論文96-33B-1-17

분산연산 방식을 이용한 이산시간 Cellular 신경회로망의 하드웨어 구현

(Hardware Implementation of Discrete-Time Cellular Neural Networks Using Distributed Arithmetic)

朴 成 畯 *, 林 俊 皓 *, 蔡 洙 翊 *

(Sungjun Park, Joonho Lim, and Soo-Ik Chae)

요 약

본 논문에서는 이산시간 cellular 신경회로망(DTCNN)의 효율적인 디지탈 하드웨어 구조를 제안한다. DTCNN은 셀간의 연결 형태를 결정하는 템플릿(template)내에서 국소적이며 공간 불변적인 특징을 가진다. 이와 같은 DTCNN의 특징과 분산연산 방식을 결합하여 간단한 하드웨어와 적은 연결선으로 DTCNN 하드웨어를 구현하였다. 또한 분산연산의 특징인 비트별 연산 방식을 사용하여 셀 간의 연결을 위한 넓은 버스 폭을 단일 비트로 줄였다. 본 논문에서는 제안한 구조를 프로그래밍이 가능한 FPGA를 사용하여 가변적인 구조를 갖는 DTCNN 보드로 구현하였다.

Abstract

In this paper, we propose an efficient digital architecture for the discrete-time cellular neural networks (DTCNN's). DTCNN's have the locality and the translation invariance in the templates which determine the patterns of the connection between the cells. Using distributed arithmetic (DA) and the characteristics of DTCNN, we propose a simple implementation of DTCNN. The bus width in the cell-to-cell interconnection is reduced to one bit because of DA's bitwise operation. We implemented the reconfigurable architecture of DTCNN using programmable FPGA.

I. 서 론

기존의 폰-노이만 방식으로는 해결하기 어려운 문제들에 대해 신경회로망의 적합성이 알려짐에 따라 신경회로망에 대한 연구가 활발해지고 그 응용 분야 역시 넓어지고 있다. 그러나 신경회로망의 응용을 제한하는 원인 중의 하나는, 신경회로망에서는 연결 선수가 많아서 이의 집적회로 구현이 어렵다는 점이다. 이런 문제를 해결하기 위해 집적회로로 구현하기 용이한 신경회로망이 많이 제안되었으며, 그 중 대표적인 것이 Cellular Neural Networks(CNN)이다 [1,2]. 기존의 신경회로망들이 많은 연결을 가지고 동작했던 것과는 달리 CNN은 주변의 원소(neighbor cell)와만 연결을 가지면서 주어진 문제의 해를 찾는 장점이 있고, 이러한 CNN의 국소적 연결 특성은 VLSI 구현시 각 셀간의 연결을 쉽게 하고 단위 면적당 셀의 수를 증가시킨다. 또한 CNN의 다른 특징인 템플릿의 공간 불변성은 각 셀들이 모두 같은 모양을 가지게 하므로 layout

* 正會員, 서울大學校 電氣工學部 및 半導體共同硏究所
(School of Electrical Engineering & Inter-University Semiconductor Research Center Seoul National University)

接受日字: 1995年5月25日, 수정완료일: 1995年12月9日

이 규칙적이다. 이와 같은 특성을 가지는 CNN은 다른 신경회로망의 경우에 비해 하드웨어 구현이 매우 용이하며 이의 응용 및 하드웨어 구현에 대한 연구가 활발히 진행 중에 있다[1,2].

CNN은 구조에 따라 연속시간(Continuous-Time, CTCNN)과 이산시간(Discrete-Time, DTCNN) 방식으로 나뉜다. 지금까지의 구현 예는 CTCNN과 DTCNN의 두 구조 모두 캐패시터, 저항 및 비선형 전류원등을 사용하는 아날로그 방식을 이용하였다 [3]. 아날로그 기법을 사용한 하드웨어는 작은 면적과 빠른 연산 속도 등의 장점을 가지고 있으나, 하드웨어의 성능이 공정 변화에 민감하며, 아날로그 신호를 사용하는 인터페이스 방식으로는 보다 큰 시스템으로의 확장이 어렵고, 내부 셀의 구조가 고정되어 있어 응용에 따라 템플릿의 크기를 가변할 수 없는 단점이 있다. 이러한 기존에 구현된 하드웨어의 단점을 극복하기 위하여 본 논문에서는 분산연산 방식과 FPGA의 유연성(flexibility)을 이용한 DTCNN의 디지탈 하드웨어 구현 방법을 제안한다.

FPGA는 디지탈 로직의 시제품이나 값싼, 저 수량 상용제품을 손쉽게 제작하는 데에 사용되며, 특히 데이타-패스와 같이 규칙적인 구조를 갖는 시스템 구현에 매우 효율적이다. 또한 FPGA의 프로그래밍 가능성은 완전 주문형(full-custom) 설계가 줄 수 없는 유연성을 제공한다. 그러나 FPGA는 구현할 수 있는 로직 크기가 완전 주문형 설계에 비해 매우 작고, 입출력 핀의 수와 로직 간의 연결선 수가 제한되는 단점이 있다. 많은 양의 하드웨어를 필요로 하는 디지탈 구현 방식의 단점과 FPGA 구현시의 자원(resource) 제한을 해결하기 위해, 제안된 구조에서는 DTCNN의 국소적 연결 특성을 이용하여 셀간 필요한 연결 수를 줄였다. 또한 분산연산 방식을 채용하여 간단한 하드웨어로 셀을 구현하여 사용가능한 셀 수를 늘렸으며, 분산연산 방식의 특징 중의 하나인 비트별 연산 특성은 디지탈 하드웨어 구현 방식의 단점인 셀간 연결을 위한 넓은 버스 폭을 하나로 줄이며, 보다 큰 시스템으로의 확장을 용이하게 한다.

본 논문은 다음과 같이 구성되어 있다. DTCNN 모델에 대해 2장에서 간략히 기술하였으며, 3장에서는 분산연산 방식을 도입하여 DTCNN을 구현할 때 얻는 장점을 설명하였다. 4장에서는 DTCNN셀의 하드웨어 구현 방법과 FPGA를 이용하여 제작된 보드에 대해 기술한 후 5장에서는 결론을 제시하였다.

Ⅱ. DTCNN 모델

DTCNN은 기본적인 연산을 담당하는 셀로 이루어진 2차원적 배열로 이루어져 있으며, 각 셀들은 단지 주변의 셀과 연결을 갖는 국소적 연결 구조를 가진다. 또한 셀들은 그들의 위치와 관계없이 인접한 셀들과 모두 같은 연결 패턴(템플릿)을 가지는 규칙적인 구조를 이룬다. 2차원 DTCNN에서 i번째 행과 j번째 열에 위치한 셀을 C(i,j)라 하면 C(i,j)로부터 이웃 거리 r을 가지는 이웃구조(neighborhood)는 $N_r(i,j)=\{C(k,l),\ |i-k|\le r \text{ and } |j-l|\le r\}$로 정의된다. 각 DTCNN 셀은 다음과 같은 세개의 변수를 갖는다:

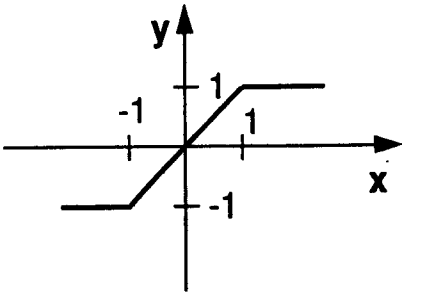

그림 1. Piecewise 선형 함수
Fig. 1. Piecewise linear function.

- 셀 상태: $x_{ij}(n)$, 시간의 함수로써 셀의 에너지 정보를 나타낸다.
- 셀 출력: $y_{ij}(n)$, 자신과 이웃구조 안의 다른 셀의 상태에 영향을 미치며, 셀 상태 $x_{ij}(n)$로 부터 그림 1에 나타낸 것과 같은 piecewise 선형 함수 $f(x_{ij})$를 통과하여 얻어진다.

$$y_{ij}(n)=f(x_{ij}(n))=\frac{1}{2}(|x_{ij}(n)+1|-|x_{ij}(n)-1|) \quad (1)$$

- 셀 입력: u_{ij}, 각 셀에 대한 외부의 여기(excitation) 정도를 나타낸다.

DTCNN 모델에서 각 셀의 동작은 다음과 같은 시스템 방정식에 의해 결정된다.

$$x_{ij}(n+1)=I_{ij}+\sum_{k,l:C(k,l)\in N_r(i,j)}[\,A(i-k,j-l)y_{kl}(n)+B(i-k,j-l)u_{kl}\,] \quad (2)$$

식 (2)에서 A(i-k,j-l)와 B(i-k,j-l)는 출력과 입력에 대한 가중치이며, $(2r+1)\times(2r+1)$ 크기의 행렬로

서 각각 되먹임, 제어 템플릿이라 부른다. 이들 템플릿 모두는, 계수들이 모든 셀에 대해서 동일한 값을 갖는 공간 불변성의 성질을 가지고 있으며 옵셋 I_{ij}와 더불어 신경망의 입출력 연산을 제어한다. 사용되는 템플릿 계수는 각 응용에 따라 달라질 수 있다.

CTCNN모델에서 셀 상태 $x_{ij}(n)$는 최대, 최소 값에 제한이 없고, 일반적인 경우 1보다 큰 5~10 정도의 동작 영역 값을 가지므로 이를 하드웨어로 구현하기가 어려운 점이 있다 [1]. 그러나 DTCNN에서는 셀의 다음 상태 $x_{ij}(n+1)$가 현재 셀 상태 $x_{ij}(n)$ 값에는 의존하지 않기 때문에 이를 저장할 필요가 없고, 단지 동적 영역이 [-1, 1]로 제한되어 있는 출력 값 $y_{ij}(n)$만을 저장하면 되므로 하드웨어 구현시 셀의 복잡도가 크게 줄어든다.

Ⅲ. 분산연산 구조

1. 기본 이론

분산연산 방식은 내적 계산을 위한 구조로서 [4], 본 절에서는 간단하게 그 동작원리를 기존의 곱셈기를 사용하는 곱셈-누적(multiplication-accumulation) 방식과 비교하여 다룬다. 크기 N의 내적을 계산할 경우 내적 y는 식 (3)과 같이 표현된다.

$$y=\sum_{i=1}^{N} a_i x_i = a_1x_1+a_2x_2+,,,+a_Nx_N \qquad (3)$$

여기서 a_i는 고정 계수이며, x_i는 크기 B의 워드길이로 표현된다. x_i를 2의 보수 형태로 표현하면 다음과 같다. 여기서 x_{i0}는 MSB로 부호비트이다.

$$x_i=-x_{i0}+\sum_{j=1}^{B-1} x_{ij}2^{-j} \qquad (4)$$

식 (4)를 (3)에 대입하면 내적 y는 다음과 같이 표현된다.

$$y=\sum_{i=1}^{N} a_i\left[-x_{i0}+\sum_{j=1}^{B-1} x_{ij}2^{-j} \right] \qquad (5)$$

식 (5)는 곱셈기를 사용하여 내적을 계산할 때의 표현 방법이다. 일반적인 내적 계산에서는 그림 2에서 나타낸 것과 같이 x_i와 고정 계수 a_i를 곱하고 난 후, 이를 누적기를 사용하여 덧셈을 수행한다.

식 (5)에서 덧셈의 순서를 바꾸어 쓰면 식 (6)을 얻을 수 있으며, 이는 분산연산 방식을 나타낸다.

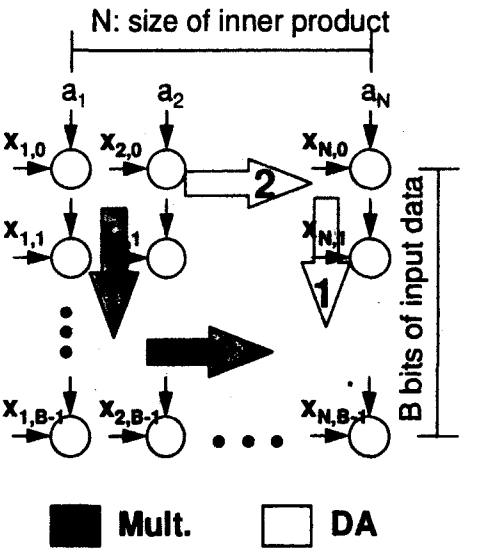

그림 2. 내적 계산을 위한 흐름도
Fig. 2. Data flow graph for the inner product.

$$y=\sum_{j=1}^{B-1}\left[\sum_{i=1}^{N} a_i x_{ij} \right] 2^{-j}-\sum_{i=1}^{N} a_i x_{i0}=\sum_{j=1}^{B-1} y_j 2^{-j}-y_0 \qquad (6)$$

분산연산 방식에서는 먼저 x_i의 각 비트 x_{ij}별로 y_j를 계산하고, 그 후에 비트별 부분곱 y_j를 차례로 더한다. 그림 3에는 내적 계산을 위한 분산연산 방식의 구조가 간략하게 도시되어 있다. 계수 a_i는 고정된 값을 갖고 x_i의 각 비트 x_{ij}은 단지 '0' 또는 '1'의 값만을 가지기 때문에, 식 (6)에서 부분곱 y_j는 2^N가지의 가능한 값을 가질 수 있다. 내적 계산시 매번 부분곱의 값을 계산하기보다는 미리 이 값들을 모두 계산하여 메모리에 저장한다. 고정 계수 a_i가 크기 K의 워드 길이로 표현될 때, 사용되는 메모리는 $K+\log_2(B)$ 넓이와 2^N의 깊이를 가진다. 동일한 위치 'j'를 갖는 N개의 각 비트 x_{ij}를 메모리에 대한 주소로 사용하여 메모리 데이타, 즉 미리 계산된 부분곱을 읽어들여 누적기에 전달하고, 쉬프트 연산 '2^{-j}'이 수행된 누적기의 이전 출력과 메모리 출력 y_j과의 덧셈을 수행한다. x_{ij}는 메모리 주소로 사용하기 위해 LSB부터 한번에 한 비트씩 메모리에 전달된다. 최종 전달 비트인 MSB가 주소로 사용될 때는 식 (6)의 $-y_0$항을 계산(뺄셈)하기 위해 제어 신호 s가 이 경우에만 '1'의 값을 갖고 다른 경우에는 '0'의 값을 가진다. y_0와 y_j의 값을 누적하기 위해 제어 신호 s를 메모리로 부터의 부분곱과 XOR시켜 누적기에 전달하

며 누적기는 B 사이클만에 내적값 y를 출력한다.

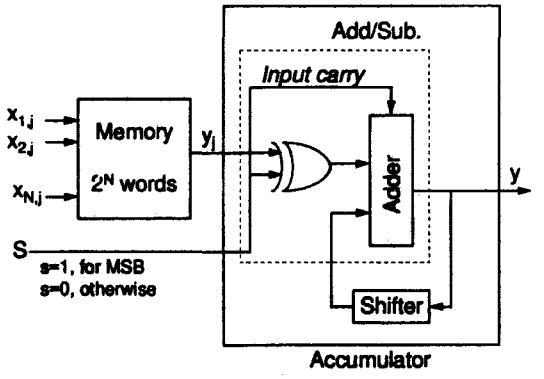

그림 3. 분산연산 구조
Fig. 3. Distributed arithmetic structure.

2. DTCNN에 분산연산 방식 채용시의 장점

국소적 연결 특성: 분산연산 방식을 사용시 필요한 메모리의 크기는 내적의 크기, 즉 템플릿 계수의 수 N에($N=(2r+1)^2$)의해서 결정된다. N이 선형적으로 증가함에 따라 필요한 메모리 크기는 2^N로 지수 함수적으로 증가한다. 그러므로 다른 종류의 일반적인 신경회로망에 비해 상대적으로 연결의 수가 적은 국소적 연결 특성을 갖는 DTCNN은 필요한 메모리의 크기가 작아 기존의 곱셈기 사용방법 대신 분산연산방법을 사용하는 데에 유리하다.

공간불변성: DTCNN의 중요한 연결 특성 중의 하나는 템플릿의 공간불변성으로 이는 각각의 모든 셀들이 위치에 관계없이 동일한 템플릿을 가지는 성질이다. 제안한 구조에서는 템플릿의 계수들의 부분곱이 메모리에 미리 저장되므로, 동일한 템플릿을 가지는 각 셀의 메모리는 이 특성으로 인해 모두 같은 내용을 저장한다. 그러므로 셀의 하드웨어 구현시, 다중 읽기 포트(multi-read port)메모리를 이용하여 그 내용을 각 셀 들이 공유할 수 있으므로 하드웨어의 복잡도를 크게 줄일 수 있는 장점이 있다.

비트별 연산: 신경회로망을 디지탈 기법을 이용하여 구현하고자 할때 큰 단점 중의 하나가 셀과 셀의 연결을 위해서 입출력의 워드길이에 따르는 넓은 폭의 버스가 필요한 점이다. 그러나 제안한 구조에서는 분산연산 방식의 특징 중의 하나인 비트별 연산을 수행하기 때문에 셀간의 연결을 위해 단지 한 선만 필요하며, 이는 VLSI 구현시 셀의 면적을 매우 작게 할뿐만 아니라, 배선을 위한 연결 선수가 제한되어 있는 FPGA사용시 매우 큰 장점이 된다. 또한 셀 간의 연결 선수가 적으므로 큰 시스템으로의 확장이 용이하다.

Ⅳ. DTCNN의 하드웨어 구현

1. 셀의 설계

내적 계산(ΣAy): 그림 4는 DTCNN셀에 대한 블록도를 나타낸다. 식 (2)의 오른쪽 항인 ∑Ay는 누적기, 쉬프터 및 메모리로 구성된 분산연산 블록을 사용하여 계산된다. 기존의 분산연산 방식에서 메모리로 ROM을 사용하는 것과는 달리 DTCNN셀에서는 여러 응용에 따라 템플릿의 계수를 변화시켜야 하므로, 이에 대한 프로그래밍 기능을 지원하기 위해 SRAM을 사용하였다. 분산연산 방식을 단일 메모리로 직접 구현시 2^N 워드 크기의 메모리가 필요하다. 예를 들어 N=9 (r=1)일 때 2^9=512 워드 크기의 메모리가 필요하다. 이는 다른 신경회로망에 비해 연결 수가 적은 DTCNN의 경우라도 매우 큰 값이며, 특히 내부에 메모리 구현이 효율적이지 않은 FPGA구현에서는 매우 치명적인 단점이 된다. 그러므로 메모리를 분할하는 방법을 사용하여 요구되는 메모리의 크기를 줄여야 한다. N(=M·L)개의 계수를 M개씩 묶어 L로 분할하면, 전체적으로 필요한 메모리는 2^N 크기의 한 개로부터 2^M 크기의 L개의 메모리가 된다.

$$\text{절감되는 메모리 비율} = \frac{\text{분할기법 사용시 메모리크기}}{\text{단일 메모리 사용시 메모리크기}} = \frac{L\cdot 2^M}{2^{M\cdot L}} = \frac{L}{2^{(M-1)\cdot L}} \quad (7)$$

그림 5는 r=1일 때의 되먹임 템플릿 A에 대한 분할 방법을 도시하고 있다. 되먹임 템플릿은 세 개의 영역(좌측,가운데,우측)으로 나뉘며 각 영역은 3개의 템플릿 계수로 이루어져 있다. 각 영역에 대한 메모리는 3개의 계수들에 대한 부분곱 $d_{ij,l}$, $d_{ij,c}$ $d_{ij,r}$만을 저장하고, 이를 외부의 덧셈기를 통해서 연산하여 최종적으로 d_{ij}를 생성한다. 이와 같은 방법을 통해 필요한 메모리 양은 512 워드에서 3 x 2^3 = 24워드가 되어 메모리 양이 단일 메모리 구현 방식에 비해 5%로 대폭 감소한다.

분산연산 방식의 누적기는 덧셈과 쉬트프 연산 '2^{-j}'을 수행한다. 쉬프트 연산과 저장기능은 간단한 쉬프트 레지스터로 구현되며, 덧셈 연산을 구현하기 위해서는 carry-save adder(CSA)를 사용하였다. 내적 계산과

같이 연속적으로 덧셈이 수행될 때 다른 방식의 덧셈기를 사용하면 매번 carry가 전파되므로 덧셈기의 동작속도를 크게 제한한다. 그러나 CSA에서는 매 단계마다 carry를 전파시키지 않고 마지막 단계에서만 전파되는 carry를 CPA(carry propagation adder)로 처리하므로, 내적 계산과 같은 다중 연산자를 위한 덧셈에서 매우 효율적인 방법이다.

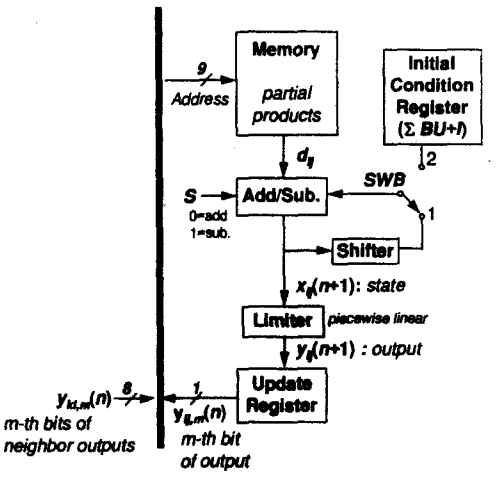

그림 4. DTCNN 셀의 블록도
Fig. 4. Block diagram of DTCNN cell.

ΣBU+I의 다운로드: 식 (2)에서 사용된 B, U와 I는 시불변한 값을 가지기 때문에 ΣBU+I값을 매번 계산할 필요가 없이, 호스트 컴퓨터에서 미리 이 값을 계산하여 초기 상태에서 ICR(Initial Condition Register)에 저장한 후 필요할 때마다 ICR로부터 ΣBU+I 값을 로드한다. 또한 식 (5)에서 보는 바와 같이 분산연산 방식을 통해 계산한 ΣAy와 ICR에 미리 저장되어 있는 ΣBU+I를 더하기 위해서는 추가의 덧셈기가 필요지만, 제안한 구조에서는 추가의 덧셈기 대신 간단한 스위치를 사용하여 첫번째 사이클에서 '0' 값의 누적기 출력 대신 스위치 SWB를 '2'에 위치시켜 ICR에 저장되어 있는 ΣBU+I의 값을 메모리 내용과 더하게 함으로써 추가의 덧셈기가 필요 없다. 그 후에는 SWB가 '1'에 위치하여 메모리로부터 오는 부분곱과 쉬프트된 누적 값을 더하는 연산을 수행한다.

Piecewise 선형 함수: 시그모이드 함수와 같은 비선형함수를 디지탈 방법으로 구현하기 위해서는 look-up table방식이나 직접 곱셈방법을 통해 이를 구현해야 하므로 많은 양의 하드웨어가 소모된다. 그러나 DT-CNN에서는 비선형함수로서 그림 1과 같이 x가 [-1, 1] 영역에서는 원점을 통과하며 기울기가 '1'이고 그 외의 영역에서는 -1, 또는 +1의 값을 갖는 piecewise 선형 함수를 사용한다. 제안된 구조에서 이는 입력 값이 [-1, 1] 영역에 있으면 그대로 출력으로 통과시키고, 이를 넘으면(overflow) +1, 또는 -1의 값을 출력하는, 그림 6과 같은 간단한 overflow검사 로직과 MUX로 이루어진 limiter로 구현된다.

주소 생성: 분산연산 방식에서 내적 계산을 위해서는 입력 값의 워드길이 B만큼의 클록 사이클이 필요하며, 이를 위해 매 사이클마다 동일한 위치를 가지는 비트로 이루어진 주소가 필요하다. ΣAy의 계산을 위해 분산연산 방식을 사용하는 DTCNN셀에서는 다음 상태의 출력 $y_{ij}(n+1)$을 계산하기 위해서 셀 자신의 출력 $y_{ij}(n)$과 이웃 구조 안의 주변 셀의 출력들, $y_{kl}(n)$이 비트별로 LSB부터 MSB까지 순차적으로 SRAM의 주소로 사용된다. B사이클이 지난 후에는 내적 ΣAy의 계산이 완료되어 새로운 출력 값 $y_{ij}(n+1)$이 갱신 레지스터(update register)에 저장되고, 이는 다시 다음 상태 $y_{ij}(n+2)$ 계산을 위해 LSB부터 비트별로 출력되어 자신과 이웃구조에 포함되어 있는 인접 셀을 위한 주소로 사용된다. 이러한 역할을 담당하는 갱신 레지스터는 계산된 현재 상태 출력을 저장하기 위한 병렬 입력 기능과 다음 상태 계산을 위한 주소 생성을 위한 비트별 직렬 출력 기능을 가지고 있다.

표 1. DTCNN 템플릿
Table 1. DTCNN templates.

	A	B	I	Input State	Border Cells
Noise Removal	$\begin{bmatrix}0&1&0\\1&2&1\\0&1&0\end{bmatrix}$		0	x(0)=image u=don't care	x=0 u=don't care
Hole Filler		$\begin{bmatrix}0&0&0\\0&4&0\\0&0&0\end{bmatrix}$	-1	x(0)=image u=don't care	x=0 u=don't care
Corners Extraction	$\begin{bmatrix}0&0&0\\0&2&0\\0&0&0\end{bmatrix}$	$\begin{bmatrix}-0.25&-0.25&-0.25\\-0.25&2&-0.25\\-0.25&-0.25&-0.25\end{bmatrix}$	-2.8	x(0)=image u=don't care	x=don't care u=-1
Corners Extraction	$\begin{bmatrix}0&0&0\\0&2&0\\0&0&0\end{bmatrix}$	$\begin{bmatrix}-0.25&-0.25&-0.25\\-0.25&2&-0.25\\-0.25&-0.25&-0.25\end{bmatrix}$	-1.8	x(0)=image u=image	x=don't care u=-1
Shadow Detector	$\begin{bmatrix}0&0&0\\0&2&2\\0&0&0\end{bmatrix}$	$\begin{bmatrix}0&0&0\\0&2&0\\0&0&0\end{bmatrix}$	0	x(0)=image u=image	x=0 u=don't care
Connected Component Detector	$\begin{bmatrix}0&0&0\\1&2&-1\\0&0&0\end{bmatrix}$		0	x(0)=image u=image	x=±1 u=don't care
Halftoning [6] 참조	-	-		-	-

2. DTCNN 구현

제안된 구조는 FPGA 칩을 사용해 구현되었다. 한

셀을 구현하기 위해서는 덧셈기, 레지스터, limiter, 메모리, XOR 게이트가 필요하며, FPGA칩에는 여러 개의 DTCNN의 셀이 포함되어 있다. 한 칩에 포함되는 셀의 수는 각 셀의 입출력 워드길이에 의해 결정된다. 표 1의 각 응용에 적합한 셀을 구현하기 위해 소요되는 게이트 수와 Xilinx로 구현시 필요한 CLB (Combinational Logic Block)수를 표 2에 나타냈다. 특히 halftoning 에에서는 시간에 따라 템플릿의 값이 바뀌며 응용에 따라 이웃 거리 r이 가변 되므로 기존의 하드웨어로는 구현하기가 어렵다 [6]. 제안한 구조에서는 시변적인 템플릿 계수를 미리 메모리에 저장하는 분산연산 방식과 r에 따라 내부 셀 구조를 쉽게 가변할 수 있는 FPGA의 프로그래밍 특성을 이용하여 이러한 응용을 쉽게 구현할 수 있다.

표 2. DTCNN셀 구현을 위한 게이트 수와 CLB수

Table 2. Gate and CLB count for various DTCNN cells.

	Gate Count	CLB Count(XC 4010)	Cell Resolution
Connected Component Detector	400	23 CLBs	4 bit
Shadow Detector	502	26 CLBs	4 bit
Border Extractor	1102	58 CLBs	8 bit
Corder Extractor	1200	62 CLBs	8 bit
Hole Filling	704	37 CLBs	4 bit
Noise Removal	1400	69 CLBs	8 bit
Halftoning (r=1)	1706	83 CLBs	8 bit
(r=2)	2304	104 CLBs	8 bit

3. FPGA 보드 구현

그림 7은 DTCNN FPGA 보드의 전체구조를 나타낸다. 호스트 컴퓨터는 전체 동작을 제어하고 FPGA 칩에 셀을 구현하기 위한 프로그래밍 정보를 CLB에 다운로드하며, DTCNN의 초기값 y(0), ΣBU+I 및 분산연산을 위한 되먹임 템플릿 계수에 대한 부분곱을 각 레지스터 및 메모리에 저장한다. 그림 8은 제작된 보드를 나타내며, 제작된 보드는 상용 DPS-1 보드위에 Xilinx 4010칩을 실장하여 구현하였다 [7]. DPS-1보드는 Sbus 접속을 위한 기능 블록을 가지고 있으며, 이를 통해 호스트 컴퓨터인 *SUN/SPARC-Classic*이 FPGA칩과 쉽게 통신하도록 돕는다. 모든 시스템 설계는 Synopsys 소프트웨어를 이용하여 설계 검증하였으며, FPGA구현을 위해서는 Xilinx XACT 소프트웨어를 사용하였다. 구현된 FPGA 보드는 10 MHz 속도에서 성공적으로 동작하였다.

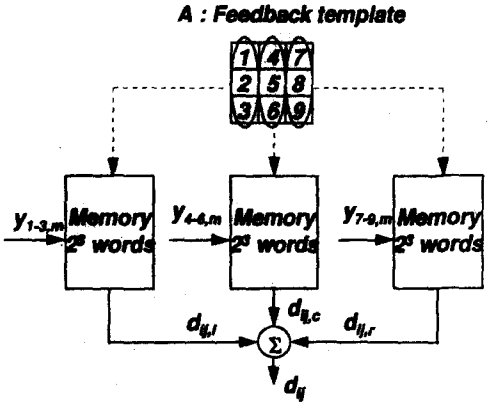

그림 5. 메모리 분할 방법

Fig. 5. Memory partitioning scheme.

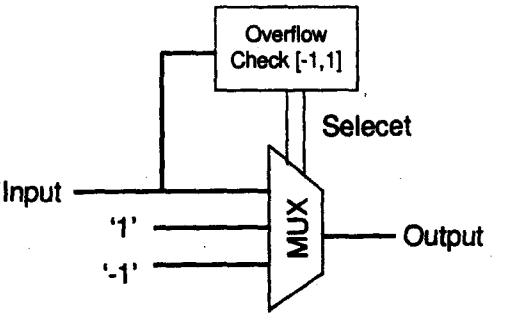

그림 6. Piecewise 선형함수의 구현

Fig. 6. Implementation of the piecewise linear function.

표 1의 halftoning의 예와 같이 대규모 영상을 (512x512 크기, r=1)을 *SUN/SPARC-20* 컴퓨터를 사용하여 소프트웨어적으로 모의실험할 경우 처리 시간이 29.5초가 소요되나, 제작된 FPGA 보드를 사용하면 72.1m초만이 필요하여 매우 큰 모의실험 시간 감소 효과를 얻을 수 있다. 그러나 한 개의 FPGA를 가진 제작된 보드로 큰 영상을 처리하기 위해서는 영상을 분할하여 처리해야 하고, 이에 따른 칩과 호스트 간의 통신에 필요한 작업량이 많아지므로, 제작된 보드로 이를 처리하는 것은 비효율적이다. 그러므로 이와 같은 대규모 network을 구현하기 위해서는 Mars-III

시스템와 같은 상용 FPGA 에뮬레이터를 사용하는 것이 적합하다 [8]. Mars-III 시스템은 많은 수의 FPGA를 어레이 형태로 가지며, 200K 게이트 정도의 복잡도를 가지는 시스템의 구현이 가능하다. 대규모 시스템을 상용 FPGA 에뮬레이터를 사용하여 구현시, 연결의 수가 많은 시스템의 경우는 자원(resource)의 많은 부분이 배선에 소요되어 구현이 효율적이지 못하다. 그러므로 DTCNN의 국소적 연결 특성과 분산연산 구조의 비트별 연산에 따르는 단일 선에 의한 셀간 연결은 전체적으로 연결의 수를 줄이므로 상용 FPGA 에뮬레이터를 사용하여 제안된 구조를 구현하는 데에 매우 유리하게 작용한다.

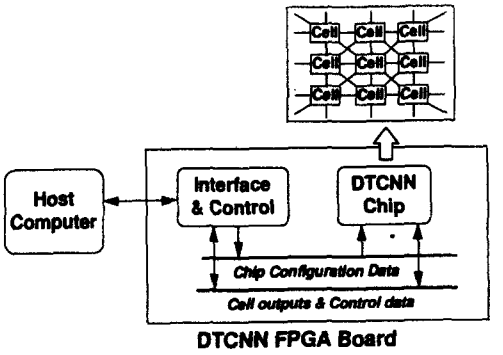

그림 7. DTCNN FPGA 보드의 블록도
Fig. 7. Block diagram of the DTCNN FPGA board.

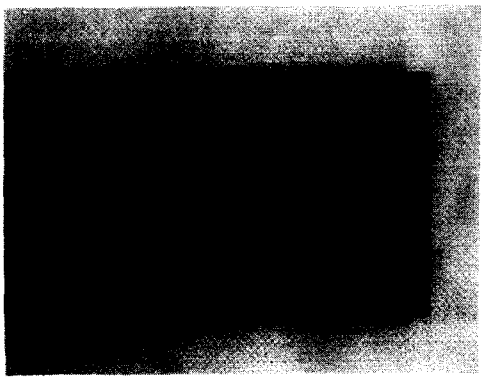

그림 8. DTCNN FPGA 보드
Fig. 8. DTCNN FPGA boark.

V. 결 론

본 논문에서는 DTCNN을 디지탈 방식을 사용하여 효율적으로 구현하는 방법을 제시하였다. 디지탈 방식을 사용시 아날로그 방식에 비해 구현이나 확장의 용이함을 가지고 있으나 많은 양의 하드웨어와 넓은 버스폭이 필요한 것이 단점이다. 이러한 단점들을 해결하기 위해 제안된 구조에서는 분산연산 방식을 DTCNN의 특성들과 결합함으로써 간단한 하드웨어로 DTCNN을 구현하였다.

DTCNN의 국소적 연결 특성은 분산연산 방식에 필요한 메모리 크기를 기타의 일반적인 신경회로망에 비해 크게 줄이는 장점이 있으며, 특히 배선에 제약이 있는 FPGA구현시에 큰 장점으로 작용한다. 템플릿의 공간 불변성은 다중 읽기-포트 메모리를 사용하여 여러 셀과 그 내용을 공유할 수 있게 함으로써 하드웨어의 복잡도를 크게 줄일 수 있는 장점이 있다. 또한 분산연산 방식은 비트별 연산을 수행함으로써 주변 셀과의 연결을 위해 단지 하나의 선만이 요구되어, 디지탈 방식의 단점인 넓은 폭의 버스가 필요치 않으므로 많은 셀을 가지는 큰 시스템으로 쉽게 확장할 수 있게 한다. 제안된 구조는 FPGA의 프로그래밍 특성을 이용하여 내부 셀 구조를 쉽게 바꿀 수 있으므로 이웃거리 r이 가변적인 응용에 적합하다.

각 셀은 Xilinx FPGA에 구현되었으며, 여러 응용에 따른 셀의 하드웨어 복잡도를 필요한 게이트와 CLB 수로 나타냈다. 제안된 구조는 FPGA 보드로 제작되었다.

참 고 문 헌

[1] L. O. Chua and L. Yang, "Cellular neural networks: Theory," *IEEE Trans. Circuits Syst.*, vol. CAS-35, pp. 1257-1272, Oct. 1988.

[2] H. Harrer, J. H. Nossek, and R. Stelzl, "An analog implementation of discrete-time cellular neural networks," *IEEE Trans. Neural Netwroks*, vol. 3, pp. 466-476, Jun. 1992.

[3] A. Rodriguez-Vazquez, S. Espejo, R. Dominguez-Castro, J. L. Huertas, and E. Sanchez-Sinencio, "Current mode techniques for the implementation of continuous- and discrete-time cellular neural networks," *IEEE Trans. Circuits*

Syst., vol. CAS-40, pp. 132-146, Mar. 1993.

[4] A. Peled and B. Liu, "A new hardware realization of digital filter," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-22, pp. 456-462, Dec. 1974.

[5] The Progammable Logic Data Book, Xilinx, 1991.

[6] H. Harrer, "Multiple-layer discrete-time cellular neural networks using time-varaint templates," *IEEE Trans. Circuits Syst.*, vol. CAS-40, pp. 191-199, Mar. 1993.

[7] User guide for Dawn VME DPS-1, Dawn, 1990.

[8] S. Walters, "Computer-aided prototyping for ASIC-based systems," *IEEE Mag. Design & Test of Computers*, pp. 4-10, Jun. 1991.

저 자 소 개

朴 成 晙(正會員) 第 31卷 B編 第 12號 參照
현재 서울대학교 전기공학부 박사과정

蔡 洙 翊(正會員) 第 31卷 B編 第 5號 參照
현재 서울대학교 전기공학부 및 반도체공동연구소 교수

林 俊 皓(正會員) 第 32卷 B編 第 3號 參照
현재 서울대학교 전기공학부 박사과정