트이다. 중간의 27비트는 NH(Next Hop)을 저장하고 있는 메모리들의 "포인터"이거나 "다음홉정보"이다. 즉, 이 값이 256보다 작으면 "다음홉정보"이고, 아니면 포인터가 된다. 최하위 4비트(k)는 윱셀길이(=k+1)를 나타낸다. 일반적인 Lookup 알고리즘에서 특정 프리픽스 집합에서 윱셀길이(k)가 크면 소요 메모리 크기가 증가하므로 압축기법을 사용한다. 그러나 본 알고리즘에서는 어떤 임계치의 k_t를 미리 정하고 이 값을 넘으면 압축기법을 무조건 적용하는 것이 아니라, 메모리 크기와 메모리 접근 회수, 프리픽스길이의 분포등을 고려하여 압축유무를 결정한다.

표 1은 IPMA 사이트에서 구한 라우팅표의 일부분이고, 표2는 이를 이용하여 재구성한 세그먼트 표이다.

(2) NHA 산출

일련의 동일 세그먼트를 가진 프리픽스 집합의 NHA를 계산하는 알고리즘은 다음과 같다.

1) 동일 세그먼트를 갖는 프리픽스들이 라우팅 표로부터 입력되면($P=\{p_0, p_1, ...p_{m-1}\}$)
2) 프리픽스 길이에 따라 순서적으로 분류한다. i번째 IP의 프리픽스 길이를 l_i, j번째 IP의 프리픽스 길이를 lj 라 하면, $l_i \le$ lj인 경우 p_i를 먼저 두고 p_j를 나중에 두는 방식으로 분류한다.
3) 집합 P의 "최대 프리픽스 길이"에 16을 뺀 값을 윱셀길이(k_n)로 정한다. 이때 NHA은 1에서 2^{kn}까지의 일차원적 배열로 표현된다(즉, NHA는 2^{kn}의 크기를 가진다).
4) 각 프리픽스(p_i) 별로 전체 NHA 배열내에서 해당 프리픽스의 구간(시작주소, 끝주소)을 계산한다. 이 구간에 해당 프리픽스의

$$
\begin{aligned}
&s_i = 1 \quad \text{if } i \le l_i - 16 \\
&s_i = 0 \quad \text{if } l_i - 16 < i < k \\
&e_i = 0 \quad \text{if } i \le l_i - 16 \\
&e_i = 1 \quad \text{if } l_i - 16 < i < k \\
&SA_n = [s_0 s_1 .. s_i .. s_{k-1}] \; AND \; [x_0 x_1 ... x_{k-1}] \\
&EA_n = [e_0 e_1 .. s_i .. e_{k-1}] \; OR \; [x_0 x_1 ... x_{k-1}]
\end{aligned}
\qquad (1)
$$

출력포트값을 배당한다. 구간 계산방법은 다음과 같다(식 1참조). 해당 프리픽스가 a.b.x.y라할 때, x.y의 이진수 표현을 $x_0x_1x_2,...x_{15}$로 한다. 윱셀길이가 $k(=l_{max}-16)$라면, k비트의 시작점 마스크 s_0, $s_1,...s_{k-1}$와 끝점 마스크 e_0, $e_1,...e_{k-1}$를 우선 다음과 같은 방법으로 구한다. k비트 중에 자신의 프리픽스 길이에서 16을 뺀 값보다 낮은 자리는 "1"로, 그외는 "0"으로 하여 시작점 마스크를 구한고, 자신의 프리픽스 길이에 16을 뺀 값보다 높은 자리는 "1"로 낮은 자리는 "0"으로하여 끝점 마스크를 구한다. 시작점 마스크와 x.y를 AND 취하여 배열내의 "시작점주소"을 구하고 끝점 마스크와 x.y를 OR 취하여 "끝점주소"을 구한다. 예를들어 p_i=a.b.60.50, 프리픽스 길이(l_i)가 26, 윱셀길이 k가 12(해당 집합에서 최대 프리픽스가 28이라 가정하면, 28-16=12이므로), 출력 포트가 2인 경우를 살펴본다. k=12이므로 전체 배열은 0에서 4095(=2^{12})까지 존재한다. 60.50을 12비트로 표시하면 011000000101 이다. 자신의 프리픽스 길이가 26이므로 10비트자리(26-16

표 1. IPMA 사이트에서 구한 실시간 라우팅 표(1999년 10월 2일)

193.23.152	22	10
193.23.164	24	1
193.23.176	22	2
193.24.16	20	9
193.24.212	24	3
193.26.20	24	6
193.26.35	24	7
193.26.202.1	32	7
194	8	6
195.1	16	9
195.1.100.98	32	17

표 2. 표 1에 근거하여 재구성된 세그먼트 표

세그먼트 주소	d	포인터/NH	k-1
193.23	0	Pointer0	7
193.24	0	Pointer1	7
193.25	0	Default NH	0
193.26	1	Pointer2	15
193.27	0	Default NH	0
193.28	"	"	0
:	"	"	0
193.255	"	"	0
194.0	0	6	0
194.1	0	6	0
:	0	6	0
194.254	0	6	0
194.255	0	6	0
195.1	1	Pointer3	15
:	:	:	:

= 10이므로) 까지는 "1", 나머지 2비트는 "0"을 채워 시작점 마스크를 구한다. 그래서, 시작점 마스크는 111111111100 이며, 끝점 마스크는 000000000011이다. 따라서 시작점은 60.50과 시작점 마스크를 AND 취하면 011000000101 = 1540, 끝점은 1543이 된다. 전체 NHA배열중에서 이 구간(1540~1543)내에 출력포트값 2를 배당한다.

5) 4)번 과정을 전 프리픽스에 대해 적용한다. 표 3은 본 과정의 구체적인 예이다.

예를 들어, 동일 세그먼트를 가진 프리픽스 집합을 [프리픽스/프리픽스길이/다음홉출력포트]의 형태로 표현한 다음과 같은 경우에

{192.168/16/1,
192.168.58/18/2, 192.168.92/24/1,
192.168.58.32/26/3, 192.168.255.240/28/5,
192.168.58.36/32/8}

최대 프리픽스길이는 32이므로 옵셋길이(k)는 16(=32-16)이다. 따라서 NHA의 크기는 2^{16}이다. 각각의 프리픽스에 대한 NHA내의 구간을 앞의 과정을 이용하여 구하면 표 3, 4와 같다. 표 3은 각 프리픽에 대한 시작 주소와 끝점 주소를 나타낸 것이다. 각 프리픽스별로 겹치는 부분이 있는데, 이를 일차원적으로 다시 배열하여 NHA를 표시한 것이 표 4이다. 그림 6은 표 4를 이용하여 NHA의 일차원적 예를 도식적으로 표현한 것이다.

그림 6. NHA의 예

표 3. 각 프리픽스별 구간 배당

프리픽스	시작점주소	끝점주소
192.168/16/1	0	65535
192.168.58/18/2	0	16383
192.168.92/24/1	23552	23807
192.168.58.32/26/3	14848	14911
192.168.255.240/28/5	65520	65535
192.168.58.36/32/8	14884	14884

표 4. 표 3의 NHA 일차원적 배열.

주소(십진수)	주소(범위)	출력포트
0	14847	2
14848	14883	3
14884	14884	8
14885	14911	3
14912	16383	2
16384	65519	1
65520	65535	5

(3) 압축 알고리즘의 적용

옵셋길이(k)가 특정치 이상인 경우 NHA의 크기가 증가하여 메모리를 많이 소모하므로 압축기법을 적용하여 메모리의 크기를 더 줄인다. 본 Lookup 알고리즘에 적용할 압축알고리즘은 다음과 같다.

우선 압축된 정보는 CWA(Code word array)로 표현되고, 이는 다시 CBM (Compression Bit Map)과 CNHA (Compressed NHA) 구성된다. 압축은 CBM과 CNHA를 유도하는 과정으로 시작한다.

일단 (2)의 방법으로 NHA을 구하고, NHA의 i번째 값을 a_i, CBM의 i번째 값을 b_i , CNHA의 j번째 값을 c_j라 하면

$$b_{i+1}=0,\ b_i=1 \qquad \text{if } a_i=a_{i+1},$$
$$b_{i+1}=1,\ c_j=a_{i+1},\quad j=j+1 \quad \text{if } a_i\neq a_{i+1}. \tag{2}$$

와 같이 CBM과 CNHA를 구할 수 있다. 즉, NHA 값의 일차원적 배열을 순차적으로 한비트씩 옮기면서 이웃하는 값이 서로 다를 경우 CBM의 해당 자리에 1을 배당하고 같은 경우는 0을 배당하여 압축 비트맵 CBM을 작성한다. CBM이 1이 될 때마다 CNHA에 해당 NH 값을 채운다. 그림 7은 이러한 압축기법을 그림으로 나타낸 것이다.

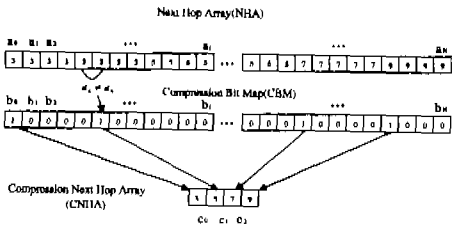

그림 7. 압축 알고리즘의 예

CBM과 CNHA로 구성된 CWA(Code word Array)를 메모리상에 저장한다. CWA는 CBM을 16비트 단위로 나누어(압축비율을 16:1로 하기 위해) 배열하고 이를 Map으로 명명한다. 이전 워드(Word)의 Map들에 누적된 "1"의 개수를 Base라 명명하여 마찬가지로 같은 주소상에 배열한다. 그림 8은 CWA의 구조를 나타낸다. Base는 CNHA의 주소를 계산하는데 사용된다. 여기서 유의해야 할 점은 현재 워드의 Map이 $0000_{(16)}$이면 Base자리에 누적된 "1"의 개수를 저장하지 않고 "다음홉"의 출력포트 번호를 직접 저장하는 것이다. Map이 $0000_{(16)}$이라는 것은 이 구간에서의 다음홉이 동일한 값이라는 것을 의미하므로 Base값을 저장할 필요가 없다. 계속 이어지는 CWA에서 Map=$0000_{(16)}$이 아닌 최초의 Base 자리만으로 누적된 1의 정보를 충분히 알 수 있기 때문이다. 이는 본 알고리즘의 중요 특징으로,

$$Base(i) = \sum_{n=1}^{n=i-1}\sum_{k=1}^{16} Map(n,k), \quad \text{if } Map_i \neq 0000_{(16)} \tag{3}$$
$$= NextHop, \quad \text{if } Map_i = 0000_{(16)}$$

많은 경우에 걸쳐 2회의 메모리 접근만으로 다음홉을 구할 수 있게 된다. 이렇게 Map과 Base가 완성되면 CNHA를 뒤이어 연속적으로 배열한다.

CBM을 16비트 단위로 나누어 저장하였으므로 우리는 NHA를 16 : 1로 압축하였다. 따라서 IP 주소중 세그먼트 16비트를 제외한 하위 16비트 주소($x_0x_1...x_{15}$)중에 k-4비트($x_0x_1...x_{k-4}$, k : 윤셀길이)를 CWA의 주소로 활용하여 Map과 Base 값을 읽어낸다. 전술한 것처럼 만약 이때의 Map값이 $0000_{(16)}$이면 현재의 Base값은 다음 홉의 출력 포트값이 된다. 그렇지 않으면 k 비트($x_0x_1...x_{k-1}$)중 CWA의 주소로 활용한 부분을 제외한 4비트($x_{k-4}x_{k-3}x_{k-2}x_{k-1}$) 주소의 십진수 값을 w(0에서 15)라 했을때, 현재의 Map의 0자리부터 w 자리까지 누적된 "1"의 개수($|w|$)를 다시 구한다. 이를 현재의 Base값에 더하여, 그 결과를 CNHA를 찾기위한 주소로 사용한다.

결국 CNHA의 주소(A_{CNHA})는

$$A_{CNHA} = P_n + 2^{k-4} + B_m + |w| \tag{4}$$

이 된다. P_n는 세그먼트에 저장된 포인터이고, 2^{k-4}는 CWA의 최대 길이(CNHA를 제외한), B_m는 현재 워드의 Base값이다.

(4) 압축알고리즘 수행의 근거

압축알고리즘의 수행 여부는 다음 몇 가지 조건을 충족해야 한다. 첫째, 초기 설계 과정에서 배당된(확정된) 메모리 크기에 비해 만들어진(재가공된) 포워딩표의 크기가 작아야 된다. 이 조건을 충족시키기 위해 압축수행을 위한 윤셀임계치(k_t, $k>k_t$이면 압축함)를 조절해야 한다. 즉, 재가공된 포워딩표의 크기가 하드웨로 결정된 메모리에 비해 여유가 있으면 압축 알고리즘을 적용할 k의 임계치를 상향 조절하여 압축 알고리즘을 적용하지 않아 단지 2번의 메모리 접속만으로 출력포트값을 얻을 수 있는 IP영역을 확대한다. 그러나 포워딩표의 크기가 증가하면 압축 알고리즘을 적용할 임계치 k값을 낮추어(예를들면 4이상) 많은 범위에 걸쳐 압축알고리즘을 수행한다. 이 경우 메모리 접근 횟수가 3회인 경우가 확률적으로 높아도 감수해야 한다.

둘째, 압축을 수행하는 임계 윤셀의 값을 증가시키면 압축하지 않는 부분이 증가하므로 재가공된 포워딩 표의 크기가 증가하게 된다. 그런데, 어느 범위 내에서 증가율이 미미하다가 특정 k_t에서 증가율이 급해진다. 즉, 임계윤셀이 특정 값이 되면 갑자기 소요 메모리가 급격히 증가하는데, 이때의 값을 윤셀임계치로 결정한다. 그림 11에서 임계윤셀이

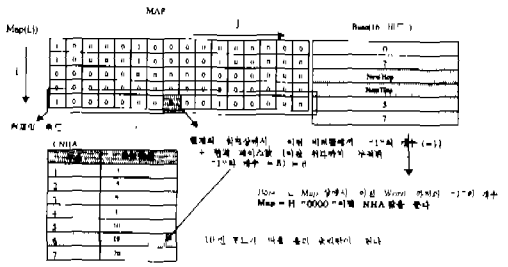

그림 8. CWA 구조

9이면 소요 메모리의 크기가 1MB로 급격히 증가함을 알 수 있다.

셋째, 일단 압축된 CWA 중에서도 2회의 메모리 접근으로 원하는 정보를 얻을 수 있는 경우(Map=$0000_{(16)}$)와 3회의 메모리 접근으로 원하는 정보를 구할 수 있는 것으로 구분되는데, 이들간의 비율이 특정수치 이상이 되어서는 안된다. 압축을 통해 메모리의 크기를 줄이는 것은 효과적인 설계이지만 그 결과 3회의 메모리 접근을 통해서 원하는 다음홉정보를 얻는 비율이 높아지면 초고속 스위칭이라는 최초의 목표를 달성하기가 어려워지기 때문이다. 특히, 2회와 3회의 메모리 접속을 단순히 1회차일 뿐이라는 판단은 잘못된 것이다. 모사실험과정에서 1회의 메모리 접속은 27ns. 2회의 메모리 접속은 ~83ns, 3번의 메모리 접속은 182ns정도가 소요되어 (가산기, 병렬16비트 가산기의 소요 시간 때문에) 하드웨어 구현시 이들간의 차이는 확연히 들어난다. 따라서 이들간의 Lookup 비용은 달리 설정하여야 한다.

2.3 하드웨어 구조

그림 9는 본 알고리즘을 구현하기 위한 하드웨어 구조이다. 우선 세그먼트 표를 위한 메모리부가 있다. IP 주소의 상위 16비트를 이용하여 세그먼트 표 자료를 읽어낸다. 포인터(P_n) 값이 256보다 작으면 이값을 곧장 출력단의 선택기를 통해 출력시킨다. 만약 256보다 크면, 이를 CWA/CNHA 혹은 NHA 값을 읽기 위한 포인터로 사용한다. 이 상황에서 최상위 1비트(d) 값이 0이면, 압축 알고리즘을 채택하지 않은 것이므로 CWA의 Map, Base를 참조하지 않고 직접 NHA 정보를 읽어 출력시킨다. d=1이면 압축알고리즘을 채택한 것이므로 포인터와 $V(B^{15}_{15-k+4})$를 더한 주소에서 Map, Base를 읽어낸다. 메모리 접근을 위한 타이밍은 내부 제어기에서 발생된다. Map값이 $0000_{(16)}$이면 이때의 Base자리의 수치가 NH 정보이므로 이를 출력시킨다. $0000_{(16)}$이 아니면, 0에서 $V(B^{15-k}_{16-k+4})$자리까지 "1"로 채운 마스크 바이트를 만들고, 이를 Map(M_m)과 AND한다. 이값을 16비트 병렬 가산기(16개의 1비트 가산기)에 입력하여 현재 워드 내부에 존재하는 "1"의 개

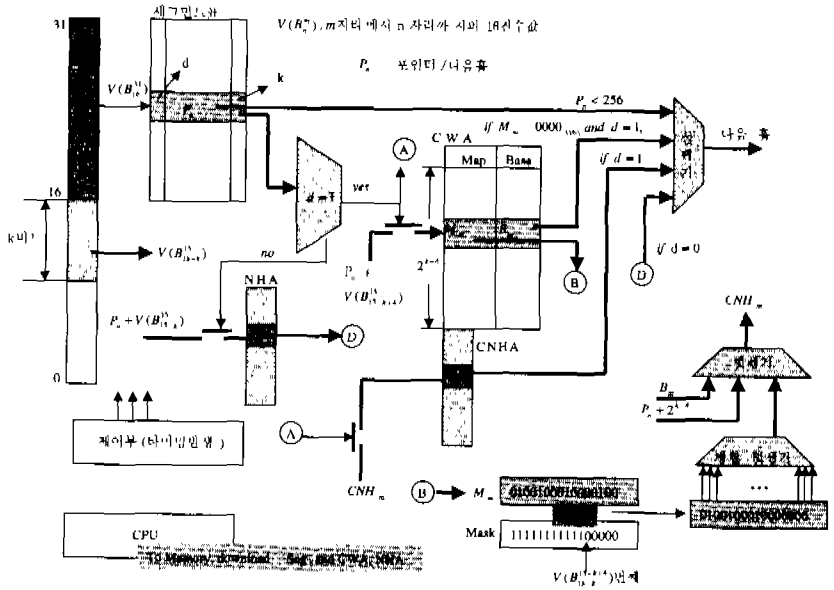

그림 9. 하드웨어 구조

수를 구한다. 이 병렬 가산기의 결과를 현재의 Base(B_m), 포인터(P_n)와 더하여 새로운 CWA의 주소를 만든다. 이 주소를 이용하여 CNHA에 접근하여 자료를 읽어내어 출력한다.

Ⅲ. 성능분석

3.1 소요 메모리와 메모리 접속 회수

본 알고리즘의 성능을 분석하기 위해 현재 운용중인 IP 라우터의 라우팅표 정보가 필요하다. 이는 IPMA(Internet Performance Measurement and Analysis) 프로젝트의 웹사이트[5]에서 구할 수 있다.

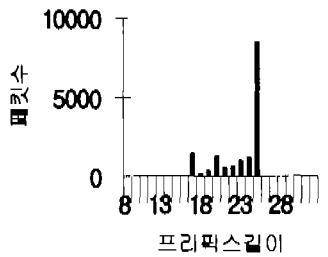

그림 10. 프리픽스 길이별 분포

IPMA 인터넷 사이트는 실시간으로 라우팅표 자료를 제공한다. 그림 10은 IPMA사이트에서 얻은 프리픽스의 길이 분포를 그래프로 표시한 것이다. 인터넷 라우터를 통해 전달되는 패킷의 프리픽스 길이는 통계적으로 24인 경우가 가장 많다(대부분 24 이내이다). 현장에서 실제 운영중인 라우팅 정보 웹사이트에서 제공받아 본 알고리즘을 검증하는 것은 검증은 가치를 높이기 위해서이다.

본 알고리즘의 성능을 3가지 관련 알고리즘 -DIR-24-8, DIR-21-3-8, SFT-[3,4]과의 비교를 통해 분석하고자 한다. 표 5는 기존의 기법과 본 기법간의 비교 분석표이다. SFT(Small Forwarding-Table)[3] 기법이 최소의 메모리 크기를 요구하고, SRAM으로 구현 가능하다. 그러나 메모리의 접근회수가 2에서 9번까지로 초고속 Lookup에는 적합하지 않다. DIR-24-8 기법은 메모리 접근 횟수는 1에서 2번 사이로 초고속 라우팅에는 적합하나, 메모리의 크기가 33MB로 비현실적으로 크다. 그래서 SRAM으로 구현이 불가능하여 DRAM으로 구현할 수밖에 없다. DIR-21-3-8 기법은 메모리 크기를 9MB까지 줄였으나 최대 3번의 메모리 접근을 요구하고, 여전히 SRAM사용이 불가능한 기법이다. 그러나 본 기법은 1999년 9월 28일부터 1999년 10월 2일까지 IPMA사이트에서 구한 라우팅 정보를 이용하여 계산한 결과 ~316KB 정도의 SRAM으로 구현 가능함을 확인하였다. 그림 11은 같은 기간동안 IPMA 사이트에서 구한 라우팅정보를 임계 윤셀값(k_i)을 4에서 9까지 가변하면서 포워딩 표의 크기를 구하여 표현한 것이다. k_i를 8보다 작게 하면 소요 메모리의 용량이 거의 비슷(~316KB)하지만 9이상이 되면 소요메모리가 1MB로 크게 증가한다. 따라

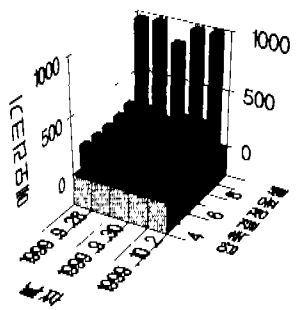

그림 11. 압축수행을 결정하는 임계윤셀에 따른 소요 메모리의 변화

표 5. 여러 Lookup 기법들의 비교

본 기법	1	15%	~316KB	Yes
	2	80%		
	3	5%		
DIR-24-8	1/2	50%	33MB	No
DIR-21-3-8	1/3	-	9MB	No
SFT	2/9	-	150~160KB	Yes

표 6. 포워딩표의 소요 메모리(단위 : KB)

'99.9.28	255	258	263	278	310	989
'99.9.29	257	260	265	280	313	997
'99.9.30	220	222	226	236	256	850
'99.10.1	259	262	268	283	316	1000
'99.10.2	258	261	267	282	315	1000

서 이 경우는 k_i를 8로 결정해야 한다. 표 6은 이러한 소요 메모리 분포를 수치로 상세히 나타낸 것이다. 본 기법을 사용하면 95%이상 2회 이내의 메모리 접속만으로 다음홉정보를 얻을 수 있다. 그러나 만약 CWA 구조에서 Map=$0000_{(16)}$인 경우를 구분하여 활용하지 않으면 2회 메모리 접속 비율이 67%로 떨어진다. 즉, Base를 단순히 CBM의 누적된 "1" 개수로만 사용한다면 3회의 접속을 필요하는 경우가 많아진다. 특히, 옵셋길이가 긴 프리픽스가 많을수록 그 비율은 더욱 증가한다.

프리픽스 길이가 24보다 큰 경우를 각 프리픽스마다 최소 하나 이상 삽입한 라우팅 표를 본 알고리즘에 적용하면 소요 메모리는 최대 2MB까지 증가한다. 사실 그림 10에 알 수 있듯이 이러한 경우는 라우터에서 확률적으로 발생빈도가 낮으나 본 알고리즘의 성능을 좀더 극단적인 경우에서 검증받기 위해서 적용해 보았다.

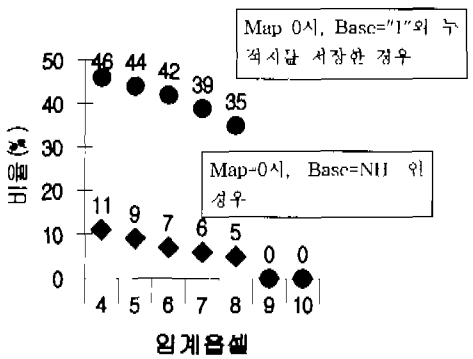

그림 12. 임계옵셋에 따른 3회접속 비율

표 7은 임의 프리픽스 집합을 선택하여 CWA를 구한 예이다. 옵셋길이가 10비트이므로 전체 CWA 워드수는 2^6(=64)이다. 여기서 주의해야 할 사항은 비록 압축하더라도 2회의 메모리 접근만으로 출력포트를 구할 수 있는 비율이 92%라는 사실이다. 이는 Map = $0000_{(16)}$인 워드의 Base자리에 "다음홉정보"를 저장함으로 가능해진 사실인데, 이러한 기법이 전체 포워딩표에 적용되면 단지 2회의 메모리 접근 비율이 95%이상으로 증가하게 되어 스위칭속도가 전반적으로 증가하게 된다. 이것이 본 논문의 주요 특징이다. 그림 12는 Map=0000인 경우에 Base 자리에 단순히 누적된 "1"의 값으로 채울 경우와 Base자리에 다음홉 포트값을 직접 기록한 경우간의 3회 메모리 접속비율을 비교 표시한 것이다.

표 7. 압축된 CWA에서 메모리 접근회수의 비교예

워드 번호	Map	Base	2회	3회	접속 회수
1	1000000000000000	0		16	16
2	0000000000000000	3(NH)	16		:
3~9	:	3(NH)	112		:
10	1000000000000000	1		16	
11	1000000000000000	2		16	
12	0000000000000000	1(NH)	16		
13~19	:	1(NH)	112		
20	0000000010001000	3		16	
21	0000000000000000	4(NH)	16		
22~32	:	4(NH)	176		
33	1100000000000000	5		16	
34	0000000000000000	1(NH)	16		
35~64	:	1(NH)	480		
26			944	80	1024
비율			92%	8%	

Map=0000시 Base 자리에 다음홉출력 포트값을 기록하면 2회접속만으로 다음홉을 구할 수 있는 비율이 크게 증가하는 것을 확인할 수 있다.

3.2 VHDL 구현을 통한 Platform 설계

그림 13은 본 장치를 VHDL로 구현하기 위해 제안한 플랫폼의 구성도이다. 우선 직렬 형태의 IP를 병렬 형태로 변환하고 헤더부를 분석, 세그먼트 자료를 읽어내는데 필요한 타이밍을 만들어내는 "IP 헤더 분석부(IP Header Analysis : IPHA)"가 있다. 이는 독립된 한개의 EEPLD(Altera사 EPM7192) 상에서 구현될 것에 대비하여 설계되었다. IP 패킷 신호가 직렬 형태로 입력되면 프레임 동기신호가 검출되어 전체 블록의 동작이 시작된다. 32 클럭 단위로 IP의 헤더부분이 나누어지고 5번째 워드가 시작되는 시점에서(원격지 주소이므로) 세그먼트 메모리로부터 정보를 읽어오기 위한 타이밍(notSeg-MemRD, notSeg MemCS) 신호가 발생된다. 세그먼트 정보는 notSegLatch에 의해 외부 Latch에 저장된다.

옵셋길이(k)는 읽혀온 세그먼트 정보의 최하위 4비트로 부터 알 수 있다. IPHA부에서 출력된 IP주소의 하위 16비트 값은 옵셋길이를 이용하여 유효한 비트(l_{max}-16)만을 추출하여 하위자리에서부터 상위로 유효비트만큼 채워나가고, 상위 비트는 "0"로

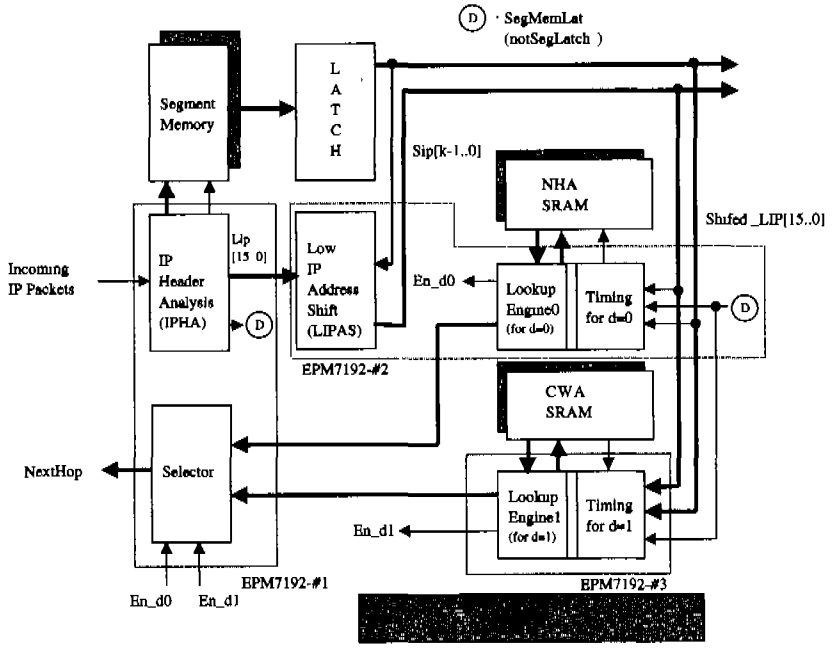

그림 13. VHDL 모델링을 위한 플랫폼 구조

채운다. 이는 유효 주소를 Hex로 변환하기 위해서이다. 이는 LIPAS(Low IP Address shifter)부에서 수행된다(그림 14참조).

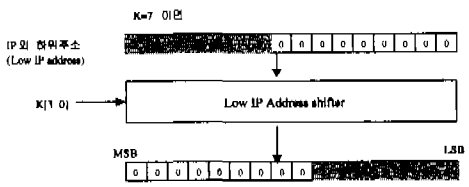

그림 14. LIPAS의 동작원리

래치(Latch)된 세그먼트 정보와 Hex로 변환된 유효 하위주소는 Lookup Engine(for d=0 or d=1)부로 전달된다. 최상위 1비트(d)는 두 Engine의 동작 상태를 결정한다. d=0이면 Lookup Engine0이 동작하고 Lookup Engine1은 정지상태가 된다. 우선 세그먼트 정보중에 포인터 자리의 값이 255이하이면 포인터가 NH이 되어 출력된다. 이때 NHA 접근을 위한 타이밍은 불필요하다. 포인터 값이 256이상이면 NHA로부터 NH 정보를 읽어와야 한다. 이를 위해 내부 타이밍 발생기는 NHA 메모리 접근을 위한 주소(포인터+Sip[k-1..0])와 타이밍을 발생시킨다. 그 결과 NHA로부터 NH 출력포트를 알 수 있고, 이를 선택기를 통해 출력시킨다.

d=1이면 Lookup Engine1이 동작하고 Lookup Engine0은 정지상태가 된다. 우선 세그먼트 정보중에 포인터 자리의 값과 Sip[k-1..4]을 이용하여 CWA정보를 읽어들인다. 물론 관련 타이밍은 메모리로 공급된다. Map값을 먼저 확인하여 그 값이 0000이면 Base 값이 NH 이므로 출력시킨다. 0000이 아니면 식(3)에 근거한데로 CNHA의 주소를 계산하고 관련 타이밍의 협조로 CNHA로부터 NH정보를 읽어들여 출력시킨다.

선택부(Selector)는 Lookup Engine0, Lookup Engine1로부터 전달받은 NH 정보를 En_d0, En_d1를 이용하여 최종 출력단으로 선택 전달한다. 전체 플랫폼은 결국 Altera사의 EPM7192 3개를 이용하여 구현될 수 있다.

3.3 구성부의 타이밍 분석

(1) 헤더분석부(IPHA)의 타이밍

그림 15는 IP 헤더 분석부(IPHA)의 입출력 내부 타이밍을 시뮬레이션한 결과이다. 프레임 동기 신호(frame_sync)를 검출하고 32비트 단위로 헤더를 분리한다. 5번째 워드가 원격지 주소이므로 이를 외부 32비트 Latch에 저장함(notSegLatch을 이용)과 동시에 세그먼트 메모리로 부터 자료를 읽기 위한 타이밍(notSegMemCS/RD)을 발생시킨다. 자료 읽기 시

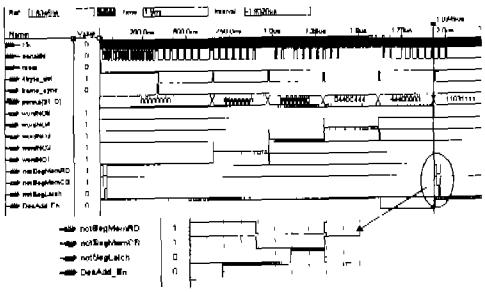

그림 15. IPHA의 타이밍

같이 10ns인 고속 SRAM을 사용한 것을 대비하여 notSegMemCS/RD는 20ns의 폭을 가지게 설계하였으며, notSegLatch는 notSegMemRD가 Enable되고 10ns 경과후 "1"이 되게 하였다. 결국 세그먼트 자료를 읽기(fetch)까지 15ns 소요된다.

(2) 1회 메모리 접속으로 NH을 구하는 경우

1회 메모리 접속으로 Lookup 완료하는 경우는 세그먼트로부터 읽은 자료가 256보다 작은 경우이다. 이 경우 역시 d=0인 경우이므로 Lookup Engine0에서 수행된다. 그림 16은 이 경우의 타이

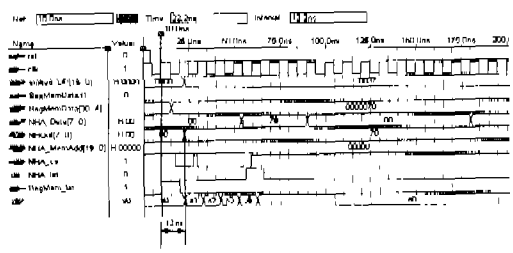

그림 16. 1회 메모리 접속으로 NH를 구하는 경우의 타이밍

밍도이다. 세그먼트부에서 읽혀온 정보는 일단 Lookup Engine0에 제공되고 이값을 판단(<256 or >255)하기까지 12ns 소요된다. 따라서 세그먼트 정보 읽기시간 15ns를 포함하여 대략 27ns의 시간이 소요된다.

(3) 2회 메모리 접속으로 NH을 구하는 경우

2회의 메모리 접속으로 Lookup 완료하는 경우는 2가지로 구분된다. d=0 인 경우(즉, 압축하지 않은 경우)와 d=1로 압축은 하였으나 Map=0000인 경우이다. 우선 d=0인 경우를 살펴보면 그림 17과 같다. Lookup Engine0가 동작하여 세그먼트 정보를 읽고,

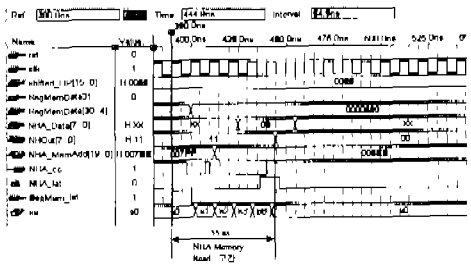

그림 17. 2회 메모리 접속으로 NH를 구하는 경우의 타이밍(d=0)

세그먼트의 포인터와 NHA의 인덱스(LIPAS의 출력)를 더하여 NHA를 읽기 위한 주소를 발생시킨다. 이 주소를 이용하여 해당 NH을 읽어 출력시킨다. 전체 소요 시간은 55ns이다. 따라서 세그먼트 자료 접근 시간 15ns와 합치면 70ns의 시간이 소요된다. 그림 17에서 포인터값이 0x689, LIPAS의 출력이 0x65이므로 NHA 주소는 0x6EE가 된다. 0x6EE번지에 읽혀온 자료가 0x08이고, 이 수치가 바로 NH 정보이다.

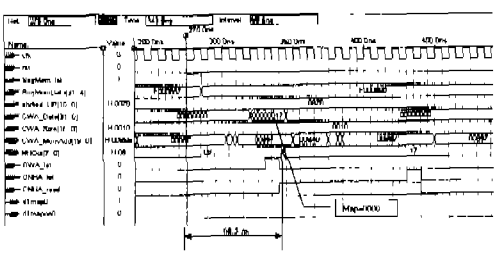

그림 18. 2회 메모리 접속으로 NH를 구하는 경우의 타이밍(d=1)

d=1이면서 2회의 메모리 접근으로 NH을 구하는 경우는 그림 18과 같다. LIPAS 출력(shifted_LIP[15..0])이 0x0078중에 하위 4비트를 제외한 상위비트에 세그먼트 자료 0x890을 더하여 CWA주소 0x897을 발생시키고, 이를 이용하여 CWA 값을 읽은 결과가 Map=0000(=CWA_Data[31..16])이므로 CWA의 하위 16비트(=CWA_Data[15..0], 즉 Base)가 NH 이 된다. 전체 소요 시간은 68.5ns 이다. 세그먼트 접근 시간과 합하면 83.5ns 이다.

(4) 3회 메모리 접속으로 NH을 구하는 경우

3회의 메모리 접속은 d=1이면서 Map이 0000이 아닌 경우이다. 그림 19가 이 경우에 해당한다. CWA 메모리는 CBM(=Map/Base)부분과 CNHA

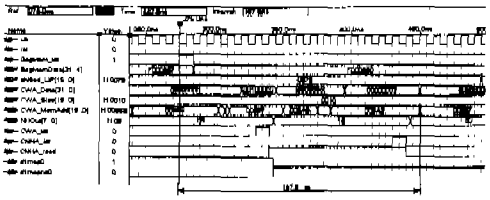

그림 19. 3회 메모리 접속으로 NH를 구하는 경우의 타이밍

부분으로 나누어진다. 우선 CBM을 읽는 부분과 CNHA부분을 읽는 부분으로 나눈다. 먼저 세그먼트 포인터 0x890과 shifted_LIP 0x78을 이용하여 CBM용 주소 0x897을 발생시킨다. 이 주소를 이용하여 Map/Base부분을 읽고(CWA_cs, CWA_lat를 이용), Map이 0000이 아니라는 것(타이밍 상에서는 0x0010)을 판단한 후(CNHA_read = 0), 다시 CNHA 값을 읽는다. Base 값이 0x0002이고 현재의 Map이 0x0010이므로 CNHA주소는 식 (4)에 근거하여 계산하면 0x8A2가 된다. 이 과정에는 16비트 병렬 가산기등과 같은 복잡한 과정 때문에 시간이 많이 소요된다. 결국 167.8ns의 시간이 소요됨을 타이밍도로부터 확인할 수 있다.

V. 결 론

본 연구는 메모리용량을 최소화하고, 하드웨어로 쉽게 구현가능한 새로운 초고속 IP 라우터용 Look-up 알고리즘에 대해 소개한 것이다. 주요 특징으로 첫째, 세그먼트 표에 옵셋 필드를 두고, 이를 근거로 세그먼트별로 NHA의 크기를 가변하여 메모리크기를 줄였고, 둘째, 프리픽스 길이가 긴 경우 압축 알고리즘을 수용하여 메모리 크기를 더욱 줄였다. 셋째, 압축 기법을 적용하여 새로 가공된 정보는 CBM/CNHA에 저장되는데, CBM에서 Map이 $0000_{(16)}$인 경우 Base자리에 다음홉의 포트값을 둠으로서 2회의 메모리 접근의 비율을 극대화하였다. 넷째, 압축 알고리즘의 적용을 결정하는 옵셀의 임계치를 고정하지 않고 현재의 메모리 상태와 프리픽스 길이의 분포등을 고려하여 결정하는 적응형(Adaptive) 기법을 도입하였다. 그 결과 IPMA 웹사이트에서 얻은 실시간 라이팅 표(프리픽스 길이가 대부분 24이내)을 본 기법에 적용한 결과 ~316KB의 메모리가 소요됨을 확인하였고, 이들의 메모리 접근 회수는 2회 이내가 95%로 초고속 스위칭이 가능함을 확인하였다.

제안한 알고리즘을 VHDL로 모델링하여 성능을 검증하였는데, 이는 곧장 하드웨어로 구현 가능하므로 실제적인 문제를 검증할 수 있었다. 10ns의 SRAM과 ALTERA사의 EEPLD 기반의 EPM7192 시리즈에 100MHz 클럭을 적용할 경우를 대비하여 설계하였고, 성능을 확인하였다.

최대 3회의 메모리 접속으로 "다음홉출력정보(NextHop)"를 얻을 수 있는 경우 182ns의 시간이 소요된다. 이는 최악의 경우이고, 전체 유입 IP중에 2회 이내의 메모리 접근으로 NH을 찾을 수 있는 경우가 대부분이며, 이 경우 70ns 혹은 83.5ns 정도가 소요된다. 만약 1회의 메모리 접속으로 NH을 찾는 경우, 소요 시간이 ~27ns이므로 대략 초당 50 x 10^6 번 Lookup이 가능하다. 좀더 빠른 ASIC기법(근래 16개의 1비트 병렬 가산을 10ns 이내로 할 수 있는 장치가 있다)을 이용할 경우 이보다 더 빠른 결과(1Gbps)를 얻을 수 있을 것이다. 현실적으로 200MHz 정도의 클럭에서 동작할 수 있다면(이는 아주 현실적으로 구현 가능하다) 초당 100x10^6 번의 Lookup이 가능하고 이는 비트율로 환산하면 1GBps[6]의 속도와 같다.

참 고 문 헌

[1] R. Jain and S. A. Routier, "Packet trains - Measurement and a new model for computer network traffic," IEEE Journal of Selected Area Communications., Vol, SAC-03, pp. 986-995, Sept. 1986.

[2] Nen-Fu Huang, Shi-Ming Zhao,"A Novel IP-Routing Lookup Scheme and Hardware Architecture for Multigigabit Switching Routers," IEEE Journal of Selected Areas in Communications, Vol. 17, No. 6, June 1999, 1093-1104.

[3] M. Doegermark, A. Brodnik, "Small forwarding tables for fast routing lookups," in Proc. ACM SIGCOMM'97, France.

[4] P. Gupta, "Routing lookups in hardware at memory access speeds," in Proc IEEE INFOCOM'98, Session 10b-1, San Francisco, CA, pp. 1240-1247.

[5] Michigan University and Merit Network, Internet Performance Measurement and Analysis(IPMA) Project [Online], Available

http://nic.merit.edu /~ipma/

[6] Henry Hong-Yi Tzeng, Tony Przygienda, "On Fast Address-Lookup Algorithms," IEEE Journal on Selected Areas in communication, VOL. 17, NO. 6, JUNE 1999.

[7] W. Doeringer, G. Karjoth, and M. Nassehi, "Routing on longest-matching prefixes," IEEE /ACM Trans. Networking, Vol. 4, pp. 86-97

강 숭 민(Seung-min Kang) 정회원

1987년 경북대학교 전자공학과 (학사)

1990년 경북대학교 전자공학과 (석사)

1990~1996년 국방과학연구소 연구원

1996년 3월~현재 : 포항선린대학 정보통신과 전임강사

1999년 : 경북대학교 전자공학과 박사 수료

<주관심 분야> 광통신, 인터넷 라우터, 컴퓨터 네트워크,

숭 재 원(Jac-Won, Song) 정회원

경북대학교 전자공학부 교수

<주관심 분야>: 집적광학, 광통신, 광네트웍