FPGA상에서 은닉층 뉴런에 최적화된 MLP의 설계 방법

경 동 욱[†] · 정 기 철[††]

요 약

일반적으로 신경망은 비선형성 문제를 해결하기 위해서 소프트웨어로 많이 구현되었지만, 영상처리 및 패턴인식과 같은 실시간 처리가 요구되는 응용에서는 빠른 처리가 가능한 하드웨어로 구현되고 있다. 다양한 종류의 신경망 중에서 다층 신경망(MLP : multi-layer perceptron)의 하드웨어 설계는 빠른 처리속도와 적은 면적 그리고 구현의 용이성으로 고정소수점 연산을 많이 사용하였다. 하지만 고정소수점 연산을 사용하는 하드웨어 설계는 높은 정확도의 부동소수점 연산을 많이 사용하는 소프트웨어 MLP를 쉽게 적용할 수 없는 문제점을 가진다. 본 논문에서는 높은 정확도와 높은 유연성을 가지는 부동소수점 연산을 사용하면서도 은닉층 뉴런수를 주기(cycle)로 빠르게 수행하는 MLP의 완전 파이프라이닝(fully-pipelining) 설계방법을 제안한다. MLP는 주어진 문제에 의해서 자연스럽게 입력층과 출력층의 구조가 결정되지만, 은닉층 구조는 사용자에 의해서 결정된다. 그러므로 제안된 설계방법은 많은 반복수행이 요구되는 영상처리 및 패턴인식 등의 분야에서 은닉층 뉴런수를 최적화 하여 쉽게 성능 향상을 이룰 수 있다.

키워드 : 신경망, 다층신경망, 파이프라이닝, 부동소수점 연산, FPGA

MLP Design Method Optimized for Hidden Neurons on FPGA

Dongwuk Kyoung[†] · Keechul Jung[††]

ABSTRACT

Neural Networks(NNs) are applied for solving a wide variety of nonlinear problems in several areas, such as image processing, pattern recognition etc. Although NN can be simulated by using software, many potential NN applications required real-time processing. Thus they need to be implemented as hardware. The hardware implementation of multi-layer perceptrons(MLPs) in several kind of NNs usually uses a fixed-point arithmetic due to a simple logic operation and a shorter processing time compared to the floating-point arithmetic. However, the fixed-point arithmetic-based MLP has a drawback which is not able to apply the MLP software that use floating-point arithmetic. We propose a design method for MLPs which has the floating-point arithmetic-based fully-pipelining architecture. It has a processing speed that is proportional to the number of the hidden nodes. The number of input and output nodes of MLPs are generally constrained by given problems, but the number of hidden nodes can be optimized by user experiences. Thus our design method is using optimized number of hidden nodes in order to improve the processing speed, especially in field of a repeated processing such as image processing, pattern recognition, etc.

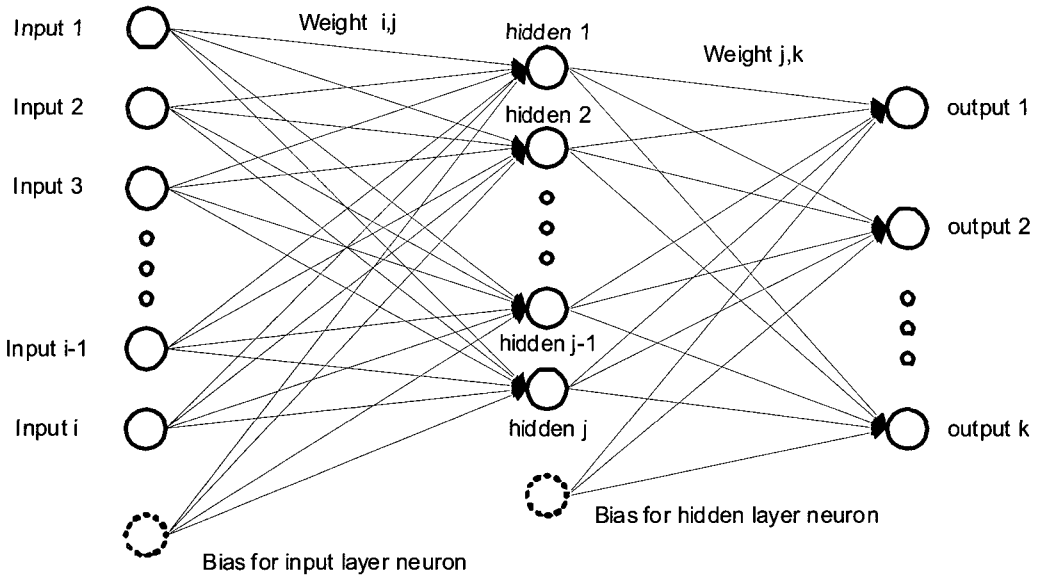
Key Words : Neural Network, MLP, Pipelining, Floating-point Arithmetic, FPGA

1. 서 론

신경망은 기본적으로 고등동물의 두뇌 구조와 그 기능을 본떠 기존의 컴퓨팅 방법으로 해결하기 어려운 문제, 즉 영상처리 및 패턴인식과 같은 비선형성을 가지는 문제에 이용하고자 개발되었지만 단위 시간당 막대한 연산을 수행하는 문제로 실시간 처리가 요구되는 분야에서는 빠른 처리속도를 가지는 하드웨어 구현이 사용되고 있다[1-2].

※ 본 연구는 숭실대학교 교내연구비 지원으로 이루어졌음.
† 준 회 원 : 숭실대학교 IT대학 미디어학부 미디어공학 박사과정
†† 종신회원 : 숭실대학교 IT대학 미디어학부 교수
논문접수 : 2006년 4월 13일, 심사완료 : 2006년 6월 7일

소프트웨어 구현에서는 일반적으로 고정소수점 연산보다 높은 정확도를 가지는 부동소수점 연산을 많이 사용하지만, 하드웨어 구현에서는 적은 면적과 빠른 처리속도 및 구현의 용이성으로 인해서 부동소수점 연산 보다는 고정소수점 연산을 많이 사용한다. 기본적인 MLP(multi-layer perceptron) 처리는 각각의 뉴런이 단위적인 동작을 수행함으로써, 병렬 또는 분산처리의 구현이 용이하며, 병렬화 방법으로는 계층 병렬화(layer parallelism : 각 계층간의 독립적인 수행 방식), 링크 병렬화(link parallelism : 각 뉴런에 연결된 가중치의 독립적인 수행 방식) 그리고 뉴런 병렬화(neuron parallelism : 각 뉴런의 독립된수행 방식) 방법으로 구성된다[3]. MLP의 병렬성을 이용한 연구로써, Xiaobin[4]은 고정소수점 연산을



(그림 1) 2계층 MLP의 구조

바탕으로 병렬성을 고려한 설계방법을 제안하였다. Xiaobin이 제안한 설계방법은 8비트 고정소수점 연산을 사용하여 은닉층은 링크 병렬화, 출력층은 뉴런 병렬화 기법을 혼합하여 적은 면적으로 빠른 처리속도를 가지는 설계방법이다. 고정소수점 연산을 사용하는 하드웨어 MLP는 높은 처리속도를 가지지만, 기존의 부동소수점 연산을 사용하는 소프트웨어 MLP를 하드웨어에 적용하기 위해서는 추가적인 학습이나 데이터 변환 문제 그리고 고정소수점의 표현범위 한계로 인한 정확도 문제를 가진다.

고정소수점의 문제를 해결하기 위해서 부동소수점 연산을 사용하는 연구가 진행되었다[5-8]. 관련 연구로는 높은 정확도를 위해 부동소수점 연산을 적용한 연구[5-6], 32비트의 부동소수점 연산 대신 10비트의 부동소수점 연산을 사용하여 일반적인 32비트 부동소수점 연산보다 빠른 처리를 수행하는 연구[7], 부동소수점 연산과 고정소수점 연산을 병행하여 빠른 처리를 수행하는 연구가 진행되었다[8]. 하지만 부동소수점 연산을 사용하는 하드웨어 MLP는 고정소수점 연산을 사용하는 하드웨어 MLP보다 느린 처리속도로 인해서 실시간 처리가 요구되는 영상처리 및 패턴인식 분야에 적용하기 어렵다. 그 이유는 영상처리 및 패턴인식 분야의 MLP는 많은 반복처리로 수행되기 때문에, 실시간 처리를 위해서는 한번의 MLP 처리시간이 적게 소요되는 설계방법이 요구된다[9-10]. 즉, 실시간 처리를 위한 기존의 소프트웨어 MLP를 데이터 변환 및 정확도 손실 없이 하드웨어로 구현하기 위해서는 32비트의 부동소수점 연산을 사용하면서도 빠른 처리속도를 가지는 설계방법이 필요하다.

본 논문에서는 높은 정확도를 가지는 부동소수점 연산을 사용하면서도 빠른 처리속도(은닉층 뉴런수의 주기로 MLP 처리)를 가지는 MLP의 완전 파이프라이닝(fully-pipelining) 설계방법을 제안한다. 제안된 완전 파이프라이닝 설계방법은 어떤 2계층 MLP라도 파이프라이닝 기술[2] 명령어를 실행하는 과정을 여러 단계로 나누어 여러 명령어를 동시에 실행함으로써 처리속도를 빠르게 하는 기술을 적용하여 은닉층의 뉴런수와 같은 빠른 처리 주기로써 한번의 수행이 완료되며, 은닉층 뉴런수의 주기로 처리되기 위해서 링크 병렬화 및 뉴런 병렬화의 방법으로 설계한다. 이때 완전연결 다층 신경망(fully-connected MLP)의 입력층과 출력층의 뉴런 수는 주어진 문제의 부호화(encoding) 방식에 의해서 자연스럽게 결정되지만, 은닉층의 뉴런수는 숙련된 사용자의 경험과, 반복적인 실험을 통해서 결정한다. 그러므로, 새로운 문제를 해결하기 위한 MLP의 설계시, 제안된 설계방법은 은닉층 뉴런수를 최적화 함으로써 높은 처리속도를 반영한다. 구현결과로써 제안한 완전 파이프라이닝 구조는 기존의 8비트 고정소수점 연산을 사용하는 하드웨어 설계[3]보다 약 3배 빠른 성능을 가지며, 소프트웨어 문자 추출 MLP를 데이터 변환 없이 하드웨어에 적용하여 비교하였을 때 처리속도는 약 11배의 빠른 성능을 가진다. 따라서 제안된 설계방법은 기존의 소프트웨어 MLP를 쉽게 적용할 수 있으며, 높은 정확도로써 반복처리에 효과적인 설계방법이다.

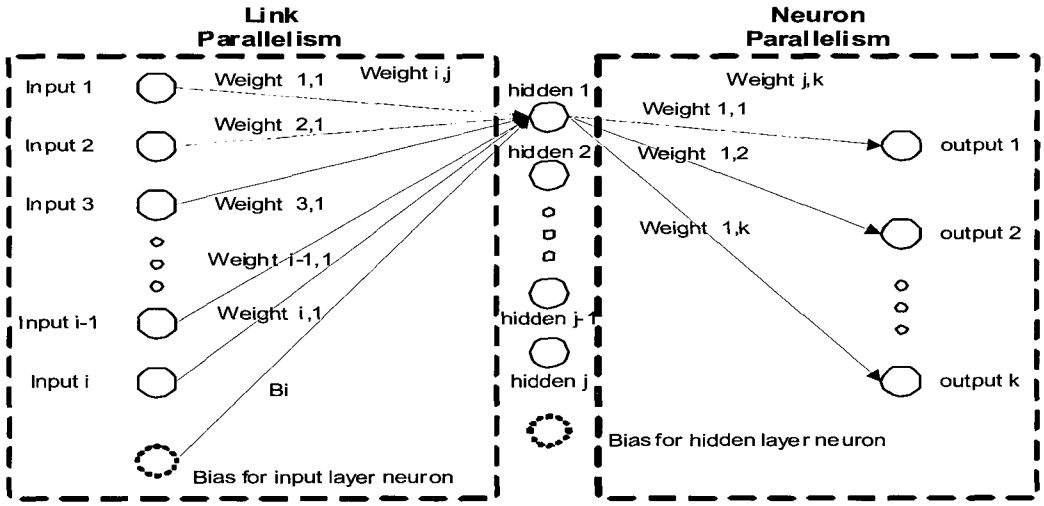
2. 2계층 MLP의 하드웨어 구현

본 장에서는 제안된 MLP의 완전 파이프라이닝 설계방법에 사용된 구조 및 연산자의 특성 그리고 완전 파이프라이닝 설계를 위한 병렬화 기법을 기술한다.

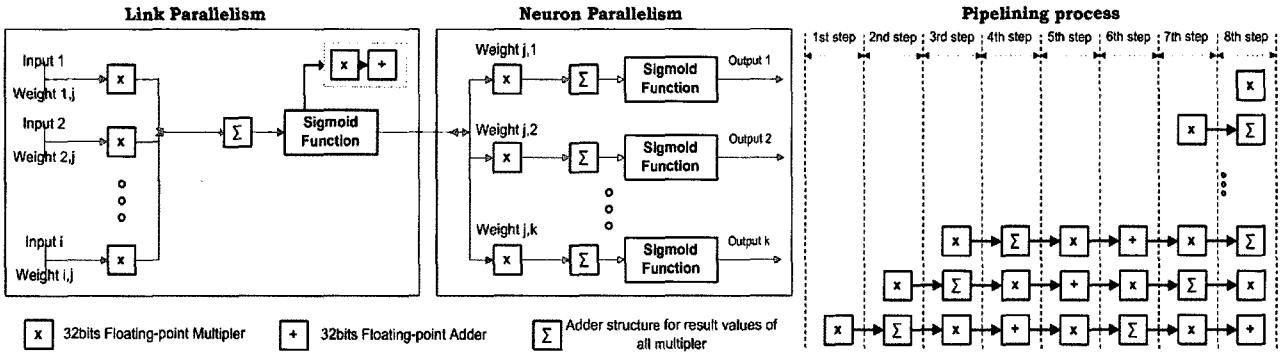
2.1 전체 시스템 구조

일반적인 2계층 MLP의 구조는 입력층, 은닉층 그리고 출력층으로 구성된다(그림 1). MLP의 처리 순서는 입력층 뉴런값과 가중치를 통해서 은닉층 뉴런의 출력값을 계산하며,

1) 명령어를 실행하는 과정을 여러 단계로 나누어 여러 명령어를 동시에 실행함으로써 처리속도를 빠르게 하는 기술



(그림 2) MLP의 병렬처리



(그림 3) 전체 MLP의 병렬화 및 파이프라이닝 구조

계산된 결과값을 이용하여 최종 출력층 뉴런을 처리한다.

은닉층 뉴런은 모든 입력 값과 가중치들을 각각 곱하여 모두 더하며, 더한 결과값을 활성화 함수를 통해서 값이 계산된다(식 1). 또한 하나의 출력층 뉴런은 은닉층 뉴런값과 가중치의 곱셈 및 덧셈 그리고 활성화 함수 순으로 계산한다(식 2). 아래의 식1,2에서 사용된 기호는 입력 값(*input* i), 가중치(*weight* (i, j) : i는 입력 뉴런, j는 은닉 뉴런), 바이어스 값(B), 그리고 활성화 함수($a(x)$)이다.

$$hidden\ \mathrm{j} = a\left(\sum_{n=1}^{i}(weight\ (\mathrm{i},\mathrm{j}) \times input\ \mathrm{i}) + \mathrm{B}\right) \quad (1)$$

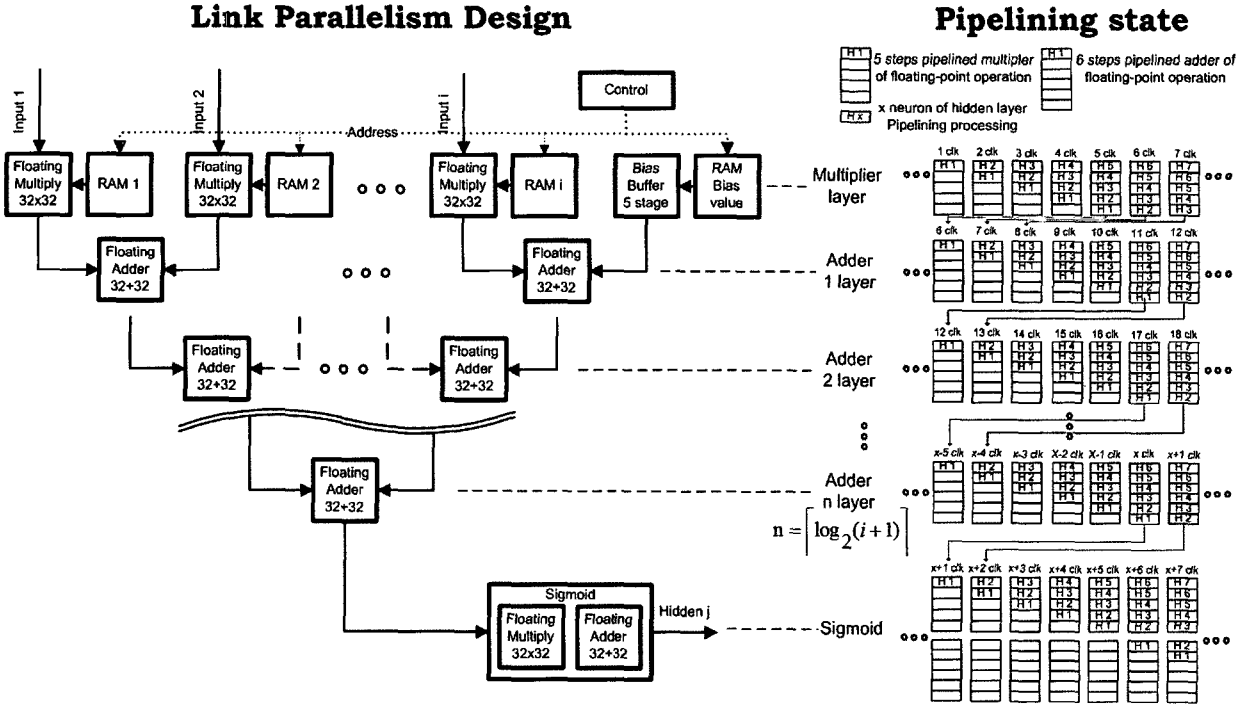
$$output\ \mathrm{k} = a\left(\sum_{n=1}^{k}(weight\ (\mathrm{j},\mathrm{k}) \times hidden\ \mathrm{j}) + \mathrm{B}\right) \quad (2)$$

전체 MLP는 입력층, 은닉층, 출력층의 뉴런수에 따라 처리되는 계산량은 다음과 같다. 하나의 입력층 뉴런이 증가할 때 마다 은닉층 뉴런수만큼의 가중치가 증가하며, 증가된 가중치만큼 곱셈 및 덧셈 연산을 처리해야 한다. 또한 하나의 은닉층 뉴런이 증가할 때 입력층 뉴런수와 출력층 뉴런 수만큼의 가중치가 증가하며, 증가된 가중치만큼 곱셈 및 덧셈 연산을 처리해야 한다. 마지막으로 출력층 뉴런이 하나 증가 할 때 은닉층 뉴런 수만큼의 증가된 가중치를 처리해야 한다. 따라서 MLP는 입력층, 은닉층, 출력층의 규모가 커지면 커질수록 더 많은 계산을 수행한다.

제안된 완전 파이프라이닝 설계는 입력층과 출력층의 규모에 관계없이 은닉층의 뉴런수 주기만으로 MLP가 수행되는 구조이다. 전체 구조는 은닉층의 링크 병렬화와 출력층의 뉴런 병렬화 구조로써, 완전 파이프라이닝 구조로 구현이 용이하다(그림 2). 링크 병렬화는 계층 사이의 가중치들이 병렬적으로 처리되며, (그림 2)에서처럼 하나의 은닉층 뉴런(hidden j)에 연결된 가중치들(Weight i, j)과 입력값(input i)을 파이프라이닝 처리로 1클럭[3] 시스템의 최소 시간단위-파이프라이닝 구조에서 1 단계 처리에 걸리는 시간을 1클럭으로 표현함에 하나의 은닉층 뉴런(hidden j)이 계산된다. 뉴런 병렬화는 1클럭 주기로 처리되는 은닉층 뉴런값(hidden j)과 연결된 가중치(Weight j,k)를 모든 출력층 뉴런(output k)들이 동시에 처리하며, 출력층 뉴런의 결과는 은닉층 뉴런이 모두 처리되어야 하기 때문에 은닉층의 뉴런수와 동일한 클럭 주기로 MLP가 처리된다.

완전 파이프라이닝 설계는 링크 병렬화와 뉴런 병렬화 방법을 사용한 전체 하드웨어 구조이다(그림 3). 제안된 설계 방법에서 사용되는 기본 연산은 부동소수점의 덧셈 및 곱셈

2) 시스템의 최소 시간단위-파이프라이닝 구조에서 1 단계 처리에 걸리는 시간을 1클럭으로 표현함



(그림 4) 은닉층의 링크 병렬화와 파이프라이닝 구조

연산이다. 부동소수점 연산은 고정소수점 연산과 같이 조합회로로 설계하지 않고 순차회로로써 몇 단계로 나누어 처리되는 파이프라인 기법으로 설계하는데, 그 이유는 단순한 구조인 고정소수점 연산은 조합회로 구조일 때 짧은 클럭 주기를 가지지만, 복잡한 구조인 부동소수점 연산은 조합회로 구조일 때 긴 클럭 주기로써 전체 시스템의 성능을 저하시키기 때문이다. (그림 3)의 파이프라이닝 과정(pipelining process)은 전체 MLP의 파이프라이닝 개념을 간략히 나타내는 것으로써, 파이프라이닝 처리방법은 1번째 단계에서 8번째 단계까지 중복 처리됨으로써 전체 처리시간을 줄인다.

부동 소수점은 IEEE 표준[11]으로써 부동소수점 수를 하나의 부호 비트, 바이어스 된 지수(biased exponent)의 8 비트, 그리고 정규화된 가수(mantissa) 또는 유효수(significand)의 23 비트로 표현한다. 부동소수점 수를 사용하는 덧셈은 6단계, 곱셈은 5단계로 구성된 파이프라인 구조이다[12]. 활성화 함수 $a(x)$는 가중치와 입력값을 가지고 덧셈연산과 곱셈연산의 결과값을 수행하여 결과값을 출력한다. 여기서 활성화 함수로 다음과 같은 시그모이드 함수를 사용한다.

$$a(x) = \frac{1}{1+e^{-x}} \tag{3}$$

시그모이드 함수는 지수연산과 덧셈연산 그리고 곱셈연산이 사용되는데 이 연산들은 많은 수행시간과 면적을 소요한다. 이 문제점을 해결하기 위해서 Piece-Wise Linear(PWL) 방식으로 구현하였다[13]. PWL방식은 비선형으로 이루어진 함수를 여러 개의 선형 함수로 처리하는 방식으로써 덧셈연산과 곱셈연산만으로 비선형 함수를 처리할 수 있다. 본 논문에서는 간단한 구현을 위해서 3 segment PWL ($y_3(x)=1$, $y_2(x)=(1/2+x/4)$, $y_1(x)=0$) 방식으로 시그모이드 함수를 구현하였으며, 파이프라이닝 구조의 덧셈연산과 곱셈연산을 사용함으로써 시그모이드 함수 역시 파이프라이닝 구조로 수행한다.

2.2 은닉층의 링크 병렬화 설계방법

링크 병렬화는 각 뉴런에 연결된 가중치의 병렬처리로 수행하는 방식이며, 링크 병렬화 처리를 완전 파이프라이닝 처리로 1클럭에 하나의 은닉층 뉴런을 계산한다. 은닉층의 구조는 가중치와 입력값을 곱셈하는 곱셈 연산층, 곱셈 연산층의 결과를 모두 더하는 덧셈연산층, 그리고 시그모이드 연산층과 같이 3가지층으로 구성된다. 곱셈연산층은 가중치 값을 저장하기 위해서 메모리(RAM)와 함께 구성되며(그림 4), 시그모이드 연산층은 PWL 방법으로써 하나의 덧셈과 하나의 곱셈으로써 구성된다. 곱셈 연산층과 시그모이드 연산층은 MLP 구조에 상관없이 하나의 층을 가지지만, 덧셈연산층은 입력값과 가중치의 곱셈 결과를 파이프라이닝으로 더하기 위해서 여러 계층으로 구성된다(그림 4). 즉, 덧셈층의 수는 입력층 뉴런수(바이어스 포함)를 $i+1$라고 할 때 $n=\lceil \log_2(i+1) \rceil$층으로 구성되기 때문에, 다양한 MLP에 쉽게 설계할 수 있다.

가중치를 위한 램(RAM)의 구성은 입력층 i번째 뉴런과 은닉층 j번째 뉴런의연결된 가중치를 Weight(i,j)로 표현하며, 가중치를 저장하는 각 램은 (그림 5)와 같은 구조로 주소(address)에 맞게 저장된 가중치를 출력한다.

링크병렬화의 파이프라이닝의 처리과정은 (그림 4)의 파이프라이닝 상태(pipelining state)로 나타낸다. 파이프라이닝 과정은 여러 명령을 동시에 처리함으로써 마지막 시그모이

RAM 1

Address	A(1)	A(2)	···	A(j-1)	A(j)
Data	Weight(1,1)	Weight(1,2)	···	Weight(1,j-1)	Weight(1,j)

RAM 2

Address	A(1)	A(2)	···	A(j-1)	A(j)
Data	Weight(2,1)	Weight(2,2)	···	Weight(2,j-1)	Weight(2,j)

⋮

RAM i

Address	A(1)	A(2)	···	A(j-1)	A(j)
Data	Weight(i,1)	Weight(i,2)	···	Weight(i,j-1)	Weight(i,j)

(그림 5) 링크 병렬화 구조에서의 램에 저장된 가중치의 값

드층의 처리과정을 살펴보면 자세히 알 수 있다. 시그모이드층 처리과정은 H1에서 Hk까지의 은닉층 뉴런이 1클럭에 하나씩 순차적으로 처리됨을 보여준다. 초기 수행은 파이프라이닝의 전파지연(propagation deley)으로 많은 클럭이 소요되지만, 두 번째 수행부터는 1클럭에 하나의 수행이 이루어진다. 첫 번째 은닉층 뉴런의 전파지연으로 처리되는 클럭수는 식 4와 같이 x클럭이 소요된다. 식 4에서의 5단계의 mul_step, 6단계의 add_step 그리고 11단계의 sig_step으로써 파이프라이닝 단계수를 나타낸다.

$$x = mul\_step + (n \times add\_step) + sig\_step \qquad (4)$$

다음은 입력층의 뉴런 수에 따른 덧셈층의 설계방법을 2개의 예를 들어 기술한다. 첫 번째는 입력층 뉴런수가 15개 그리고 바이어스의 값을 포함하여 총 16개 일 때, 4개의 층(즉, $4=\lceil \log_2 16 \rceil$)으로 설계된다(그림 6), 두 번째는 입력층 뉴런수가 16개와 바이어스의 값을 포함하여 총 17개 일 때, 5개의 층(즉, $5=\lceil \log_2 17 \rceil$)으로 구성된다(그림 7).

여기서 입력층의 뉴런수와 바이어스의 합이 2^n이면 버(buffer)가 필요 없지만, 그렇지 않을 경우에는 완전 파이프라이닝 처리를 위해서 덧셈 및 곱셈 연산층에서 계산되지 않는 값을 저장하기 각 연산의 파이프라이닝 단계와 같은 구조의 버퍼로 구성된다. 예를 들어, 총 17개의 입력(16개의 입력 뉴런과 1개의 바이어스)일때의 구조는 (그림 7)과 같다. (그림 7)에서 1층9(1layer 9)의 바이어스의 값을 1layer덧셈 연산을 수행하는 동안 저장하며 각 2layer 5, 3layer 3, 4layer 2에서도 마찬가지로 덧셈 연산을 수행하는 동안 결과 값을 유지하기 위해서 버퍼가 필요하며, 각버퍼는 덧셈연산과 같은 6단계로 구성된다.

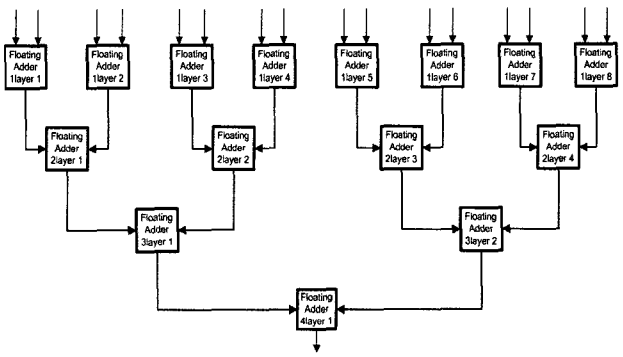
2.3 출력층의 뉴런 병렬화 설계방법

뉴런 병렬화는 하나의 상위층 뉴런과 하위층의 모든 뉴런에 연결된 가중치를 병렬처리 하는 방식으로써, 하위층의 모든 뉴런이 동시에 처리된다. 고정소수점 연산은 입력과 동시에 결과를 출력하는 조합회로(combinational circuit)로써 순차적으로 입력되는 곱셈의 결과와 덧셈의 결과를다시 입력으로 하여 뉴런 병렬화 구조를 쉽게 구현할 수 있었지만, 부동소수점 연산은 입력과 결과를 출력하기 위해서 많은 처리단계를 가지는 순차회로(sequential circuit)로써 기존의 고정수서점에서 사용된 방법으로 구현되기 어렵다. 제안된 뉴런 병렬화 구조는 부동소수점 연산을 사용하면서도 쉽게 완전 파이프라이닝으로 설계할수 있는 방법이다. (그림 8)은 하나의 출력층 뉴런을 위한 뉴런 병렬화 구조를 링크 병렬화 구조와 비교하여 설명한다. 링크 병렬화 구조에서 부동소수점 연산인 곱셈 연산, 덧셈 연산은 여러 개의 연산을 사용하지만, 개선된 뉴런 병렬화 구조는 링크 병렬화의 각 층에 해당하는 곱셈 및 덧셈 연산을 하나씩 사용하여 처리한다(그림 8). 출력층의 뉴런 병렬화 구조에서 버퍼는 상위 계층에서 처리된 결과값을 저장한다. 왜냐하면 곱셈 및 덧셈 연산은 2개의 입력값을 가지고 수행되기 때문에 두 번째 결과값이 입력되기 전에 하나의 결과값을 저장하기 위한 용도이다.

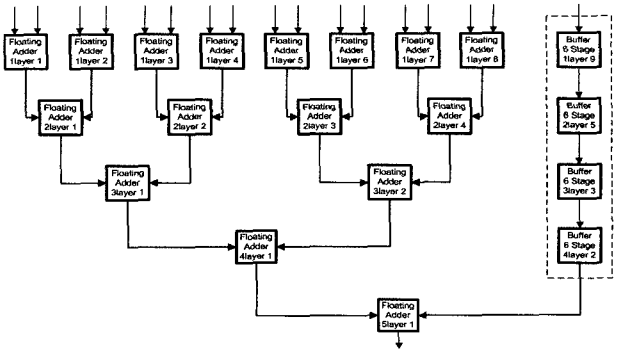
뉴런 병렬화 구조에서 필요한 연산자의 수를 살펴보면 곱셈 연산과 시그모이드 연산이 출력층 뉴런 수만큼 필요하며 덧셈연산은 은닉층의 뉴런의 수와 바이어스를 포함한 수를 $j+1$이라고 할 때 $n=\lceil \log_2(j+1) \rceil$이면 덧셈연산의 수는 $n \times k$개이다. 이때 k는 출력층의 뉴런수이다(그림 9).

전체 출력층 구조에서 사용되는 램은 은닉층 j번째 뉴런과 출력층 k번째 뉴런의 연결된 가중치를 Weight(j, k)로 저장하며, 각주소(address)에 맞게 저장된 가중치를 출력한다(그림 10).

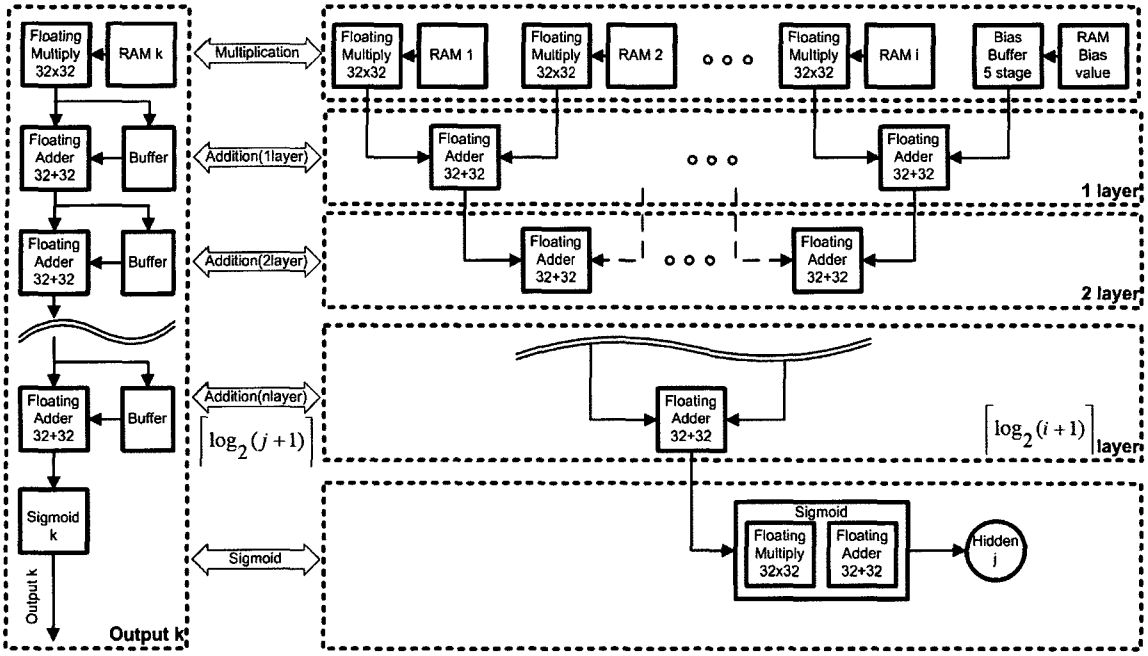
(그림 11), (그림 12)는 개선된 뉴런 병렬화의 파이프라이닝 처리과정으로써, 은닉층 뉴런수의 간격으로 처리됨을 보여준다. (그림 11)은 입력층 뉴런이 15개, 7개의 은닉층 뉴런 그리고 4개의 출력층 뉴런일 때의 처리과정이며, 마지막 층의



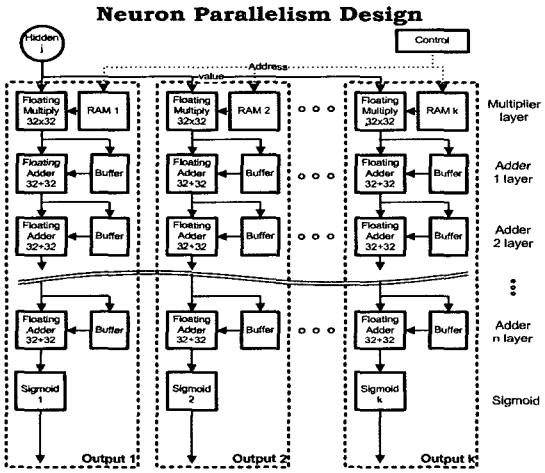
(그림 6) 입력 수 15개 일 때의 덧셈 구조



(그림 7) 입력 수 16개 일 때의 덧셈 구조



(그림 8) 개선된 뉴런 병렬화의 구조



(그림 9) 전체 출력층의 개선된 뉴런 병렬화 구조

RAM 1

Address	A(1)	A(2)	···	A(j-1)	A(j)
Data	Weight(1,1)	Weight(2,1)	···	Weight(j-1,1)	Weight(j,1)

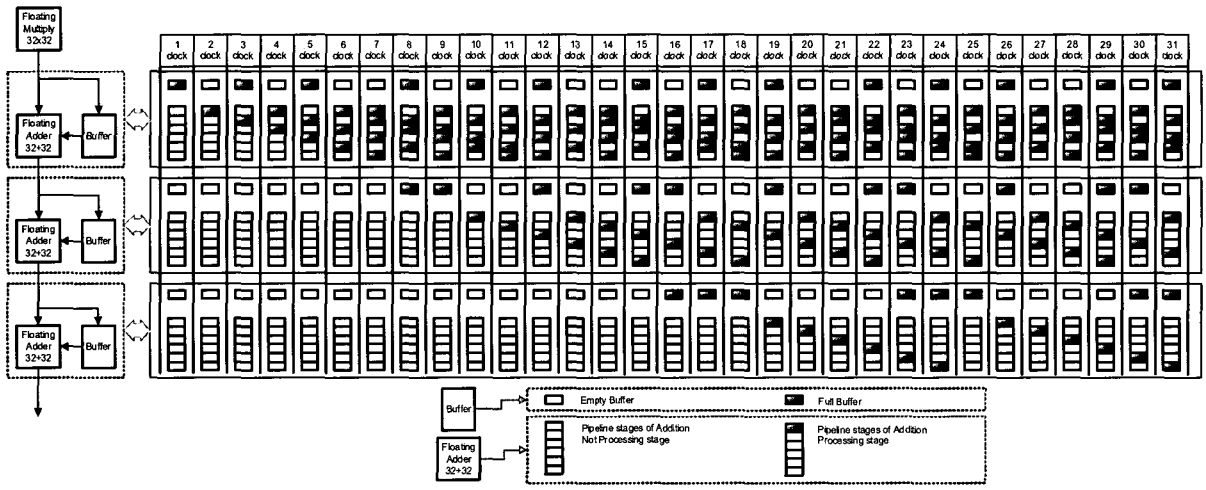
RAM 2

Address	A(1)	A(2)	···	A(j-1)	A(j)
Data	Weight(1,2)	Weight(2,2)	···	Weight(j-1,2)	Weight(j,2)

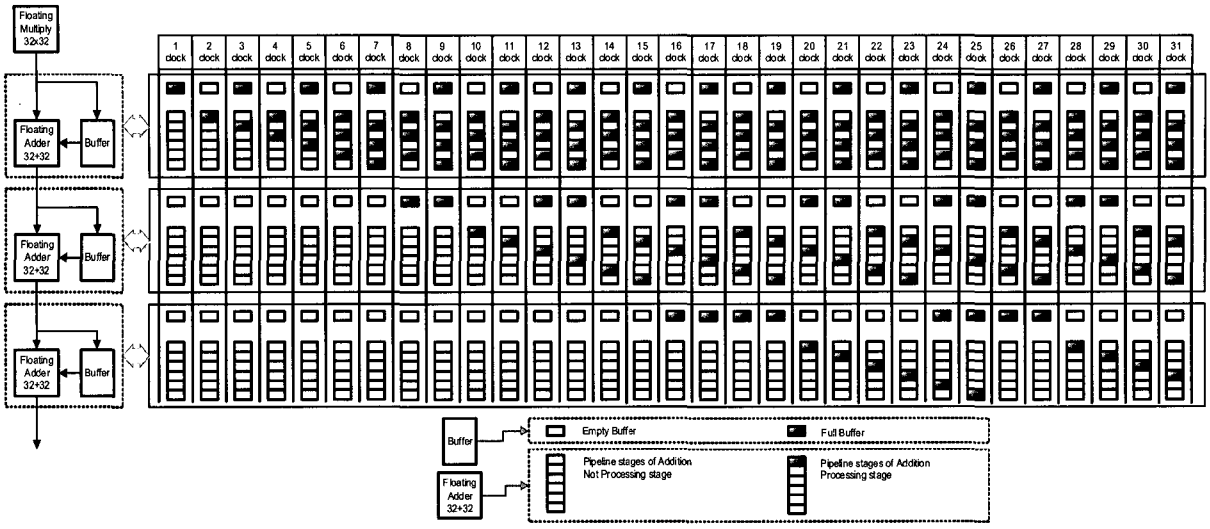
RAM k

Address	A(1)	A(2)	···	A(j-1)	A(j)
Data	Weight(1,k)	Weight(2,k)	···	Weight(j-1,k)	Weight(j,k)

(그림 10) 뉴런 병렬화 구조에서의 램에 저장된 가중치의 값



(그림 11) 은닉층 뉴런이 7개일 때의 파이프라이닝 단계의 상태



(그림 12) 은닉층 뉴런이 8개일 때 파이프라이닝 단계의 상태

〈표 1〉 32bit 부동소수점 연산의 성능 분석

부동소수점 연산	면적(gates)	주기(Frequency)(MHz)	파이프라인 단계	수행시간(ns)
Adder	9,239	90.917	6	65.994
Multiplier	20,655	81.274	5	61.520
Sigmoid	29,941	80.658	11	136.378

〈표 2〉 두 MLP의 성능 분석

MLP	면적(gates)	주기(Frequency)(MHz)	1번째 수행된 단계 수	1번째 수행시간(ns)
A	479,991	56.223	80	1422.880
B	498,824	55.923	87	1555.647

〈표 3〉 완전 파이프라이닝 MLP의 수행 결과

MLP	2번째 수행된 단계 수	3번째 수행된 단계 수	…	n번째 수행된 단계 수	수행시간(ns)
A	7	7	7	7	124.502
B	8	8	8	8	143.048

덧셈연산과 버퍼상태를 살펴보면 한번의 수행은 은닉층 뉴런수 만큼의 클럭(7 클럭 : 19번째 클럭에서 26번째 클럭 간격) 주기로 수행된다.

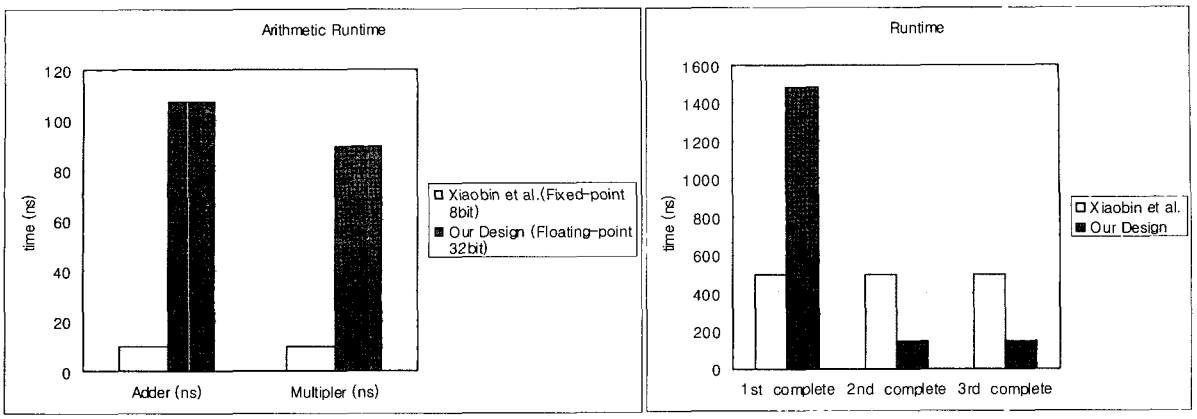
(그림 12)는 16개의 입력 뉴런, 8개의 은닉 뉴런 그리고 4개의 출력 뉴런일 때의 처리과정을 보여주며, 마지막 덧셈층은 은닉층 뉴런 수만큼의 클럭(8 클럭 : 20번째 클럭에서 28번째 클럭 간격) 주기로 수행됨을 보여준다.

(그림 11)과 (그림 12)에서와 같이 제안된 설계방법은 입력층과 출력층의 뉴런수에 관계없이 은닉층 뉴런수의 주기로 수행이 완료된다.

3. 실험과 분석

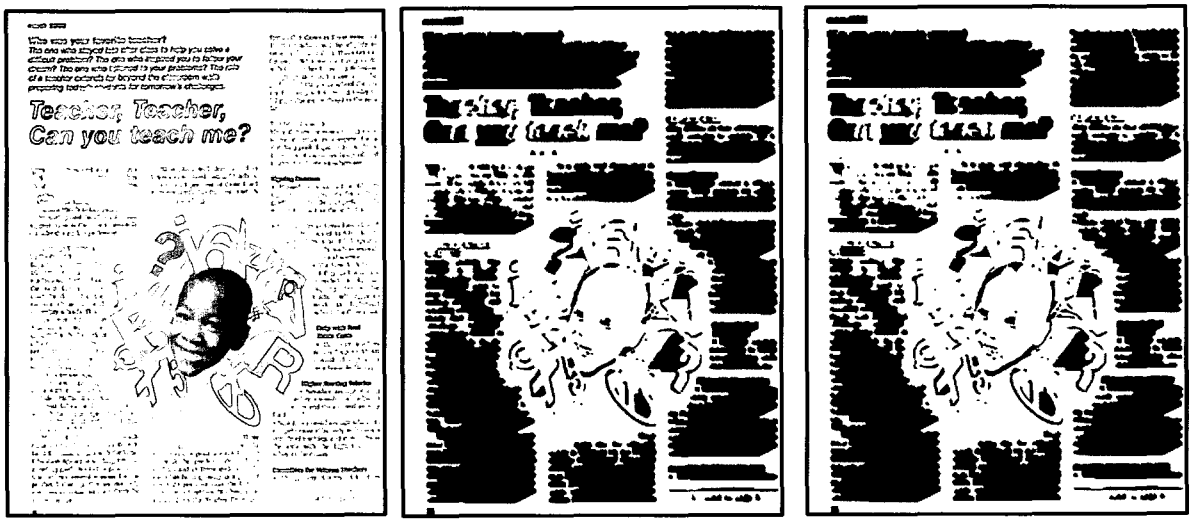
본 논문에서는 VHDL의 합성을 위해서 Xilinx ISE 7.1를 사용하였고, 자이링스(Xilinx)사의 XC2V8000의 FPGA Chip상에서 설계하였다. <표 1>은 완전 파이프라이닝 MLP를 구현할 때 사용된 기본 연산의 성능을 나타낸다. 기본 연산의 처리속도는 전체 시스템의 성능에 많은 영향을 주며, 사용된 부동소수점 연산은 파이프라인 설계로 인해서 빠른 처리속도를 가진다.

본 논문에서 제안된 설계방법을 분석하기 위해서 2개의 MLP를 사용하였다(<표 2>와 <표 3>). 첫 번째 MLP는 15개의 입력, 7개의 은닉층 뉴런 그리고 4개의 출력층 뉴런을 가지며, 두 번째 MLP는 16개의입력, 8개의 은닉층 뉴런 그리고 4개의 출력층 뉴런을 가진다. <표 2>에서 첫 번째 수행은 많은 단계를 거쳐 수행되지만 파이프라이닝 단계를 수행하는 <표 3>에서처럼 2번째부터는 각각 은닉층 뉴런의 수만큼의 단계를 거쳐 수행됨을 볼 수 있다. 즉 첫 번째 수행은 파이프라이닝 구조의 전파지연으로 인해서 MLP(A)는 80클럭을 수행하며, MLP(B)는 87클럭으로 수행이 완료된다.



(a) (b)

(그림 13) Xiaobin의 MLP와 비교



(a) (b) (c)

(그림 14) 실험 결과 : (a) 원본 이미지, (b) CPU상의 결과 그리고 (c) FPGA상의 결과

하지만 두 번째 수행부터는 파이프라이닝 구조로 MLP(A)는 7클럭, MLP(B)는 8클럭으로써 각각 은닉층 뉴런 수만큼으로 수행이 완료됨을 보여준다. 따라서 완전 파이프라이닝 설계방법을 적용하였을 때 적용되지 않았을 때보다 약 10배 정도 빠름을 <표 2>와 <표 3>에서 알 수 있다.

본 논문에서 제안된 설계방법을 평가하기 위해서 2가지 방법을 사용하였다. 첫 번째는 기존의 8비트 고정소수점 연산을 사용하는 설계방법과 비교하였으며, 두 번째는 소프트웨어에서 사용되는 MLP를 하드웨어에 적용하여 비교하였다.

첫 번째 평가방법은 제안된 설계방법과 Xiaobin의 설계방법을 동일한 MLP인 입력층 16개의 뉴런, 은닉층 8개의 뉴런, 그리고 4개의 출력층 뉴런을 가지고 성능을 비교하였다. (그림 13) (a)에서처럼Xiaobin의 8비트 고정소수점 연산인 덧셈기와 곱셈기의 수행 속도는 10(ns)로서 약 10배정도 빠른 성능을 보임을 알 수 있다. 하지만 본 논문에서 제안된 덧셈기와 곱셈기는 32비트 부동소수점 연산을 사용하기 때문에 수행시간은 많이 걸리지만 정확도 면에서는 월등히 높다. 하지만 이렇게 느린 수행속도를 가지는 연산기를 사용하면서도 (그림 13) (b) 에서처럼 첫 수행은 느리지만 2번째 수행부터는 약 3배 정도 빠름을 알 수 있다. 따라서 많은 반복 처리가 요구되는 영상처리 및 패턴인식 분야에서 높은 정확도와 빠른 수행속도로 처리한다.

두 번째 평가방법은 문자추출에 사용된 소프트웨어 MLP를 드웨어 구현에 적용한 결과로써, 많은 반복 처리로 수행되는 패턴인식 및 영상처리에 얼마나 효율적으로 수행되는가를 분석하였다. 사용된 MLP는 2계층으로써, 41개의 입력, 32개의 은닉층 뉴런 그리고 2개의 출력층 뉴런을 가진다. <표 4>는 문자추출 MLP의 성능을 나타내며, 초기 수행은 파이프라이닝 구조의 전파지연으로 많은 시간이 소요되지만, 파이프라이닝 단계를 수행하는 <표 5>에서는 은닉층 뉴런 수와 동일한 주기로 수행됨을 보여준다.

문자추출 MLP를 FPGA와 비교 분석하기 위한 PC의 실험 환경은 펜티엄4 2.4 GHz의 CPU 그리고 메모리 512 Mbyte 이다. 원본 이미지의 크기는 1152 × 1546로써, 1722120번의 반복 수행으로 문자추출이 이루어진다(그림 14). (그림 14)에서 동일한 부동소수점 연산을 사용하면서도 CPU상의 결과와.

〈표 4〉 문자추출 MLP의 성능 결과

MLP	면적(gates)	주기(MHz)	1번째 수행된 단계수	1번째수행시간(ns)
문자추출	1,426,312	54.259	111	2045.73

〈표 5〉 완전 파이프라이닝 MLP의 수행 결과

MLP	2번째 수행된 단계수	3번째 수행된 단계수	…	1722120 번째 수행된 단계수	수행시간(ns)
문자추출	32	32	32	32	589.76

〈표 6〉 FPGA와 CPU상에서의 성능 비교

FPGA	CPU
1.04(s)	11.73(s)

FPGA상의 결과가 다른 이유는 활성화 함수 설계에서 간단한 3PWL의 방식으로 구현되었기 때문이다. 실제 3PWL보다 많은 선으로 구현한다면 CPU상의 결과와 비슷한 결과를 얻을 수 있다. 본 논문에서 제안한 설계방법은 MLP의 전체 구조에 상관없이 은닉층 뉴런수의 주기로 MLP가 처리됨으로써, PC보다 11배의 빠른 성능을 보여준다(<표 6>). 따라서 많은 반복처리를 요구하는 영상처리 및 패턴인식 분야에서 높은 성능으로 수행됨을 알 수 있다.

MLP의 설계시 입력층과 출력층은 해결하고자 하는 문제에 따라 결정되지만, 은닉층은 사용자의 경험 및 실험을 통해서 결정된다. 제안된 설계방법의 실험결과에서처럼 은닉층 뉴런수의 주기로 빠르게 처리됨으로써, MLP 설계시 은닉층을 최적화 하여 설계시 보다 고성능의 MLP 하드웨어를 설계할 수 있다.

4. 결 론

본 논문에서는 높은 정확도와 높은 유연성을 가지면서도 은닉층의 뉴런수의 주기로 빠른 처리가 가능한 MLP의 완전 파이프라이닝 설계방법을 제안하였다. 기존의 부동소수점 연산을 사용하는 하드웨어 MLP는 느린 처리속도로 인해서 실시간 처리를 요구하는 분야에서 사용하기 어려웠다. 하지만 MLP의 완전 파이프라이닝 설계방법은 소프트웨어에서 사용하는 32비트의 부동소수점 연산을 그대로 사용함으로써 높은 유연성과 높은 정확도를 가지며, 개선된 뉴런 병렬화와 링크 병렬화 기법으로써 쉽게 완전 파이프라이닝 구조로 구현할 수 있는 장점을 가진다. 완전 파이프라이닝 설계로 인해서 부동소수점 연산의 처리속도 문제를 은닉층 뉴런 수와 같은 짧은 주기로써 해결하였다. 실험 결과에서 고정소수점 연산을 사용하는 설계방법 보다 약 3배정도의 빠른 처리속도를 가지며, 쉽게 소프트웨어 MLP를 하드웨어에 적용하여 높은 성능을 보여줌으로써, 많은 반복처리를 요구하는 기존의 영상처리 및 패턴인식 소프트웨어 MLP를 높은 정확도와 높은 처리속도로써 수행할 수 있다.

참 고 문 헌

[1] S. Coric, I. Latinovic and A. Pavasovic, "A Neural Network FPGA Implementation," In Proceedings of the 5th Seminar on Neural Network Applications in Electrical Engineering, Belgrade, pp.117-120, Sept. 2000.

[2] Nan Bu, Taiji Hamamoto, Toshio Tsuji and Osamu Fukuda, "FPGA Implementation of a Probabilistic Neural Network for a Bioeletctric Human Interface," In Proceedings of the 47th IEEE Midwest Symposium on Circuits and Systems, Vol.3, pp.29-32, 2004.

[3] Ossoinig, H., Reisinger, E., Steger, C., Weiss, R., "Design and FPGA-Implementation of a Neural Network," In Proceedings of the 7th International Conference on Signal Processing Applications & Technology, pp.939-942, 1996.

[4] Ma Xiaobin, Jin Lianwen, Shen Dongsheng, Yin Junxun, "A Mixed Parallel Neural Networks Computing Unit Implemented in FPGA," In Proceedings of the 2003 International Conferenc on Neural Networks and Signal Processing Vol.1, pp.324-327, 2003.

[5] Sameh A,El Kader, M.S.A.A, "Generic Floating Point Library for Neuro-Fuzzy Controllers based on FPGA Technology," In Proceedings of the 4th IEEE International Symposium on Signal Processing and Information Technology, pp.369-372 Dec. 2004.

[6] Hikawa H, "Pulse Mode Multilayer Neural Network based on Floating Point Number Representation," In Proceedings of the IEEE International Symposium on Circuits and Systems, Geneva, pp.145-148, May, 2000.

[7] Watkins S.S, Chau P.M, "Reduced-complexity Circuit for Neural Networks," Electronics Letters, Vol.31, No.19, pp.1644-1646, 1995.

[8] Wust H, Kasper K, Reininger H, "Hybrid Number Representation for the FPGA-realization of a versatile neuro-processor," In Proceedings of the 24th Euromicro Conference, Vol.2, pp.694-701, 1998.

[9] Keechul Jung, "Neural Network-based Text Localization in Color Images," Pattern Recognition Letters, Vol.22, No.14, pp.1503-1515, 2001.

[10] Keechul Jung and JungHyun Han, "Hybrid Approach to

Efficient Text Extraction in Complex Color Images," Pattern Recognition Letters, Vol.25, No.6, pp.679-699, Apr., 2004.
[11] ANSI and IEEE, "IEEE Standard for Binary Floating-Point Arithmetic," ANSI/IEEE standard, 754-1985, New York, 1985.
[12] Attila Hidvegi, "Implementation of Neural Networks in FPGA," http://www.sysf.physto.se/~attila/ANN.pdf, 2002.
[13] K. Basterretxea, J.M. Tarela and I. del Campo, "Approximation of Sigmoid Function and the Derivative for Hardware Implementation of Artificial Neurons," In Proceedings of the IEE Circuits, Devices and Systems, Vol.151, No.1, pp.18-24, Feb., 2004.

경 동 욱

e-mail : kiki227@ssu.ac.kr
2002년 동의대학교 산업공학과(공학사)
2005년 부산대학교 컴퓨터공학과 (공학석사)
2005년~현재 숭실대학교 IT대학 미디어학부 미디어공학 박사과정
관심분야 : HCI, 영상처리/컴퓨터 비전, 하드웨어 설계, 인공지능, 정보보호

정 기 철

e-mail : kcjung@ssu.ac.kr
1994년 경북대학교 컴퓨터공학과(공학사)
1996년 경북대학교 대학원 컴퓨터 공학과 (공학석사)
1999년 Intelligent User Interfaces group at DFKI(The German Research Center for Artificial Intelligence GmbH), Germany, 방문연구원
2000년 Machine Understanding Division, Electro Technical Laboratory in Japan, 방문연구원
2000년 경북대학교 대학원 컴퓨터공학과(공학박사)
2000년~2002년 PRIP lab, Michigan State University Postdoc
2003년~현재 숭실대학교 IT대학 미디어학부 교수
관심분야 : HCI, 콘텐츠공학, 인터랙티브 게임, 영상처리/컴퓨터 비전, 증강현실, 인공지능