碩士學位論文

Hopfield 아날로그 신경회로망의 성능개선과 디지털 신경회로망의 ASIC 설계

濟州大學校 大學院

電 子 工 學 科

高 京 希

1998年 4月

Performance Improvement of Hopfield Analog Neural Network and Digital Neural Network Design Using an ASIC Design Techniques

Kyung-Hee Ko

(Supervised by Professor Min-Je Kang)

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF ENGINEERING

DEPARTMENT OF ELECTRONIC ENGINEERING
GRADUATE SCHOOL
CHEJU NATIONAL UNIVERSITY

1998. 4.

Hopfield 아날로그 신경회로망의 성능개선과 디지털 신경회로망의 ASIC 설계

指導敎授 康 珉 齊

高 京 希

이 論文을 工學 碩士學位 論文으로 提出함

1998年 4月

高京希의 工學 碩士學位 論文을 認准함.

審査委員長 _____印

委 員 _____印

委 員 _____印

濟州大學校 大學院

1998年 4月

목 차

Summary

The method eliminating the third term of computational energy of Hopfield analog neural network has been proposed and the digital neural network is designed by using an ASIC design technology.

The performance of neural network has been improved by eliminating the third term of computational energy. An analytical method is used to improve the performance of the neural network and this method is compared to the previous heuristic methods.

VHDL is used to model the bidirectional associative memory and SYNOPSIS CAD tool is used to synthesize logically. To implement the system by using FPGA, the FLEX8000 library of ALTERA company is used.

Ⅰ. 서 론

컴퓨터의 처리방식을 인간의 특성과 유사하게 만들고자 하는 노력은 오래 전부터 계속되어 왔는데 그 중의 하나가 신경회로망이다. 신경회로망은 1943년 McCulloch와 Pitts의 논문에서 시작이 되어 발전해 오다가 60년대 말 기대했던 퍼셉트론(Perceptron)이 간단한 선형분리 문제도 풀어내지 못함을 알게되면서 침체기를 걷게 되었다. 그러던 것이 80년대가 되면서 Hopfield, Hinton, Grossberg, Rumelhart, Kohonen 등이 신경회로망에 대한 새로운 관심을 불러일으키면서 활발한 연구가 진행되고 있다(Zurada, 1992).

1984년 Hopfield가 궤환성을 갖는 단층신경회로망을 처음 소개한 이래로 그의 공동연구자 Tank과 함께 신경회로망 특성 및 신경회로망을 이용한 응용회로들을 발표하였다. 그후로 지금까지 궤환성을 갖는 신경회로망은 최적화, 패턴인식, 데이터분류화, 통신망에서의 라우팅, 초기동기 등의 응용분야에서 많은 연구가 계속되고 있는데 특히, 연상기억이나 최적화 문제를 병렬적으로 푸는데 많이 사용되고 있다. 궤환성을 갖는 신경회로망은 대부분의 경우 흩어져있는 에너지 함수의 전체최소점으로 수렴하며, 비록 지역최소점으로 수렴하더라도 전체최소점과 매우 근접한 곳으로 수렴한다. 수렴하는 과정은 많은 뉴론들이 병렬로 연결되어서 빠른 속도로 처리하는데, 이런 특성은 컴퓨터 프로세싱 방법보다는 인간들이 프로세싱 하는 방법에 더욱 근접 하다고 하겠다. 그래서, 기존의 알고리즘으로 해결하기 어려운 패턴인식이나 이성적 사고판단을 요하는 문제들에서는 궤환성을 갖는 신경회로망이 기존의 컴퓨터 보다 적합하다. 신경회로망을 응용하기 위해서 현재로는 해결해야 할 많은 문제들이 남아 있으나, 이런 문제들이 해결되면 신경회로망은 현존하는 알고리즘의 문제점들을 보완할 수 있는 차세대의 대안으로 생각되고 있다.

궤환성을 갖는 신경회로망은 신호의 형태에 따라 아날로그 신경회로망과 디지털 신경회로망으로 크게 나눌 수 있다. 아날로그 신경회로망은 주로 최적화 문제

를 푸는데 사용되며 종종 지역최소점으로 수렴해버리는 문제점을 가지고 있다. 디지털 신경회로망은 패턴이식 문제를 다루는데 사용되며 소프트웨어적으로 시스템을 구성하기에는 간단하나, 하드웨어로 구성하려면 많은 고려해야할 사항이 있다.

본 논문에서는 지역최소점으로 수렴하는 문제를 개선하여 아날로그 신경회로망의 성능을 개선시키고, 확장성이 크고 재사용이 편리한 디지털 신경회로망을 블록설계하였다.

아날로그 신경회로망이 지역최소점으로 빠지는 문제를 해결하기 위하여 많은 연구가 되어 왔는데 그 방법들을 살펴보면, 어닐링(annealing)방법(Kirkpatrick et al., 1983; Arts and Van Laarhoven, 1985; Lee and Sheu, 1988), 잡음필터링(noise filtering)방법(Geman, 1984), Boltzman함수 이용방법(Rummelhart et al., 1986), 초기치조건 조정방법(Rutenbar, 1989) 등이다. 이러한 방법들은 문제점을 어떻게 해결하느냐에 따라 다소 차이는 있으나 근간은 휴리스틱(heuristic)한 방법들이다. 휴리스틱한 방법들은 소규모의 신경회로망에 적용하는 데는 문제가 없으나, 뉴론의 수가 증가함에 따라 계산량이 방대해져서 대 규모의 신경회로망에는 적용이 어려워진다.

본 논문의 Ⅱ장에서는 Hopfield의 에너지 이론을 살펴보고, 그 에너지 이론에 따른 계산에너지함수가 시간에 따라 에너지가 감소하는 방향으로 수렴해가는지를 증명한다. Ⅲ장에서는 지역최소점으로 빠지는 문제를 해결하기 위하여 계산에너지함수를 분석하였다. 이 방법은 해석적인 방법으로서, 뉴론의 증가와 관계없이 계산량이 일정함으로 대규모의 신경회로망에도 적용 가능한 방법을 제시하였다.

디지털 신경회로망을 하드웨어로 구성하려면 먼저 사용할 설계방법, 설계툴과 소자의 크기에 따라 시스템의 구조를 결정하여야 한다. 확장성이 크고, 재사용이 편리한 하드웨어 구현언어(VHDL: Very high speed integrated circuit Hardware Description Language)를 사용하여 블록설계를 하였는데 시스템의 동작을 위해서 입력단, 출력단, 연산장치, 기억장치가 있어야 하며, 이러한 블록들을 제어하기 위하여 상태발생기 블록을 설계하였다.

Ⅳ장에서는 디지털 신경회로망인 양방향 연상 기억장치의 ASIC 설계를 위하여 VHDL을 이용하여 다섯 개의 블록으로 설계한 구조를 살펴보고, 각 블록마다 SYNOPSYS라는 CAD tool을 사용하여 시뮬레이션하고 합성한 회로도를 제시하였고, 전체 시스템의 시뮬레이션 결과를 보였다. Ⅴ장에서 결론을 맺고, 부록에서는 이해를 돕기 위해 주문형 반도체(ASIC: Application Specific Integrated Circuit)에 대한 일반적인 내용과 VHDL로 설계한 파일을 첨가하였다.

Ⅱ. Hopfield의 에너지 이론

인간의 뇌는 약 천억 개의 뉴론(neuron)으로 구성되어 있고, 각 뉴론들은 만 개 정도의 다른 뉴론들과 연결되어 있다. 이러한 뉴론과 뉴론은 시냅스(synapse)라는 연결강도로 연결되어 있으며, 연결강도들은 뇌의 사용정도에 따라 학습되어 간다. 즉, 인간이 교육해 나가는 과정은 뉴론들을 연결하는 연결강도들의 세기를 조정해 가는 과정이라 할 수 있다. 외부의 감각기관을 통해 신호가 입력되면 뉴론과 연결강도들을 통해서 다른 뉴론의 입력에 신호량이 전달되며, 뉴론들은 그 상황에 맞는 판단을 하게 된다. 판단이 결정되면 더 이상 뉴론들의 출력변화는 없게 되어 안정된 상태가 유지된다. 즉, 판단이 결정되기까지는 에너지가 불안정한 상태라 할 수 있고, 판단을 하는 과정은 에너지상태가 낮은 곳으로 수렴하는 과정이라 생각할 수 있다. 그림 1은 뉴론과 연결강도의 구성을 보여준다.

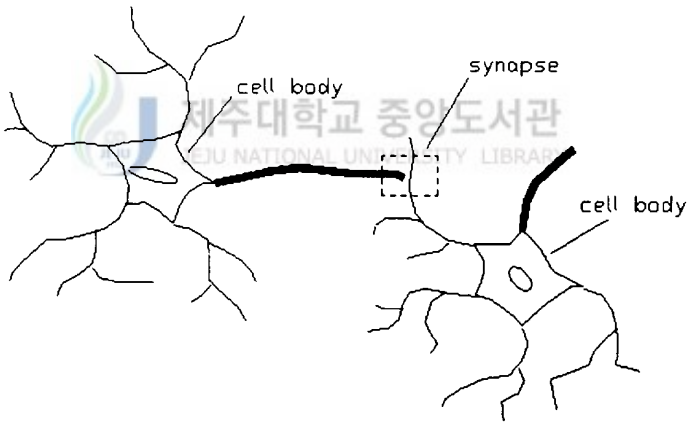


Fig. 1. Biological neuron

1984년에 Hopfield는 이러한 현상에 착안하여 뉴론, 연결강도 및 외부입력으로 구성된 궤환성을 갖는 신경회로망을 발표하였다. Hopfield 신경회로망은 계산에너

지를 가지고 있으며, 계산에너지함수가 감소하는 방향으로 수렴하여 안정하게 된다. Hopfield 신경회로망의 계산에너지함수의 굴곡은 그림 2와 같이 산과 계곡 등으로 이루어진 지형으로 간주할 수 있다. 신경과 신경을 잇는 연결강도 및 외부 입력신호에 의해 함수의 굴곡의 정도는 결정되는데, Hopfield 신경회로망의 연결강도 및 외부 입력신호는 최적화하려는 비용함수 및 저장된 입력 패턴에 의해 결정된다. 신경회로망은 계산에너지함수를 감소시키는 방향으로 변해 가는데, 이러한 이치는 빗물이 위치에너지가 낮은 바다로 흘러가는 것과 같고, 전체최소점으로 수렴하지 못하고 지역최소점으로 수렴해버리는 경우는 빗물이 위치에너지가 낮은 곳으로 흘러가다가 작은 웅덩이 또는 계곡에 빠져버리는 경우와 같은 것이다.

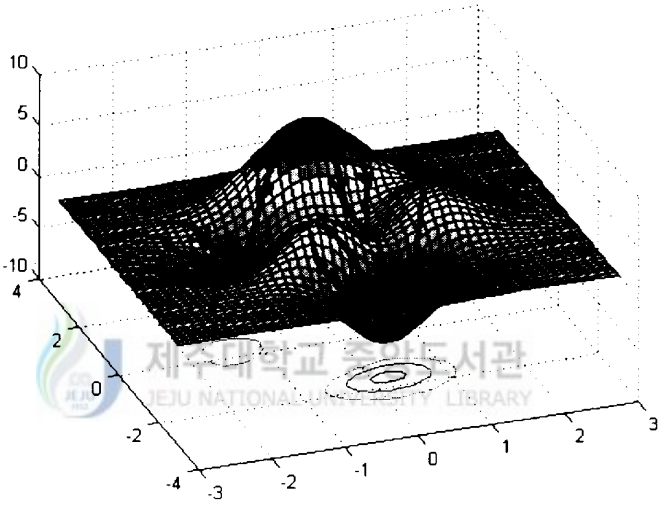


Fig. 2. Computational energy of Hopfield Neural Networks

그림 3은 궤환성을 갖는 아날로그 신경회로망의 전기적 모델을 보여준다. 이런 형태의 신경회로망은 궤환성을 가지며 뉴론의 출력들이 다시 다른 뉴론의 입력으로 연결강도(w_{ij})들을 통하여 연결된다. j번째 뉴론의 출력(v_j)과 i번째 뉴론의 입력(u_i)을 연결하는 연결강도(w_{ij})들은 전기적 모델에서는 컨덕턴스를 사용하

며, 뉴론의 출력값은 입력단의 값에 매핑 되는 함수로써 아날로그인 경우 시그모이드(sigmoid) 함수가 쓰인다.

아날로그 신경회로망의 전기적 모델에서 뉴론의 입력단에서 세운 KCL (Kirchhoff's Current Law)을 이용하여 다음의 공식이 유도된다.

$$C_i\left(\frac{du_i}{dt}\right) = I_i + \sum_{j=1, j\neq i}^{n} w_{ij}v_j - u_i\left(\sum_{j=1, j\neq i}^{n} w_{ij} + g_i\right) \quad (1)$$

여기서, g_i와 C_i는 입력단을 안정하게 하기 위하여 연결되는 입력컨덕턴스와 캐패시터이다.

식 (1)의 우변은 캐패시터 C_i로 유입되는 전류의 총량을 나타내며, i번째 뉴론의 입력에 연결된 전체 컨덕턴스를 G_i로 다음과 같이 나타내면

$$G_i \equiv \sum_{j=1, j\neq i}^{n} w_{ij} + g_i \quad (2)$$

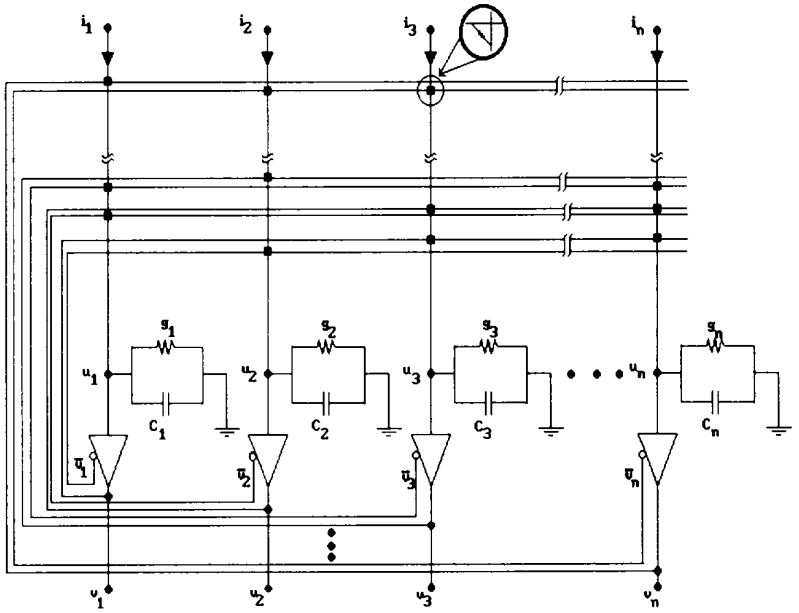
식 (1)은 다음과 같이 간략하게 나타내어진다.

$$C_i\left(\frac{du_i}{dt}\right) = I_i + \sum_{j=1, j\neq i}^{n} w_{ij}v_j - u_iG_i \quad (3)$$

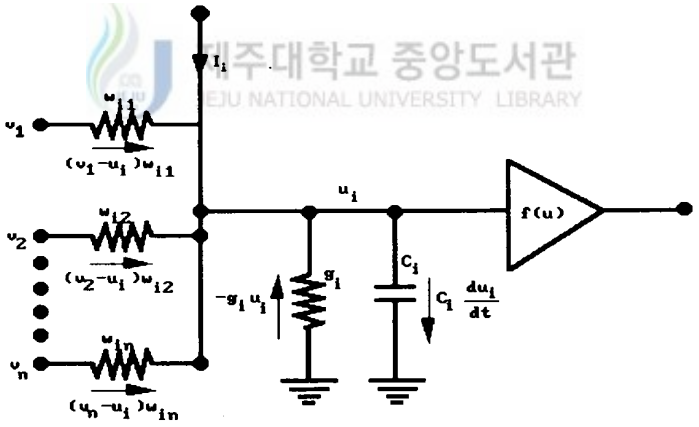
벡터형식으로 상태방정식과 출력방정식을 다음과 같이 표현 가능하다.

$$C\left(\frac{du(t)}{dt}\right) = Wv(t) - Gu(t) + I(t) \quad (4)$$

$$v(t) = f(u(t)) \quad (5)$$



(a)



(b)

Fig. 3. (a) Electrical model of analog neural network with feedback. (b) Input node of neuron I.

n차원의 상태방정식으로 표현되는 시스템이 어떤 계산에너지함수의 값을 따라 점근적으로 안정되게 수렴하여 간다면 그 계산에너지함수를 그 시스템의 Lyapunov함수라 한다. 그림 3에서 신경회로망의 Lyapunov함수는 다음과 같이 알려져 있다.

$$E(v) = -\frac{1}{2} v^T W v - I^T v + \sum_{i=1}^{n} G_i \int_{\frac{1}{2}}^{v_i} f_i^{-1}(z)\, dz \tag{6}$$

그림 3의 시스템에서 뉴론의 출력 v는 이 계산에너지함수가 감소하는 방향으로 변하면서 안정되게 수렴해 간다.

식 (6)의 제 3항은 뉴론 매핑함수의 역함수를 적분한 것으로 이것을 미분하면 다음의 관계가 성립한다.

$$\frac{d}{dv_i}\left(G_i \int_{\frac{1}{2}}^{v_i} f_i^{-1}(z)\, dz\right) = G_i u_i \tag{7}$$

연결강도 행렬이 대칭인 점을 고려하여 식 (6)을 체인규칙(chain rule)을 이용하여 시간에 관해 미분해보면 다음과 같다.

$$\begin{aligned} \frac{dE}{dt} &= \frac{dE}{dv}\frac{dv}{dt} \\ &= (-Wv - I + Gu)^T \frac{dv}{dt} \end{aligned} \tag{8}$$

식 (4)를 이용하여 식 (8)을 재정리하면 다음과 같다.

$$\frac{dE}{dt} = -\left(C\frac{du}{dt}\right)\frac{dv}{dt} \tag{9}$$

그런데 뉴론의 출력 v는 입력 u의 함수이므로 체인규칙을 사용하면 식 (9)을 다음과 같이 나타낼 수 있다.

$$\begin{aligned} \frac{dE}{dt} &= -C\frac{du}{dt}\frac{du}{dt}f'(u) \\ &= -C\left(\frac{du}{dt}\right)^2 f'(u) \end{aligned} \tag{10}$$

$f(u)$는 시그모이드 함수이므로 항상 증가함수이고 미분값은 양이다. 따라서, 식 (10)의 우변은 항상 음이므로 이 시스템은 시간의 흐름에 따라 에너지가 감소하는 방향으로 수렴하는 시스템임을 알 수 있다.

Hopfield 디지털 신경회로망은 패턴인식 등에 사용되는데 아날로그 신경회로망과 비교해보면, 아날로그 신경회로망은 뉴론의 출력이 시그모이드 함수이고 외부에서의 입력을 외부입력(I_i)에서 인가하는데 반하여, 디지털 신경회로망은 뉴론의 출력이 계단함수이고 외부입력이 뉴론의 출력에 직접 인가된다는 것만 다르다. 즉, 디지털 신경회로망은 외부입력이 없고, 연결강도들은 저장된 패턴의 상황에 따라 결정된다. 출력에 인가된 왜곡된 패턴의 신호는 연결강도를 통해 뉴론의 입력으로 전달되며, 신경회로망은 계산에너지함수가 감소하는 방향으로 수렴하여 원하는 패턴을 찾게되며 안정화된다.

Ⅲ. 에너지함수에 근거한 Hopfield 아날로그 신경회로망의 성능개선

1. Hopfield 아날로그 신경회로망의 성능개선

Hopfield 신경회로망은 종종 지역최소점으로 수렴하여 원하지 않는 결과를 발생하는 문제가 있다. 이러한 현상은 디지털 출력을 위하여 계산에너지함수에 부수적으로 첨가된 항과 식 (6)의 계산에너지함수에 존재하는 제 3항 때문이다. 제 3항 에너지를 최소화하기 위하여 다음의 뉴론 입출력 함수 $f(u)=1/(1+e^{-\lambda u})$에서 시그모이드 이득률 λ를 크게 하는 방법을 사용하고 있으나 λ를 너무 크게 하면 함수 $f(u)$는 $u=0$ 근처에서 급히 상승하게 되어 단위함수에 가까워지고 시스템이 안정되게 수렴하는데 문제가 된다.

이 장에서는 임의의 λ값을 사용해도 시스템이 안정되게 수렴하는데 문제가 없으면서, 제 3항 에너지를 제거하는 새로운 방법을 제안한다.

제 3항 에너지는 각 뉴론에서 입력 u_i의 초기치들이 0이므로 출력 $f(u)$의 초기치들은 0.5이며, 다음과 같이 표현된다.

$$E_3 \equiv \sum_{i=1}^{n} G_i \int_{\frac{1}{2}}^{v_i} f(\lambda, z)^{-1} dz \tag{11}$$

여기서,

$$f(\lambda, z)^{-1} = \frac{1}{\lambda} \ln\left(\frac{z}{1-z}\right) \tag{12}$$

제 3항 에너지를 자세히 표현하면 식 (13a)와 같다.

$$E_3 = \sum_{i=1}^{n} \frac{G_i}{\lambda_i} \int_{\frac{1}{2}}^{v_i} \ln\left(\frac{z}{1-z}\right) dz \tag{13a}$$

식 (13a)에서 적분부분을 계산하면 결과는 다음 식과 같다.

$$E_3 = \sum_{i=1}^{n} \frac{G_i}{\lambda_i} (v_i \ln v_i + (1-v_i)\ln(1-v_i) + \ln 2) \tag{13b}$$

식 (13b)에서 제 3항의 에너지는 $v=1/2$에서 0이며, v가 0 또는 1로 변화함에 따라 제 3항 에너지는 식 (14)와 같이 수렴함을 알 수 있다.

$$\lim_{v_i \to 0,1} E_3 = \sum_{i=1}^{n} \frac{G_i}{\lambda_i} \ln 2 \tag{14}$$

그림 4는 2비트 A/D 변환기의 제 3항의 에너지 분포도를 나타낸다. 시스템에서 뉴론의 수 n이 증가할수록 연결강도의 수가 증가하게 되며, 따라서 G_i도 증가하여 제 3항의 에너지는 더욱더 커지게 된다.

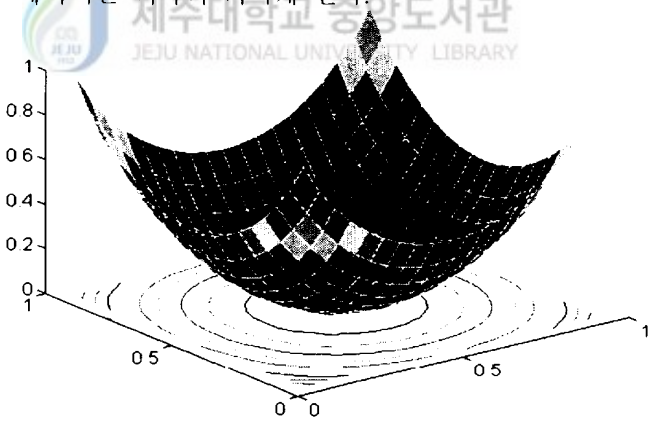


Fig. 4. Energy distribution chart of third term for two-bit A/D converter, λ=2, G_i=2 ℧

에너지 함수를 자세히 고찰하기 위하여 식 (5)를 출력전압 v에 관해서 미분하면 다음 식을 얻는다.

$$\frac{dE}{dv} = (-Wv - I + Gu) \tag{15a}$$

식 (6)과 식 (15a)를 비교해보면, 식 (15a)의 우변에서 제 3항이 제거되면 식 (6)의 방정식에서도 제 3항이 제거됨을 알게 된다.
식 (15a)를 그림 3b와 비교하기 위하여 재정리하면

$$\frac{dE}{dv} = (-W(v-u) - I + gu) \tag{15b}$$

식 (15b)와 그림 3b를 비교해보면 I_i는 노드 i로 입력되는 바이어스 전류를 표현하고, $W(v-u)$는 각 뉴론의 출력들에서 연결강도 W들을 통해서 노드 i로 입력되는 전류를 나타내고, Gu는 노드 i에 연결된 g를 통해서 접지로 흘러나가는 전류임을 알 수 있다. 따라서, 식 (15a)에서 제 3항을 제거하려면 G 즉, $g+W$값이 0이 되어야하므로 $-W$값에 해당하는 g를 사용하여 제 3항을 제거하면 된다. 하지만 현실적으로 음의 값을 갖는 컨덕턴스를 제조하기란 쉽지 않으므로 컨덕턴스 g 대신에 노드 i의 전압 u_i에 의해 조정되는 종속전류원 $K_i u_i$를 연결하고, 노드 i로 전류를 유입시키면 제 3항의 에너지를 완전히 제거할 수 있다. K_i값은 그 뉴런의 해당 연결강도의 합과 같으며 다음 식으로 표현된다.

$$K_i = \sum_{j=1, j\neq i}^{n} W_{ij} \qquad i = 1, 2, \cdots n \tag{16}$$

여기서, W_{ij}는 i번째 뉴론의 입력과 j번째 뉴론의 출력을 연결하는 연결강도이다.

2. 시뮬레이션 및 결과

시뮬레이션을 위해서 사용된 A/D 컨버터는 최적화 문제의 특별한 예로써 Hopfield가 그의 논문에서 사용한 이후로, Hopfield 신경회로망의 성능분석을 위하여 많은 논문에서 사용되고 있다. 신경회로망의 입력은 아날로그 값 x이며, 출력은 디지털 값이다. 즉, 아날로그 값 x를 가장 근접한 디지털 출력값으로 최적화 시키기 위하여 LMS(Least Mean Squared) 형식으로 표현된 비용함수와 Hopfield 신경회로망의 계산에너지함수를 일치시켜서 시간이 경과함에 따라 계산에너지함수를 최소화하려는 Hopfield 신경회로망의 성질을 이용하여 입력값 x에 가장 근접한 디지털 출력을 얻는 시스템이다. 그림 5는 제안된 방법으로 에너지함수의 제 3항을 제거한 궤환성을 갖는 신경회로망의 4비트 A/D 컨버터의 전기적 모델이다.

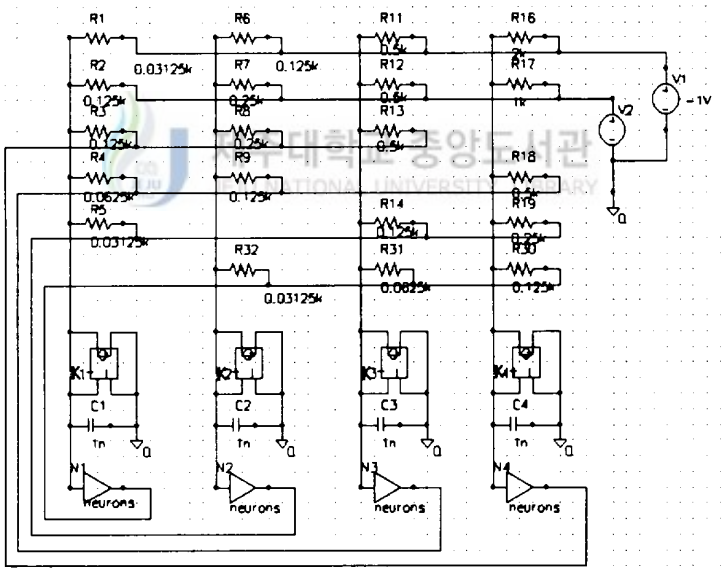


Fig. 5. Electrical model of the eliminated third term for Hopfield Analog Neural Network with feedback

4비트 A/D 컨버터의 비용함수를 LMS 형식으로 표현하면 다음과 같이 된다.

$$E_c = \frac{1}{2}(x - \sum_{i=0}^{3} v_i 2^i)^2 \tag{17}$$

식 (17)을 최소화하면 아날로그 입력 x와 $\sum_{i=0}^{3} v_i 2^i$ 의 값이 같아지지만, v_i의 값이 이진값을 보장하지는 않는다. 그래서, 뉴론 출력 v_i의 값이 이진수가 되도록 도와주는 항이 부수적으로 첨가되어야 하는 데 다음과 같다.

$$E_a = -\frac{1}{2}\sum_{i=0}^{3} 2^{2i} v_i(v_i - 1) \tag{18}$$

v_i가 0이나 1이면, E_a의 값은 0이고, v_i가 이진수 이외의 값이면 E_a는 양의 값을 지니게 되므로 v_i가 이진값 즉, 0 또는 1일 때 가장 낮은 에너지 값을 갖도록 함으로써, 식 (18)은 v_i가 이진값을 갖도록 도와준다.

시스템의 연결강도 u와 외부입력 I를 구하기 위하여, 식 (17)과 식 (18)을 합한 후, 식 (6)의 신경회로망 계산에너지함수에서 제 3항이 제거된 식과 비교하면 다음과 같다.

$$\frac{1}{2}(x - \sum_{i=0}^{3} v_i 2^i)^2 - \frac{1}{2}\sum_{i=0}^{3} 2^{2i} v_i(v_i - 1) = -\frac{1}{2} v^T W v - I^T v \tag{19}$$

위 식의 좌변을 확장하여 우변과 비교하면 연결강도와 외부입력은 다음과 같이 구할 수 있다.

$$w_{ij} = -2^{i+j} \tag{20a}$$

$$I_i = -2^{2i-1} + 2^i x \tag{20b}$$

4비트 A/D 컨버터인 경우에 매트릭스 형태로 다시 표현하면 다음과 같다.

$$W = -\begin{bmatrix} 0 & 2 & 4 & 8 \\ 2 & 0 & 8 & 16 \\ 4 & 8 & 0 & 32 \\ 8 & 16 & 32 & 0 \end{bmatrix} \qquad I = -\begin{bmatrix} \frac{1}{2} - x \\ 2 - 2x \\ 8 - 4x \\ 32 - 8x \end{bmatrix} \tag{20c}$$

식 (20)에서 연결강도들이 음의 값을 가짐을 알 수 있다. 그러나 음의 연결강도를 표현하기 위해 전기적 모델에서 음의 값을 갖는 컨덕턴스를 만들기가 용이하지 않으므로 음의 값을 갖는 뉴론과 양의 컨덕턴스를 사용함으로써 음의 연결강도와 같은 효과를 가진 전기적 모델을 구성하였다. 그림 6은 음의 값을 갖는 뉴론의 입출력 함수들을 나타내는데, 출력은 시그모이드 함수이고, 이득률 λ가 큰 값일수록 단위함수와 유사해짐을 알 수 있다.

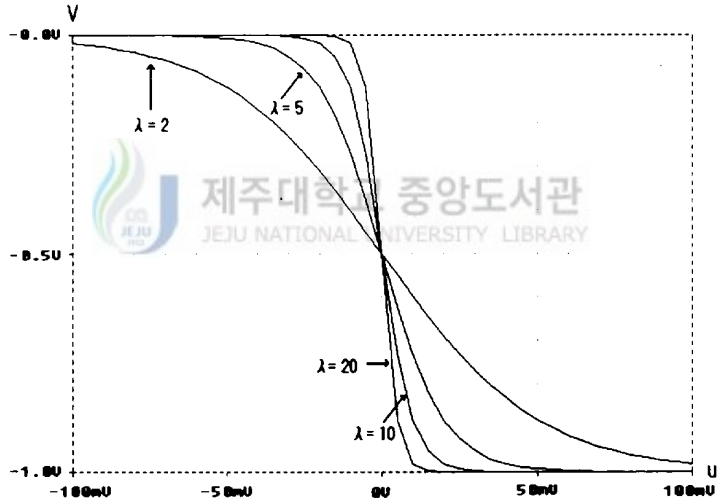


Fig. 6. Input and output functions of neuron

시스템의 전력소모를 감소시키기 위하여 계산에너지함수에 전체적으로 10^{-3}을 곱하여 연결강도를 축소시켰고, 연결강도(W)의 값인 컨덕턴스를 표현하기 위해서

전기적 모델에서는 저항을 사용하였다. 그래서, 종속전류원들의 이득률 K의 값은 식 (16)에 의해서 $K_1 = 5.6\times10^{-2}$, $K_2 = 4.4\times10^{-2}$, $K_3 = 2.6\times10^{-2}$, $K_4 = 1.4\times10^{-2}$ 이다. 즉, 저항 값은 컨덕턴스의 역수를 취해서 구했기 때문에 Hopfield의 연구에서 나타난 연결강도 값에서 역수를 취하고 10^3을 곱한 값이다. 시뮬레이션은 PSpice를 사용하였고, 그림 7a는 제 3항이 있는 시스템에서 입력전압이 2.3V, λ = 2, 그리고 g_i = 10 ℧일 때 과도응답 시뮬레이션 결과이고, 그림 7b는 새로 제안된 방법에서 임의의 입력전압이 2.3V이고 λ = 2일 때 과도응답 시뮬레이션 결과를 보여준다. 양의 연결강도를 사용하기 위하여 음수값을 갖는 뉴론을 사용했고, 뉴론들에서 입력 u의 초기값들이 0이므로 출력 v의 초기값들은 -1/2에서 시작하며 시간이 지나면서 일정한 값으로 수렴해감을 보여준다. 제 3항이 있는 경우(그림 7a)에서는 V_0(LSB)가 이진값으로 수렴하지 못하고, 제안된 방법(그림 7b)에서는 이진값으로 수렴해 감을 알 수 있다.

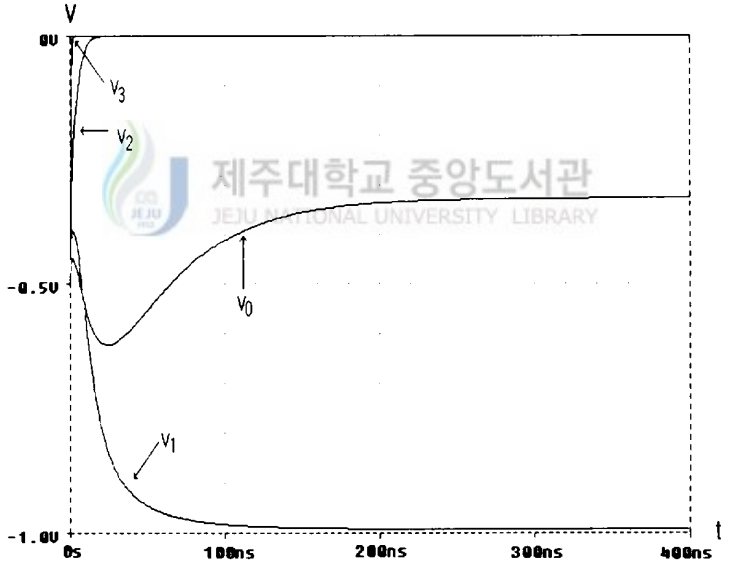


Fig. 7a. Electrical model of Hopfield Analog Neural Network with feedback where the third term exists. input voltage=2.3V, λ =2, g_i=10 ℧

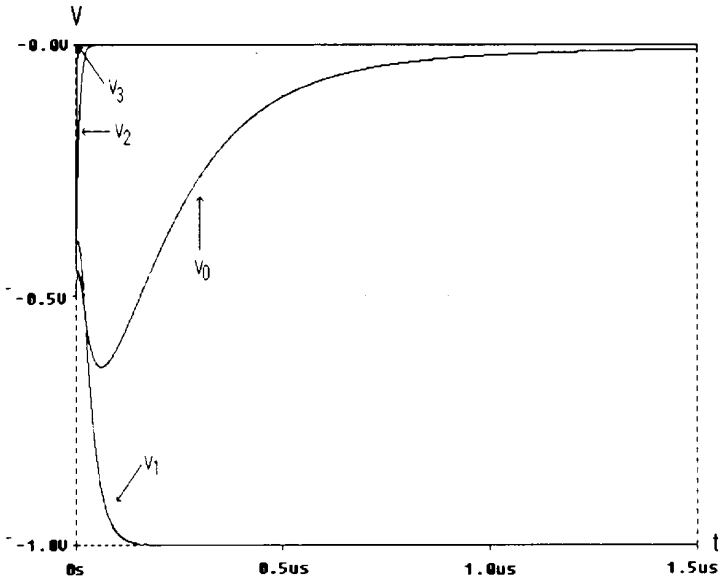


Fig. 7b. Electrical model of Hopfield Analog Neural Network with feedback where the third term is eliminated. input voltage = 2.3V, λ =2

제안된 방법과 제 3의 에너지 항이 존재하는 방법과의 결과를 비교해보기 위해 아날로그 입력 x는 [0, 15.4]의 영역을 0.1씩 증가하면서 4bit A/D 변환기에 적용하여 155가지의 경우에 대해 과도응답 시뮬레이션을 하였다. 뉴론의 입력단에 연결된 g_i의 값에 따른 영향을 보기 위하여 뉴론의 이득률 λ를 10으로 고정시키고, g_i의 값을 100 ℧, 10 ℧, 1 ℧, 0.01 ℧로 변화를 주어 시뮬레이션 한 결과 155가지의 경우의 수에서 각각 40, 34, 20, 20개의 오류가 발생하였다. 결과에서 알 수 있듯이 g_i값이 클수록 오류가 발생하는 수가 증가하였으며, 이것은 g_i의 값이 크면 제 3항 에너지가 증가하므로 예상된 결과이다. 그러나 제 3항 에너지가 존재하는 모델에서 g_i의 값이 1 ℧이하로 감소시켜도 더 이상의 에러를 줄이지 못하고 20개의 에러를 가진 135개의 경우들이 정확한 값으로 수렴함을 나타내었다.

그리고 제 3항이 존재하는 경우 λ의 값이 감소함에 따라 뉴론의 출력 V_i들이 0 또는 1의 이진값에 수렴하지 못하는 경우가 발생하는 반면, 제안된 모델에서는 λ값의 변화(2, 5, 10)에도 항상 이진값으로 수렴하였고 155개의 경우 중 145개가 정확한 값으로 수렴함을 보였다.

표 1, 2에서는 λ=10일 때 시뮬레이션 결과를 보여주며 표에서 오류가 발생한 부분은 음영으로 표시하였다. 오류가 발생한 부분을 관찰해 보면 표 1, 2에서 모두 점대칭적으로 발생하는 것을 알 수 있었다. 즉, 표 1에서 6.2, 6.3, 6.4에서 오류가 생겼다면 7.5를 중심으로 4비트 시스템의 역학적으로 대칭인 8.8, 8.7, 8.6에서 오류가 역시 생겼음을 발견할 수 있었다. 표 2에서도 역시 7.5를 중심으로 하여 대칭적으로 성능이 향상되었고, 그래서 잔류된 오류들도 대칭적으로 발견되었다.

Table 1. Electrical model of Hopfield Analog Neural Network with the third term, g_i = 0.1 σ λ=10

입력 (정수부)	소 수 부									
	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
0	0	0	0	0	0	0				1
1	1	1	1	1	1	1	2	2	2	2
2	2	2	2			2.5	3	3	3	3
3	3	3	3	3	3	3		4	4	4
4	4	4	4	4	4	4.4			5	5
5	5	5	5	5	5	6	6	6	6	6
6	6	6				7	7	7	7	7
7	7	7	7	7	7	7	8	8	8	8
8	8	8	8	8	8	8				9
9	9	9	9	9	9	9	10	10	10	10
10	10	10	10			10.5	11	11	11	11
11	11	11	11	11		12	12	12	12	12
12	12	12	12	12	12	12.4			13	13
13	13	13	13	13	13	14	14	14	14	14
14	14				15	15	15	15	15	15
15	15	15	15	15	15					

Table 2. Electrical model of Hopfield neural network where the third term is eliminated, λ =10

입력 (정수부)	소 수 부									
	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
0	0	0	0	0	0	0.5	1	1	1	1
1	1	1	1	1	1	1			2	2
2	2	2	2	2	2	2.5	3	3	3	3
3	3	3	3	3	3	3			4	4
4	4	4	4	4	4	4.5	5	5	5	5
5	5	5	5	5		6	6	6	6	6
6	6	6	6	6	6	6.5	7	7	7	7
7	7	7	7	7	7	7	8	8	8	8
8	8	8	8	8	8	8.5	9	9	9	9
9	9	9	9	9	9	9		10	10	10
10	10	10	10	10	10	10.5	11	11	11	11
11	11	11	11			12	12	12	12	12
12	12	12	12	12	12	12.5	13	13	13	13
13	13	13	13			14	14	14	14	14
14	14	14	14	14	14	14.5	15	15	15	15
15	15	15	15	15	15					

Ⅳ. ASIC 설계기술을 이용한 디지털 신경회로망의 구현

1. 디지털 신경회로망인 연상 기억장치(Associative Memory)의 기본적인 개념

일반적으로 '연상하다'라는 말은 현재의 사물이나 사건을 통해 과거의 어떤 기억들을 떠올리거나 유사한 새로운 개념을 만들어내는 것을 의미하는데 연상 기억장치는 이러한 인간의 연상기능을 모방한 것이다. 어떤 패턴을 입력했을 때 저장된 패턴들 중에서 입력된 패턴과 가장 유사하게 닮거나 관계 있는 패턴을 찾아내는 것으로 응답한다. 그러한 기억장치들은 바이트로 저장된 패턴들을 그것의 주소를 사용하여 부르는 디지털 컴퓨터에서의 전통적인 주소-주소 기억장치(address-addressable memory)와는 대조되는 내용-주소 기억장치(content-addressable memory)로 기억정보의 일부분을 이용하여 원하는 정보가 기억된 정보에 접근하는 시스템이다.

연상 기억장치에는 자동연상 기억장치(autoassociative memory)와 이질연상 기억장치(heteroassociative memory)가 있다.

$$X^{(s)} \rightarrow Y^{(s)} \qquad s = 1, 2, \cdots, t \tag{21}$$

여기서 s는 기억장치에 저장된 대표패턴의 수를 나타낸다.

이질연상 기억장치는 식 (21)에서 $Y^{(s)} \neq X^{(s)}$인 경우를 나타내는 것으로 왜곡되거나 방향이 바뀐 X패턴이 입력되었을 때 그와 연관된 Y패턴까지 동시에 찾아내어 가장 유사한 패턴 쌍($X^{(s)}$, $Y^{(s)}$)을 연상해 내는 것이다.

자동연상 기억장치는 $Y^{(s)} = X^{(s)}$인 경우로 물체와 개념의 일부분을 보여주었을 때 시스템 전체를 연상하는 장치로 X패턴이 입력되었을 때 저장된 패턴 $\{X^{(1)}, X^{(2)}, \cdots X^{(p)}\}$들로부터 가장 유사한 $X^{(s)}$를 연상하는 기억장치다.

2. 양방향 연상 기억장치(BAM: Bidirectional Associative Memory)

양방향 연상 기억장치는 두개의 층으로 구성된 내용-주소 기억장치이면서 이질 연상 기억장치이다. 그림 8은 디지털 양방향 연상 기억장치의 구조를 나타내는데 화살표는 정보의 흐름을 표시한 것이다. 두 개의 층 X와 Y는 뉴론으로 구성된 $X=(x_1, x_2, \cdots, x_n)$와 $Y=(y_1, y_2, \cdots, y_m)$의 행벡터이고, 연결강도 W로 상호연결 되어 있다. 입력패턴을 이용하여 저장된 패턴을 연상하기 위해 순방향과 역방향의 정보흐름을 사용한다.

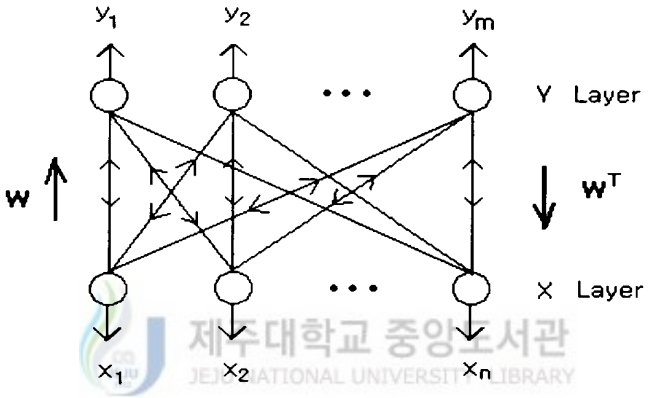


Fig. 8. Topology of BAM

초기행렬 X가 연결강도 W를 통해서 뉴론 층 Y의 입력으로 들어간다면 Y'는 다음과 같이 구할 수 있다.

$$Y' = a(WX) \quad \text{또는,} \quad y_j' = a\left(\sum_{i=1}^{n} w_{ij}x_i\right) \qquad j=1,2, \ldots, m \qquad (22)$$

식 (22)로 입력패턴 X는 Y층의 출력패턴으로 처리되고 변환된다. 여기서, a(·)는 문턱함수이다. 지금 벡터 Y'는 연결강도 W^T를 통해서 역방향인 X층으로 보내

지고 다음의 출력을 만든다.

$$X' = a(W^T Y') \quad \text{또는,} \quad x_i' = a\left(\sum_{j=1}^{m} w_{ij}^T y_j\right), \qquad i=1,2, \ldots, n \qquad (23)$$

연상된 X'가 입력 X와 다르면 X'는 다시 Y층의 입력으로 들어가고 식 (22)에 따라 Y''를 만든다. 이 처리과정은 X와 Y의 변화가 없을 때까지 계속된다.

$$\begin{aligned} &Y' = a(WX) \text{ 첫 번째 순방향과정} \\ &X' = a(W^T Y') \text{ 첫 번째 역방향과정} \\ &Y'' = a(WX') \text{ 두 번째 순방향과정} \qquad (24) \\ &\quad \vdots \\ &X'^k = a(W^T Y'^k) \text{ k 번째 역방향과정} \end{aligned}$$

적당한 메모리 동작을 위해서 각층의 변화는 같은 시간에 발생하지 않는다고 가정한다.

1) 양방향 신경회로망의 부호화

양방향 연상 기억장치는 뉴론들 사이의 연결강도들을 수정함에 의해서 학습한다. X층의 뉴론의 수가 n이고, Y층의 뉴론의 수가 m일 때 연결강도는 n열 m행의 행렬이고, X와 Y층 사이의 연결강도는 모든 입력들이 에너지가 감소하는 안정한 방향으로 빠르게 수렴해 간다.

저장될 벡터쌍 $\{(X^{(1)}, Y^{(1)}), (X^{(2)}, Y^{(2)}) \cdots (X^{(p)}, Y^{(p)})\}$들이 p개 있다면 저장가능한 벡터의 용량 p는 X층에 n개의 뉴론이 존재하고 Y층에 m개의 뉴론이 존

재한다면 저장될 패턴의 수는 각 층에 존재하는 뉴론 수 보다 작다. 따라서, $p < \min(n, m)$이고, 좀더 안정적인 패턴인식을 위한 저장용량은 패턴의 수를 $p < \sqrt{\min(n, m)}$로 정해주는 것이 좋다고 알려져 있다. (Kosko, 1988)

Hebb의 가설에서 연결강도의 값이 입력패턴과 출력패턴 간의 상관(correlation) 값에 따라 변한다고 가정하였는데, 입력이 이진형태 일 때 쌍극형태로 만들어주어 연결강도를 구하는 것이 상관부호화가 개선된다.

$$W = \sum_{s=1}^{p} X^{(s)T} Y^{s} = X^{(1)T} Y^{(1)} + X^{(2)T} Y^{(2)} + \cdots + X^{(p)T} Y^{(p)} \tag{25}$$

여기서, 열벡터 $X^{(i)T}$는 행벡터 $X^{(i)}$의 벡터변환이다.

$$W = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1m} \\ w_{21} & w_{22} & \cdots & w_{2m} \\ & \vdots & & \\ w_{n1} & w_{n2} & \cdots & w_{nm} \end{pmatrix} \tag{26}$$

행렬성분 w_{ij}는 뉴런 x_i와 뉴런 y_j사이에서 대칭적 연결강도를 나타낸다.

양방향 연상 기억장치의 특별한 경우는 X층과 Y층이 같고 모든 $X^{(i)} = Y^{(i)}$일 때 발생한다. 그러면 $W = W^{T}$이고, 에너지가 최소일 때 단일 패턴 X^{i}에 저장하는 대칭적인 자동연상 기억장치가 된다.

2) 양방향 신경회로망의 복호화

복호화는 오류가 있는 패턴이 입력되었을 때 연결강도를 통하여 저장된 패턴을 연상해내는 과정이다. 입력패턴 X가 X층에 입력되었다면 n개의 뉴론들은 연결강도를 가로질러 Y층의 m개의 뉴론의 값을 만들어내는데 복호화를 위한 입력패턴은 이진형태를 가지며 뉴론 값이 0이면 동작하고, 1이면 동작하지 않는다.

X층에서 입력패턴 X가 $X^{(i)}$와 가장 유사한 입력이라면 Y'는 다음 식과 같다.

$$
\begin{aligned}
Y'^{(j)} &= X^{(j)}W = X^{(j)}\sum_{s=1}^{p}X^{(s)T}Y^{(s)} \\
&= X^{(j)}X^{(j)T}Y^{(j)} + X^{j}\sum_{s\neq j}^{p}X^{(s)T}Y^{(s)}
\end{aligned}
\tag{27a}
$$

식 (27a)에서 입력된 X가 $X^{(j)}$와 가장 유사한 입력이므로 $X^{(j)}X^{(j)T}Y^{(j)}$가 가장 큰 값을 가지면서 ($X^{(j)}$, $Y^{(j)}$)에 수렴해가고 만약 저장된 입력벡터들이 서로 직교(orthogonal)한다면 서로 다른 $X^{(s)}$와 $Y^{(j)}$에서 $X^{(s)T}X^{(j)}=0$이므로 $Y'^{(j)}$는 $Y^{(j)}$가 되어 더욱 정확한 연상이 가능하다. 출력 이진신호를 만들기 위해서 $Y'^{(j)}$을 문턱값으로 자르는데 입력 합이 $Y'^{(j)}$의 문턱값을 초과하면 $Y'^{(j)}$의 출력은 1이고, 문턱값보다 작으면 $Y'^{(j)}$의 출력은 0, 문턱값과 같으면 $Y'^{(j)}$는 현재의 상태를 그대로 유지한다. 구해진 패턴 $Y'^{(j)}$는 연결강도를 통해서 X층의 $X'^{(j)}$로 출력신호가 나간다. X층에서 Y패턴을 구하기 위해서는 W를 사용하고 Y층에서 X패턴을 구하기 위해서는 W^T를 사용한다.

$$
\begin{aligned}
X'^{(j)} &= Y'^{(j)}w_{j1}^{T} + Y'^{(j)}w_{j2}^{T} + \cdots + Y'^{(j)}w_{jj}^{T} + \cdots + Y'^{(j)}w_{jm}^{T} \\
&= Y'^{(j)}w_{ii}^{T} + Y'^{(j)}w_{j1}^{T} + \cdots + Y'^{(j)}w_{jm}^{T}
\end{aligned}
\tag{27b}
$$

이렇게 구해진 각 $X'^{(j)}$는 합해진 입력들로부터 문턱함수를 이용하여 이진신호를 만들고, 그것을 다시 Y층으로 돌려보내는 방식으로 반복하여 에너지가 최소화되는 점으로 빠르게 수렴한다.

3. 양방향 연상 기억장치의 설계

시스템을 ASIC 설계기술을 이용하여 하드웨어로 구현하는 방법에는 여러 가지가 있는데 VHDL을 이용하여 모델링하고 FPGA로 칩을 구현하는 방식으로 양방향 신경회로망을 설계하였다(부록참조). 먼저 양방향 연상 기억장치의 구조와 동작특성을 살펴보고 그에따른 설계사양을 결정한다. FPGA로 구현할 수 있는 칩의 크기를 고려하여 X층과 Y층의 뉴론의 개수는 8개와 4개로 제한하여 설계하였다.

X층과 Y층의 패턴입력을 위한 신호(x, y)와 시스템의 동작을 제어하기 위한 신호(clr), 내부신호들을 발생시키기 위한 일정한 신호(clk)를 인가할 수 있게 모두 14개의 입력포트를 구성하고, 연상을 끝낸 패턴을 출력하기 위하여 출력포트를 12개로 구성하였다.

양방향 연상 기억장치의 구현을 위하여 다섯 개의 블록으로 나누어 설계하였는데 각 블록은 VHDL로 코딩하였고, 블록마다 구조적 시뮬레이션을 통하여 기능을 검증하고, 검증이 끝나면 합성을 하였다. 합성된 회로를 가지고 게이트레벨 시뮬레이션을 하여 구조적 시뮬레이션과 같은 결과가 나타나는지를 검증하고, 검증된 블록들을 전체 합성하여 FPGA로 구현하기 위한 네트리스트를 생성하였다.

그림 10은 양방향 연상 기억장치의 전체 회로도이다. 양방향 연상 기억장치의 다섯 개의 블록은 상태발생기블록, 입력블록, 기억블록, 연산블록, 출력블록이다.

상태발생기블록은 상태신호를 발생하여 정해진 규칙에 따라 일정하게 동작할 수 있도록 내부에 신호를 공급하는 블록이고, 입력블록은 연상해낼 패턴을 입력하기 위한 블록으로 in_reg 블록과 b_input 블록으로 구성되어 있다. 기억블록은 미리 계산된 대표패턴을 저장하고 연산과정 동안 저장된 값을 신호에 맞게 출력하는 블록으로 b_rom 블록과 b_ad 블록이 있으며, 연산블록은 대표패턴과 입력패턴간의 벡터곱에 대한 연산을 수행하는 블록이고, 출력블록은 연상을 통하여 오류가 제거된 패턴을 출력하는 블록으로 b_comp 블록과 out_reg 블록으로 이루어져 있다.

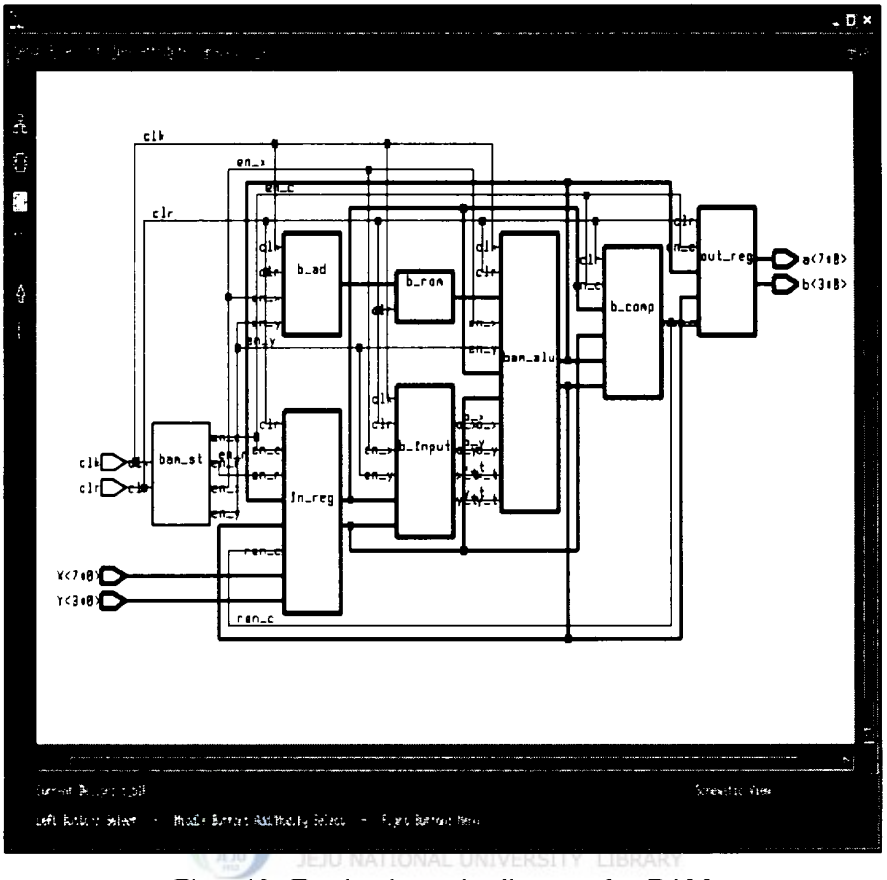


Fig. 10. Total schematic diagram for BAM

1) 상태발생기블록

상태발생기는 외부에서 입력되는 클럭을 가지고 두 개의 층(8개의 뉴론과 4개의 뉴론)을 가진 연상 기억장치를 구동시키기 위한 en_x, en_y 신호를 발생시키고, 정확한 연상을 위한 반복을 제어하는 en_c 신호와 외부 입력패턴과 연산을 통해 구해진 입력패턴의 제어를 위한 en_r 신호를 발생시킨다.

그림 11은 상태발생기블록의 심볼과 구조적 시뮬레이션으로 검증된 후 게이트 레벨로 합성된 모습을 보여준다.

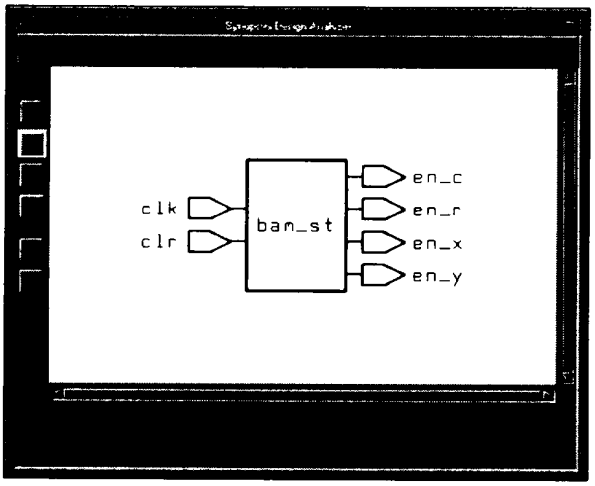


Fig. 11a. Symbol diagram of State generator

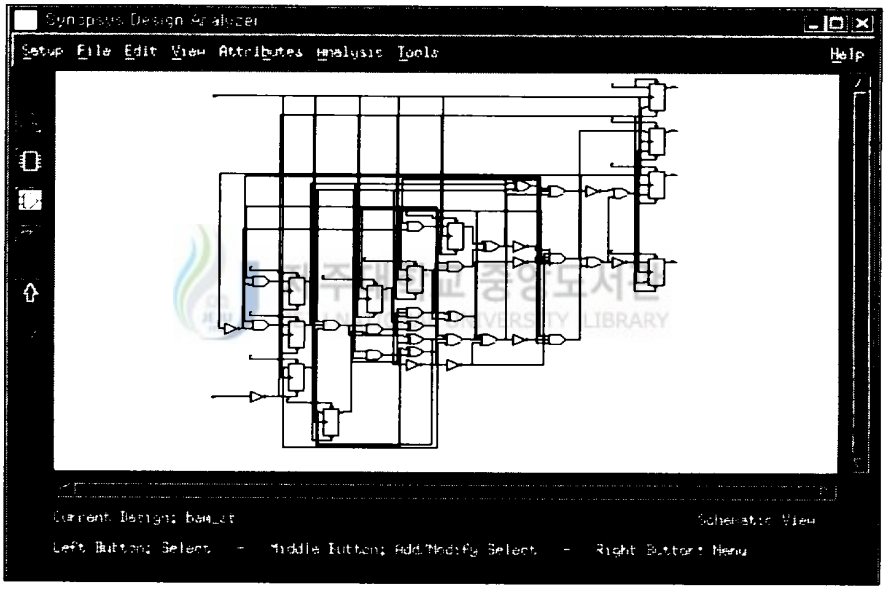


Fig. 11b. Schematic diagram of State generator

clr : 상태발생기의 동작을 정지시키는 신호이고, 신호들의 초기상태를 결정해 주는 신호.

clk : 외부에서 일정하게 들어오는 클럭 신호.

en_x : 입력패턴 X를 입력 가능하게 하는 신호로 clr이 '0'일 때, clk가 32번 동작할 동안 '1'의 상태를 유지한다.

en_y : 입력패턴 Y를 입력 가능하게 하는 신호로 clr이 '0'이고, en_x가 '0'일 때 clk가 32번 동작할 동안 '1'이 되는 상태를 반복한다.

en_c : en_x와 en_y가 한번씩 동작(clk가 64번 동작했을 때)하면 한 주기라고 하는데 한 주기가 지날 때 마다 발생하는 신호로 비교기의 동작을 제어한다.

en_r : 외부 입력패턴과 연산을 통해 들어온 입력패턴을 제어하는 신호.

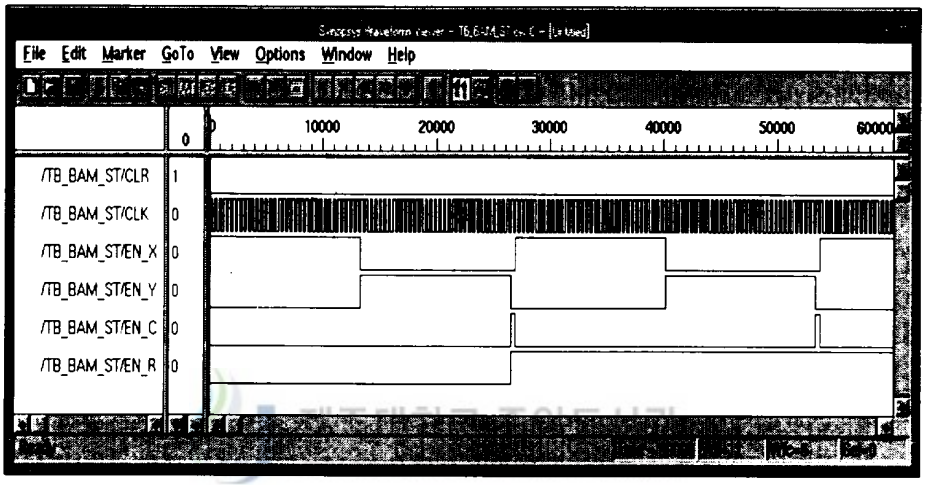


Fig. 12. Output of State generator

VHDL을 이용하여 그림 12와 같은 출력을 얻는 회로를 설계할 경우 언어의 입장에서 보면 여러 가지 방법이 있다. if 구문만을 이용하여 설계할 수도 있고, if 구문과 elsif 구문을 섞어서 설계할 수도 있고, case구문을 이용하여 설계하는 방법으로도 구현할 수 있다. 하지만 하드웨어의 측면에서 보면 크기와 속도 등을 고려하여 신중한 선택이 필요하다. case구문은 모든 신호의 상태를 기술해주는 ROM을 설계하는 경우에 사용하고 상태발생기의 경우에는 부적당하다. if구문만을 이용하여도 elsif구문으로 연결해 사용한 것과 같은 결과가 나타난다면 elsif구문은 if구문으로만 잘라서 설계하는 것이 좋다. elsif구문은 프로그램 상의 이해를

돕기는 좋으나 하드웨어 구현 시 Flip- Flop 소자가 더 많이 사용된다. 합성한 결과 if구문만을 사용했을 경우 게이트 수가 12개 줄어들었다. en_c와 en_r 신호를 발생시키기 위해서는 회로내부에서 카운터를 해 주는 변수가 있어야 한다. 이때 손쉽게 정수형 변수를 사용하기 쉬우나 최소한의 논리벡터(std_logic_vector) 변수를 사용하는 것이 필요 없는 셀 블록을 이용하여 합성하게 되는 것을 막고 하드웨어의 크기를 줄일 수 있다. 상태발생기블록을 합성할 경우 변수를 정수로 사용했을 때 라이브러리에 있는 add(b_st_DW01_inc_32) 셀 블록을 가져다 합성하고 전체 사용 게이트의 수가 512개인 반면, std_logic_vector(6 downto 0)를 사용해서 설계했을 경우 기존의 셀은 이용되지 않고, 전체 게이트의 수가 170개만을 사용하여 상태발생기 블록을 구현하였다.

2) 입력블록

입력블록에는 IN_REG 블록과 B_INPUT 블록이 있다.

IN_REG는 외부에서 주어진 입력패턴을 출력하거나 기대되는 완전한 출력패턴을 얻기 전에 연산한 패턴을 다시 입력으로 받아들여 출력하기 위한 블록이다.

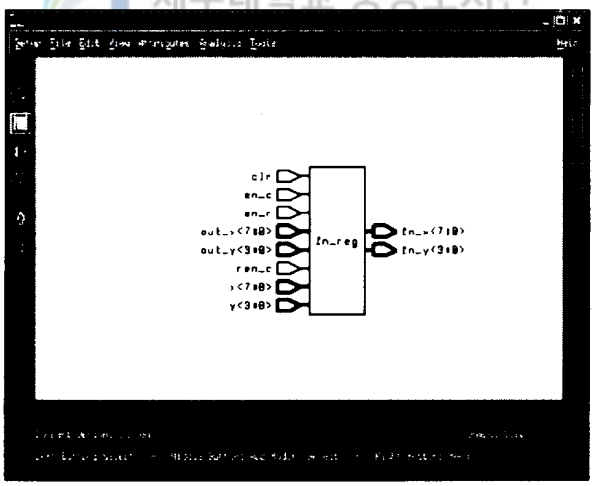


Fig. 13a. Symbol diagram of Input register

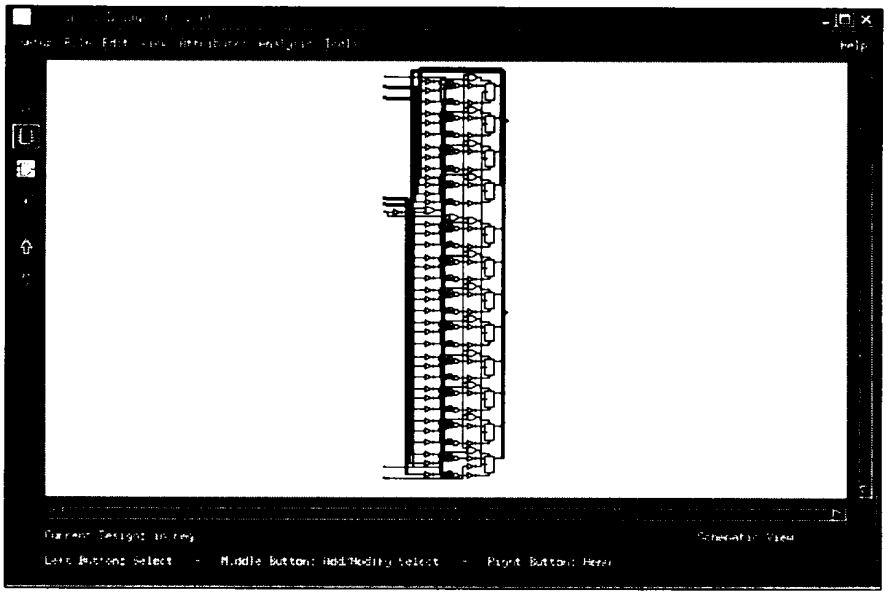


Fig. 13b. Schematic diagram of Input register

clr : 블록의 동작을 정지시키는 신호.

en_c : '0'이면 기존에 입력된 패턴을 출력하고, '1'이 되는 순간에 입력패턴을 다시 받아들여 출력한다.

en_r : 외부 입력패턴과 연산을 통해 들어온 입력패턴을 제어하는 신호로 외부 패턴이 입력되는 한 주기만 '0'이고 이후부터는 '1'이다.

out_x, out_y : 연상과정을 거쳐서 들어온 X층과 Y층의 입력패턴 신호.

x : 8개 뉴론의 외부 입력패턴 신호.

y : 4개 뉴론의 외부 입력패턴 신호.

ren_r : 비교기의 출력신호로 신호가 '0'이면 더 이상의 연상할 필요가 없으므로 버퍼가 출력을 내보내지 않고, '1'이면 out_x와 out_y가 in_x, in_y로 출력된다.

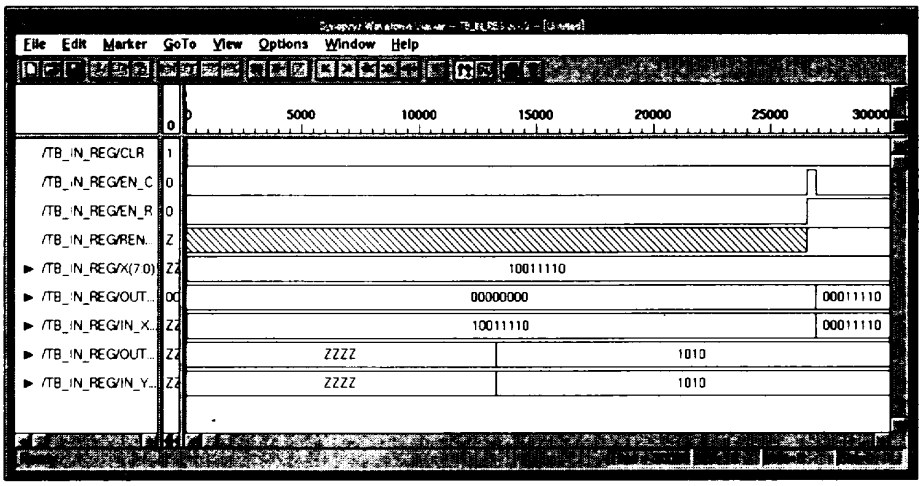


Fig. 14. Output of In_reg

B_INPUT은 입력패턴과 기억된 연결강도의 벡터곱을 위하여 입력패턴을 뉴론(1비트)단위로 출력하고, 연상작용으로 출력패턴을 구하기 위한 입력패턴과 연결강도의 곱에 따른 행과 열의 구분을 위한 신호를 내보내는 블록이다.

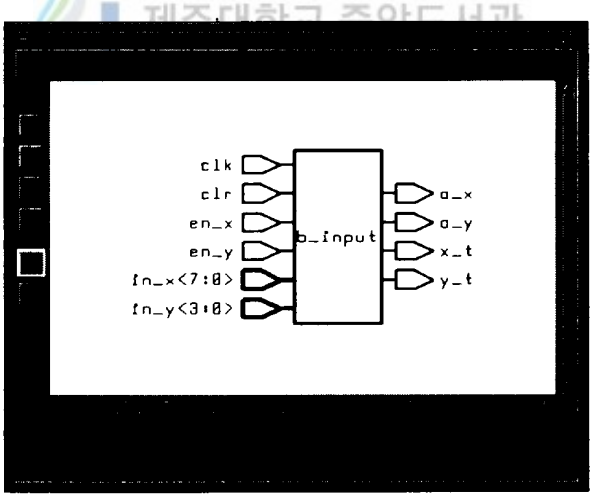


Fig. 15a. Symbol diagram of B_input

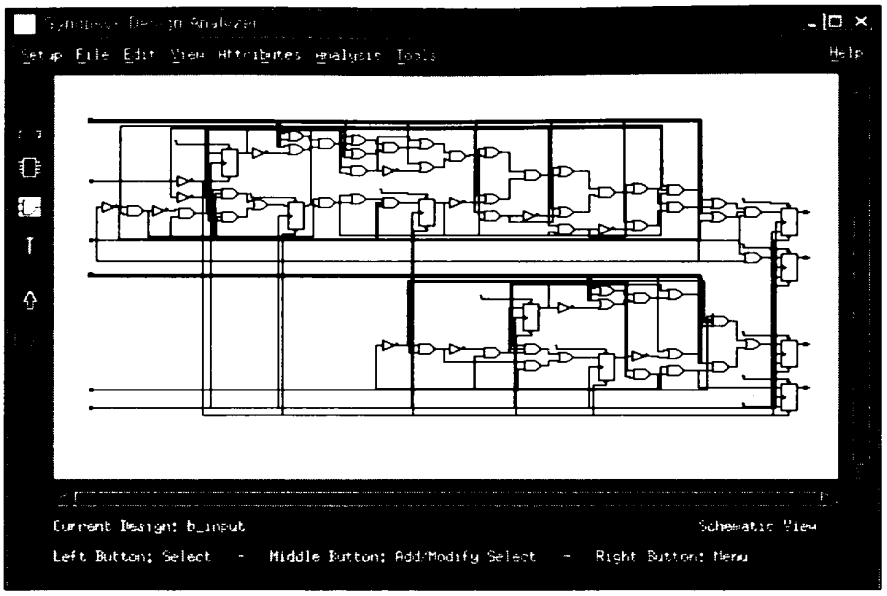


Fig. 15b. Schematic diagram of B_input

o_x : IN_REG 블록에서 나온 입력패턴 in_x를 en_x가 '1'일 때 비트(뉴론)단위로 출력한다.

o_y : IN_REG 블록에서 나온 입력패턴 in_y를 en_y가 '1'일 때 비트 단위로 출력한다.

x_t : 연상된 출력 Y패턴을 구하기 위한 신호로 X 입력패턴이 행벡터(1×8)이므로 벡터곱이 실행되는 8번째 클럭이 동작할 때마다 '1'의 신호가 출력된다.

y_t : 연상된 출력 X패턴을 구하기 위한 신호로 Y 입력패턴이 행벡터(1×4)이므로 벡터곱이 실행되는 4번째 클럭이 동작할 때마다 '1'의 신호가 출력된다.

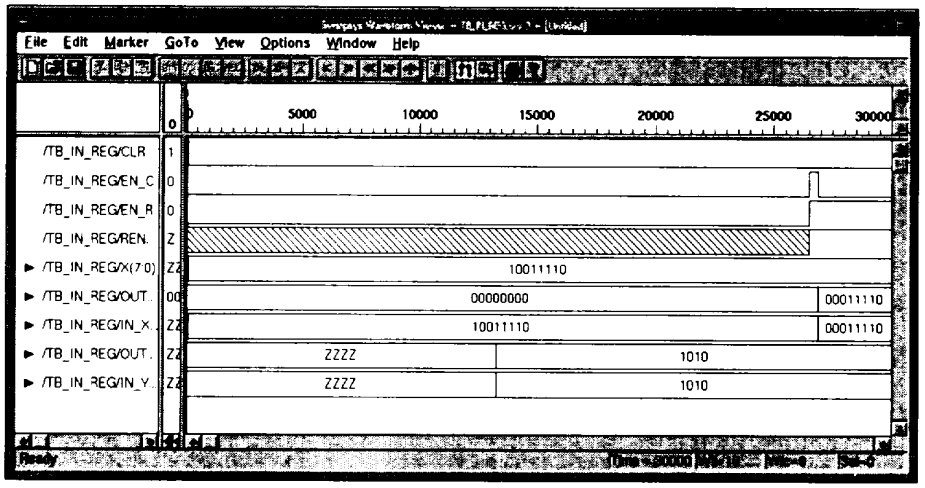


Fig. 16. Output of b_input

3) 기억블록

계산된 연결강도의 값을 저장한 B_ROM과 연결강도와 입력패턴의 곱을 위하여 B_ROM에 저장된 내용을 불러오게 하는 주소를 출력하는 B_AD가 있다.

B_ROM은 연결강도를 계산하고 그 값을 저장한다.

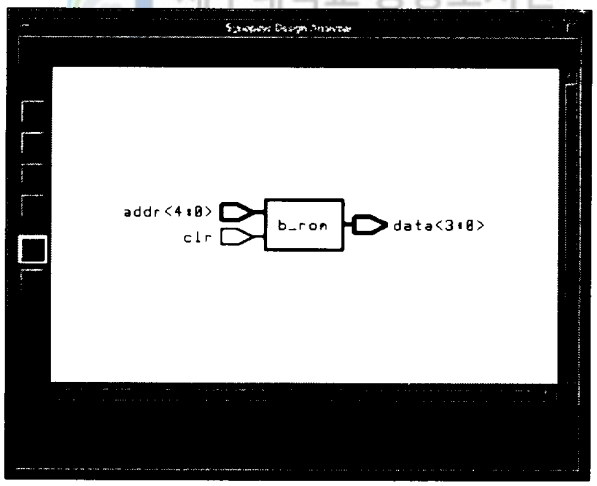


Fig. 17a. Symbol diagram of b_rom

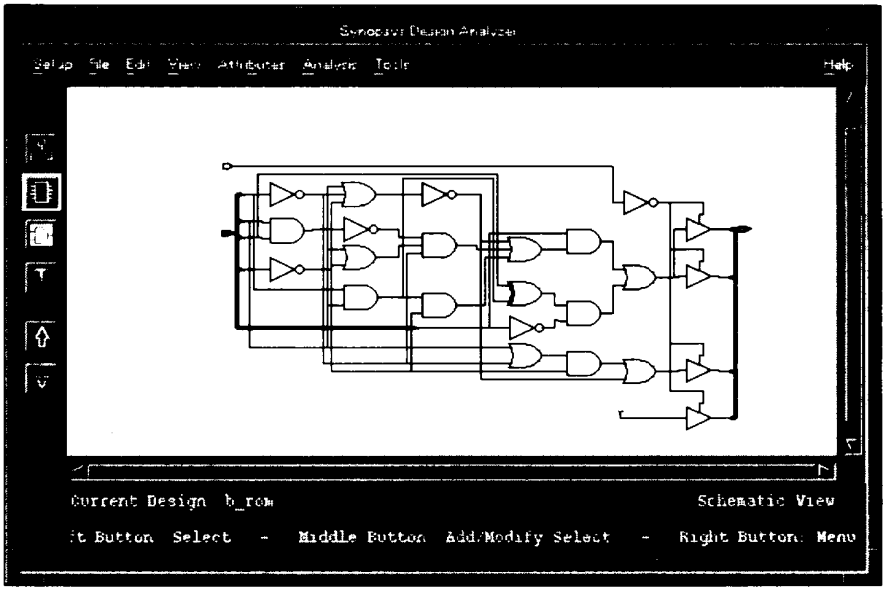


Fig. 17b. Schematic diagram of b_rom

대표패턴은 다음과 같다.

$X^{(1)}$ = { 1 0 0 1 1 1 1 0 } $Y^{(1)}$ = { 1 0 1 0 }

$X^{(2)}$ = { 0 0 1 0 0 1 1 1 } $Y^{(2)}$ = { 0 0 1 1 }

$X^{(3)}$ = { 1 1 1 0 1 0 0 1 } $Y^{(3)}$ = { 1 1 0 0 }

세 쌍의 대표패턴을 연결강도를 구하는 식을 이용하여 계산하는데 입력패턴을 쌍극성을 가진 값으로 변환시킨다.

$X^{(1)}$ = { 1 -1 -1 1 1 1 1 -1 } $Y^{(1)}$ = { 1 -1 1 -1 }

$X^{(2)}$ = { -1 -1 1 -1 -1 1 1 1 } $Y^{(2)}$ = { -1 -1 1 1 }

$X^{(3)}$ = { 1 1 1 -1 1 -1 -1 1 } $Y^{(3)}$ = { 1 1 -1 -1 }

연결강도를 구하는 식 (25)을 이용해서 저장될 패턴 p의 개수가 3인 연결강도

W를 구한다.

X패턴의 요소의 개수 N=8, Y패턴의 요소의 개수 M=4이므로 연결강도는 크기가 8행 4열인 행렬인데 ROM으로 저장할 때는 열을 따라 각 요소들을 저장한다.

ROM의 크기는 32×4bit이다.

$$W=\begin{bmatrix} 3 & 1 & -1 & -3 \\ 1 & 3 & -3 & -1 \\ -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 \\ 3 & 1 & -1 & -3 \\ -1 & -3 & 3 & 1 \\ -1 & -3 & 3 & 1 \\ -1 & 1 & -1 & 1 \end{bmatrix} \Rightarrow$$

주소	내용
00000	0011
00001	0001
00010	1111
⋮	⋮
11111	0001

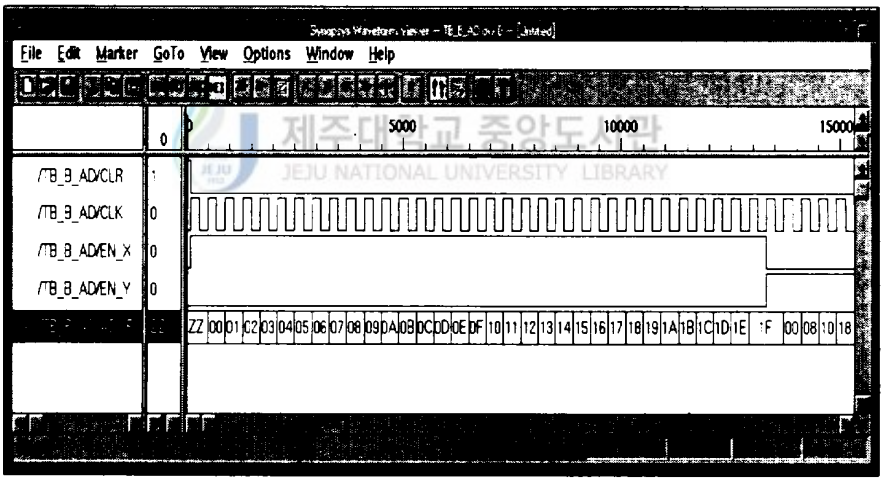


Fig. 18. Output of BAM_ADDRESS using a Synopsys

B_AD는 ROM에 저장된 연결강도를 입력패턴과의 벡터곱이 가능하도록 내용을 출력하기 위한 블록이다.

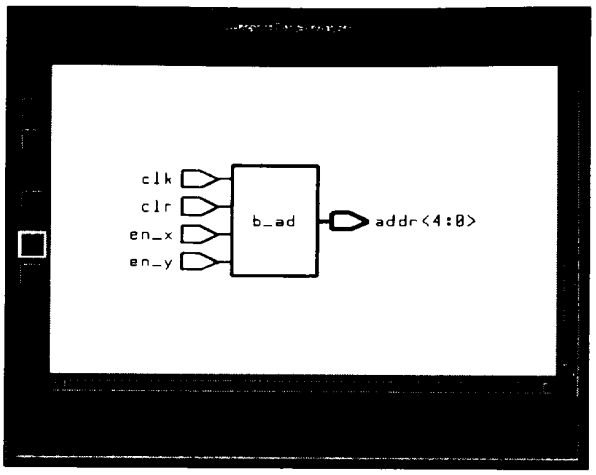


Fig. 19a. Symbol diagram of b_ad

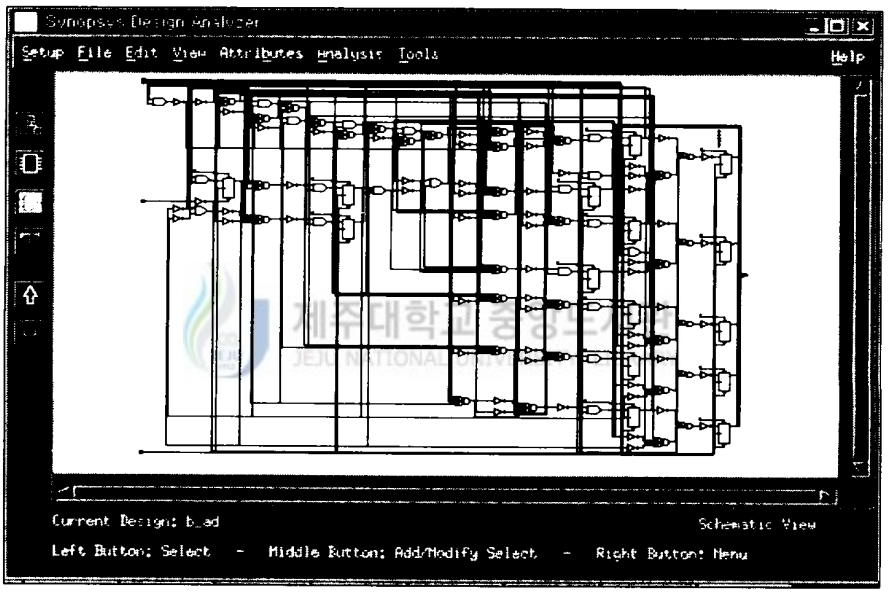


Fig. 19b. Schematic diagram of b_ad

addr : en_x가 '1'일 때 X패턴을 이용하여 출력패턴 Y를 구하는 것($Y = WX^T$)으로 연결강도의 열을 이용하여 행렬 곱을 하므로 출력은 clk가 동작할 때마다 1씩 증가하고, en_y가 '1'일 때 출력패턴 X를 구하는 것($X = W^T Y^T$)으

로 연결강도의 행을 이용하여 행렬 곱을 하므로 출력 addr은 행의 곱을 할 때는 clk가 동작할 때마다 8씩 증가하고 열의 이동을 위해 4씩 증가한다.

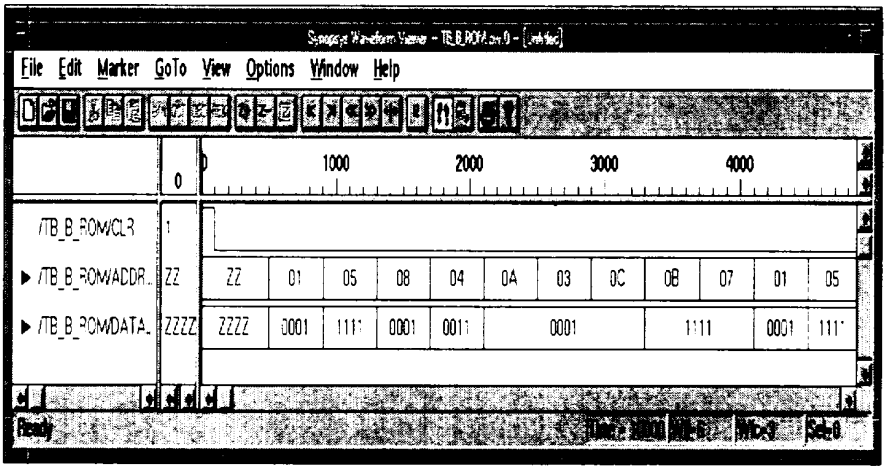


Fig. 20. Output of BAM_ROM

4) 연산블록

저장된 패턴을 연상해 내기 위하여 입력패턴과 저장된 연결강도와의 벡터곱을 하는 블록이다.

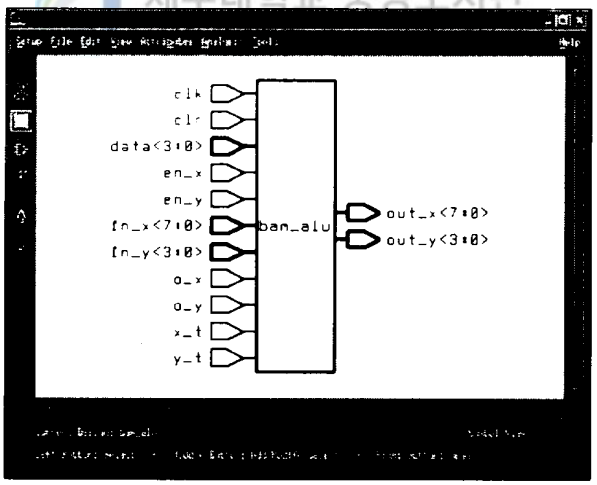


Fig. 21a. Symbol diagram of BAM_ALU

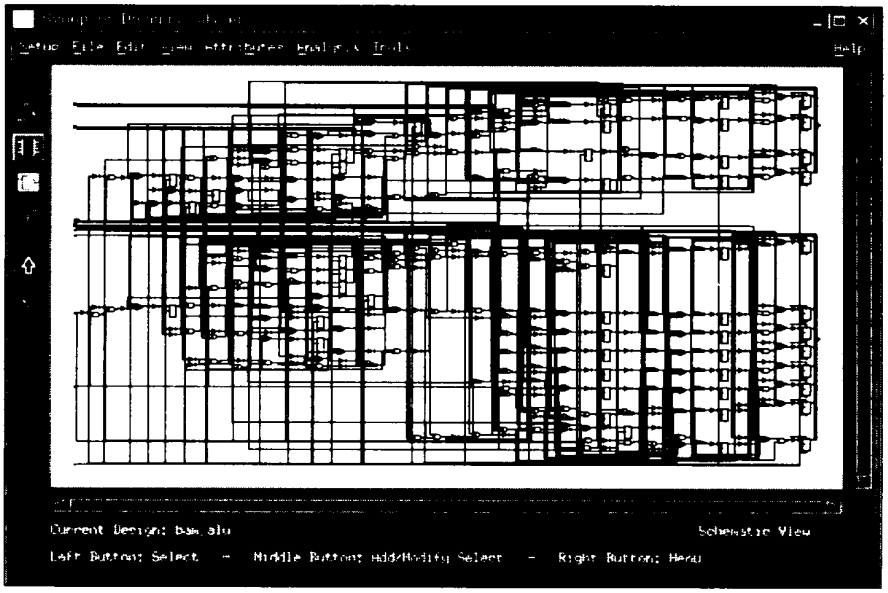


Fig. 21b. Schematic diagram of BAM_ALU

벡터곱을 하는데는 요소간의 곱셈과 덧셈이 필요하지만 입력패턴이 {0, 1}로 표현되므로 입력이 '1'이면 ROM에서 데이터를 읽어와 그 값을 더하면 벡터곱이 가능하다. 식 (27)을 이용하여 연상작용을 한다.

x_t : 8비트의 X 입력패턴을 이용하여 4비트의 y 출력벡터를 구하기 위한 제어신호로 en_x 신호가 '1' 일 때 4번의 클럭 신호가 발생한다.

y_t : 4비트의 y 입력패턴을 이용하여 8비트의 x 출력벡터를 구하기 위한 제어신호로 en_y 신호가 '1' 일 때 8번의 클럭 신호가 발생한다.

in_x, in_y : 외부에서 주어진 입력패턴이거나 연산에 의해서 구해진 불안한 out_x, out_y이 입력블록을 거쳐서 들어온 입력이다. 벡터의 곱이 '0'이면 출력은 '0'이 되는 것이 아니라 y 입력패턴의 값을 가진다.

out_x, out_y : 연산블록의 입력패턴 in_x, in_y가 ROM에서 출력된 값과 벡터곱을 한 출력으로 벡터곱의 값이 0이면 out_x, out_y 값이 0이 되는 것이 아니라 in_x, in_y 값을 유지한다.

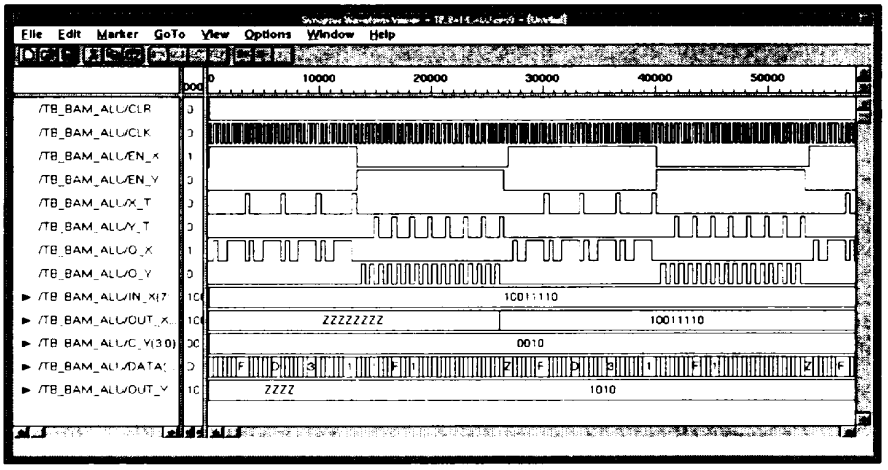


Fig. 22. Output of BAM_ALU

5) 출력블록

출력블록에는 한 주기 전의 출력패턴과 연산블록 연산과정을 거친 출력패턴을 비교하는 비교기 블록과 완전한 연상작용을 끝낸 패턴을 출력하는 출력기블록이 있다. B_COMP(비교기)는 연산블록의 입력패턴(한 주기 전의 출력패턴)과 연산과정을 거친 출력패턴을 비교하는 블록으로 출력패턴의 변화가 없으면 완전한 연상을 한 것이므로 출력기가 동작할 수 있는 신호를 출력한다.

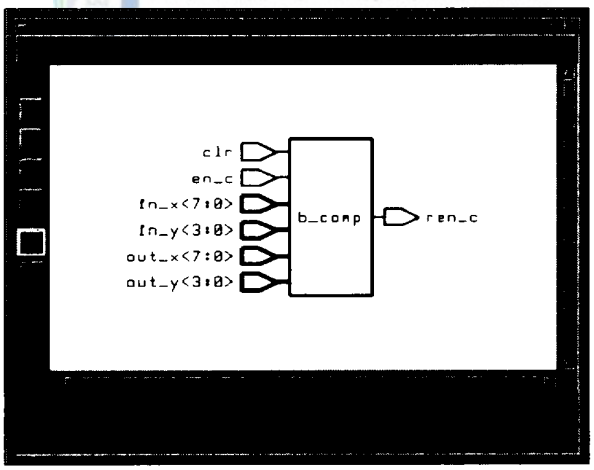


Fig. 23a. Symbol diagram of B_comp

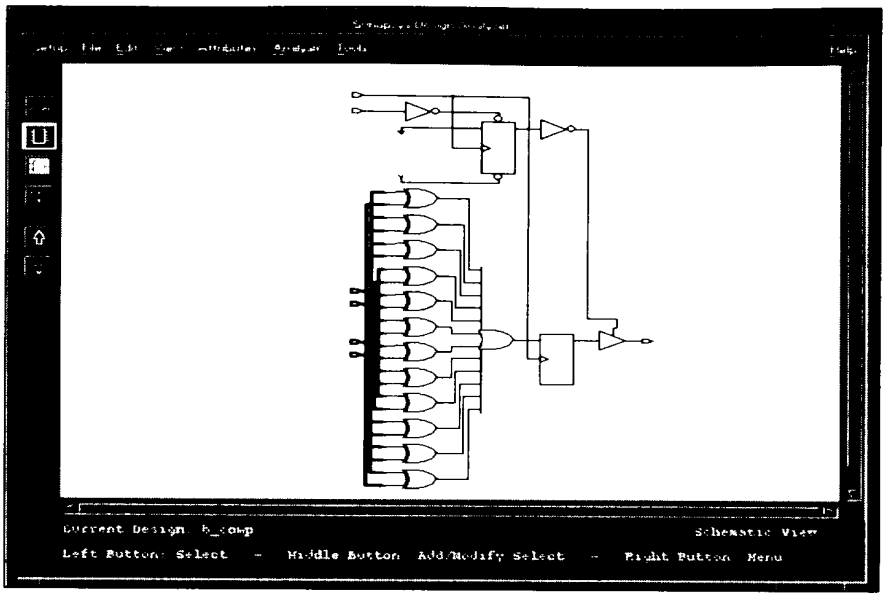


Fig. 23b. Schematic diagram of B_comp

en_c : 한번의 주기가 끝날 때마다 비교기를 동작하기 위해서 입력받는 신호.

in_x, in_y : 연산블록에서 연상과정을 거치지 않은 패턴 신호.

out_x, out_y : in_x, in_y가 연상을 위하여 연산블록에서 연산을 수행한 후에 출력된 패턴 신호.

ren_r : 저장된 패턴의 연상을 위하여 연산블록을 거친 out_x와 out_y가 완전한 출력인지 아닌지를 출력하는 신호로 out_x와 in_x, out_y와 in_y를 비교하여 모두 변화가 없으면 '0' 변화가 있으면 '1'을 출력한다.

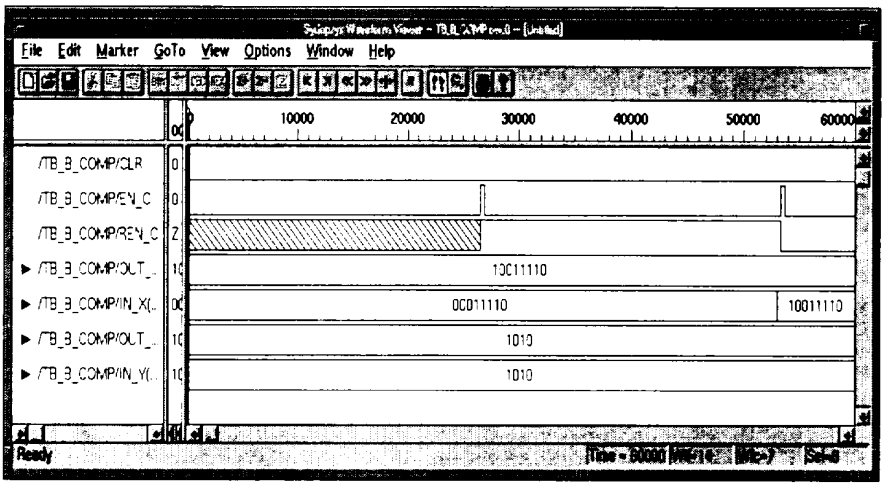


Fig. 24. Output of B_COMP

출력기는 연상과정을 거친 개선된 출력패턴이 연상과정을 거치기 전의 입력패턴(한 주기 전의 출력)과 변화가 없을 때 출력기의 입력패턴 out_x와 out_y를 출력한다.

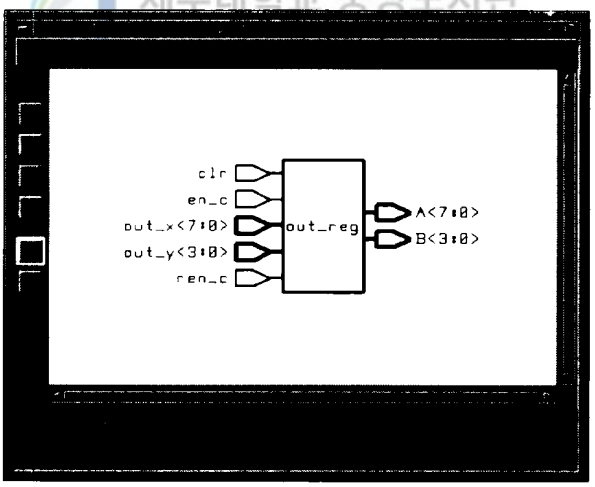


Fig. 25a. Symbol diagram of OUT_REG

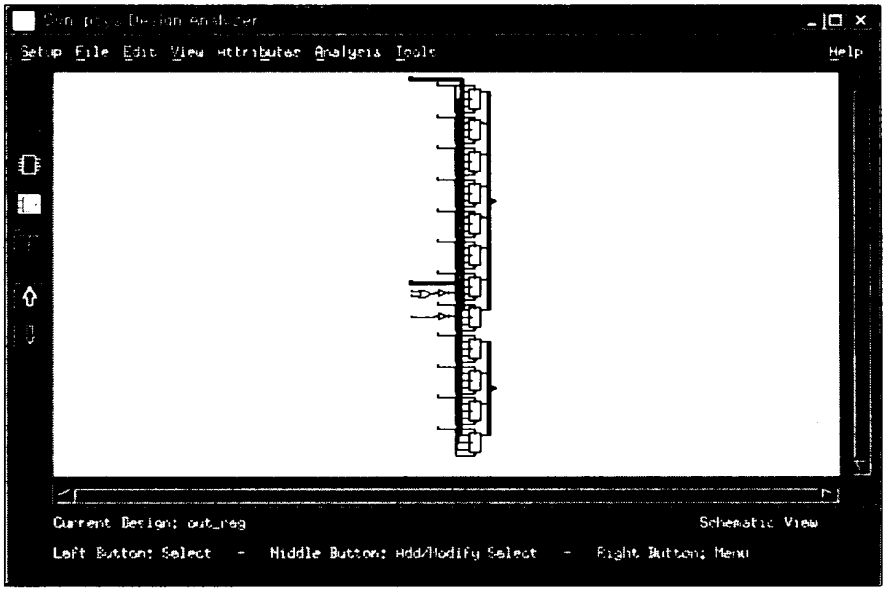


Fig. 25b. Schematic diagram of OUT_REG

A, B : en_c가 '1'이고, 비교기의 출력인 ren_r이 '0'이면 out_x, out_y를 출력한다.

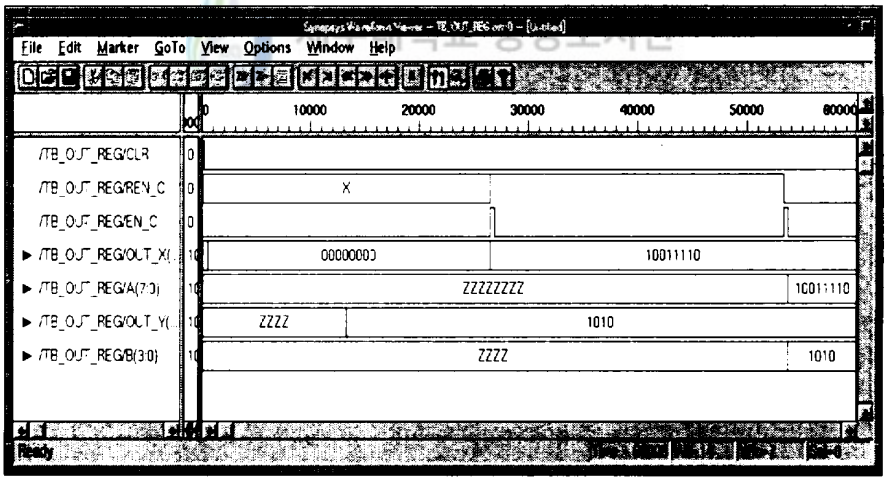


Fig. 26. Output of OUT_REG

5. 시뮬레이션 결과.

양방향 연상 기억장치는 두 개의 층으로 이루어져 있는데 비슷한 환경에서는 각 층의 뉴론의 개수가 많을수록 더 정확한 연상을 할 수 있다. 그러나 하드웨어로 구현하는 데는 구현할 소자의 크기에 따라 한계가 있고, 확장성이 있는 블록 설계를 FPGA로 간단히 제작하기 위해서 8개의 뉴론을 가진 X층과 4개의 뉴론을 가진 Y층으로 이루어진 시스템을 구성하였다.

앞 절에서 저장될 패턴의 수는 두 개의 층의 뉴론 중에서 작은 수의 뉴론에 영향을 받는다($p < \min(n, m)$)는 것을 알았다. Y층의 뉴론의 개수가 4개이므로 안정된 연상을 위해서는 4개의 패턴을 저장하고, 좀더 안정된 패턴의 연상을 위해서는 2개의 패턴을 저장하는 것이 좋은데, 3개의 패턴을 이용하여 양방향 연상 기억장치를 설계했다. X층과 Y층의 뉴론의 개수를 증가시키면 저장할 수 있는 대표패턴의 수는 더욱 커진다.

연결강도를 구하기 위한 X층과 Y층의 세 개의 대표패턴은 아래와 같고, 연결강도를 구하기 위하여 이진형태를 쌍극형태로 변환시킨다. 그림 27은 대표패턴의 뉴론을 '0'일 때 흰 사각형, '1'일 때 검은 사각형인 그림으로 표현한 것이다.

$$X^{(1)} = \{\ 1\ 0\ 0\ 1\ 1\ 1\ 1\ 0\ \} \qquad Y^{(1)} = \{\ 1\ 0\ 1\ 0\ \}$$
$$X^{(2)} = \{\ 0\ 0\ 1\ 0\ 0\ 1\ 1\ 1\ \} \qquad Y^{(2)} = \{\ 0\ 0\ 1\ 1\ \}$$
$$X^{(3)} = \{\ 1\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ \} \qquad Y^{(3)} = \{\ 1\ 1\ 0\ 0\ \}$$

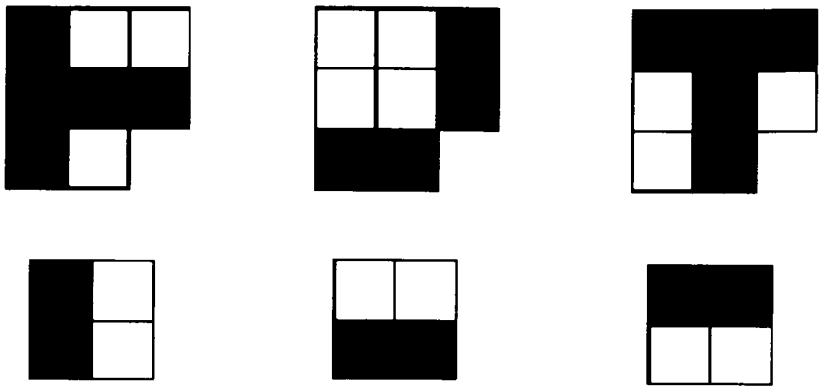


Fig. 27. Input patterns

먼저 시스템이 제대로 동작하고 있는지를 살펴보기 위하여 왜곡이 없는 세 개의 대표패턴을 입력패턴으로 하여 양방향 신경회로망을 시뮬레이션 한 결과 출력은 다음과 같다.

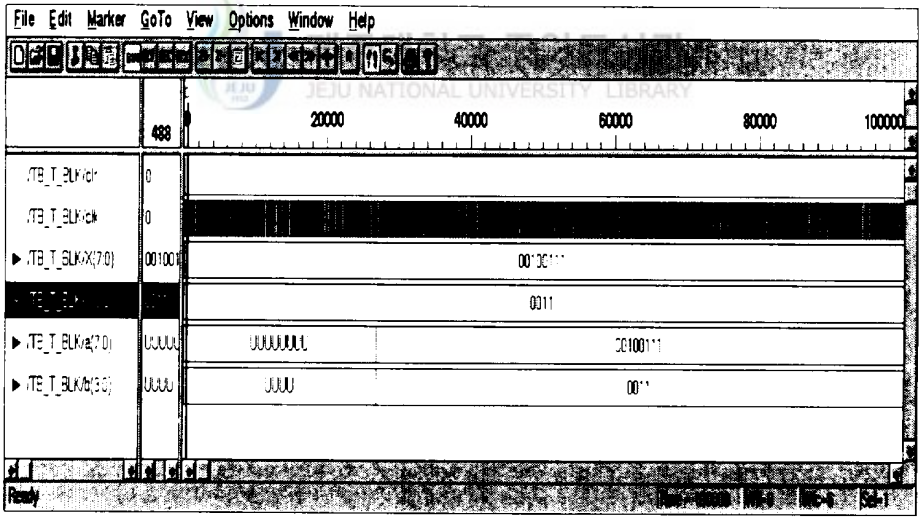


Fig. 28. Output of total block using a $X^{(1)}$ input pattern

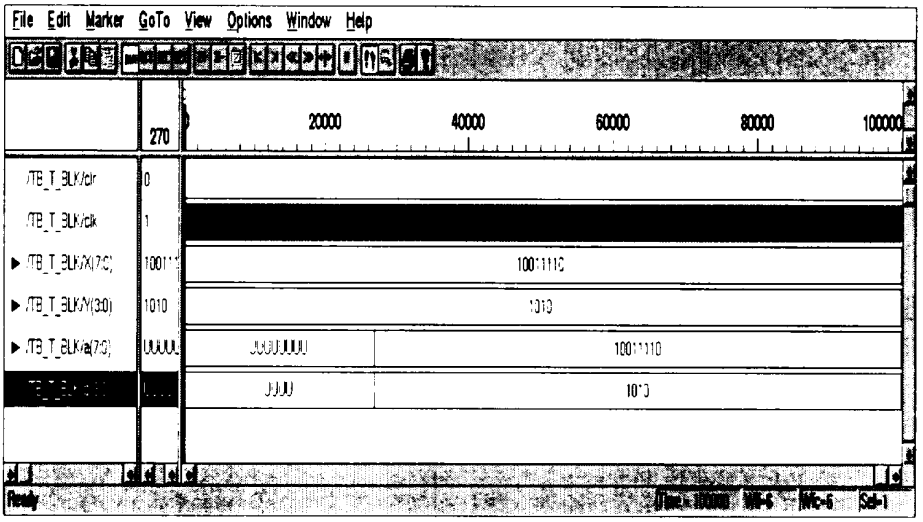


Fig. 29. Output of total block using a $X^{(2)}$ input pattern

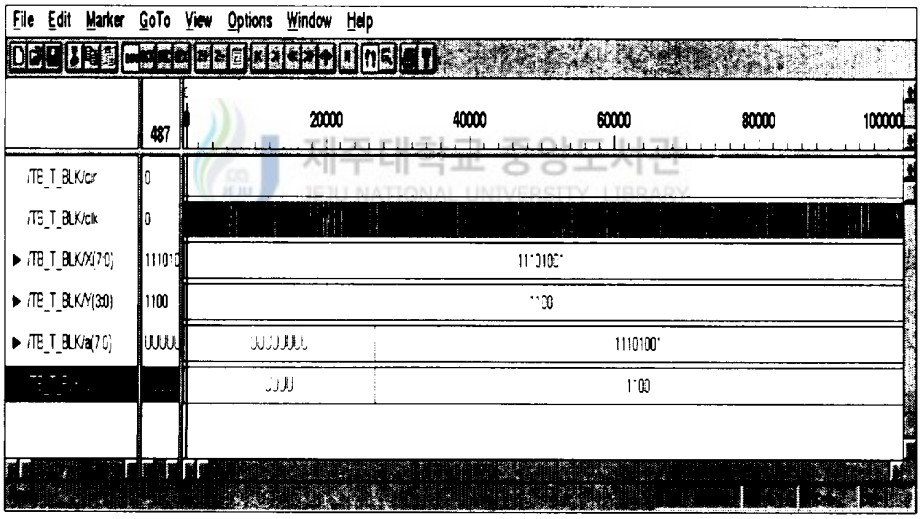


Fig. 30. Output of total block using a $X^{(3)}$ input pattern

⇒ 왜곡이 없는 패턴을 입력했을 때 같은 패턴을 연상해 냄을 보였다.

X 입력패턴에 하나의 왜곡이 있을 때 양방향 신경회로망의 출력은 다음과 같다.

$X^{(1)}\{1\ 0\ 0\ 1\ 1\ 1\ 1\ 0\}$ ---> $X = \{1\ 1\ 0\ 1\ 1\ 1\ 1\ 0\}$ $Y = \{0\ 0\ 0\ 0\}$

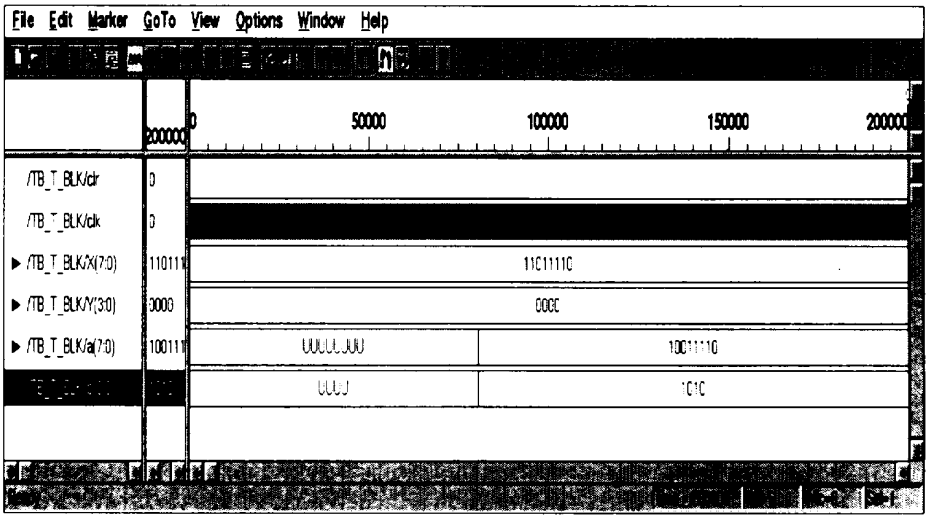


Fig. 31. Output of total block using a $x^{(1)}$ input pattern

X 입력패턴에 두개의 왜곡이 있을 때 양방향 신경회로망의 출력은 다음과 같다.

$X^{(1)}\{1\ 0\ 0\ 1\ 1\ 1\ 1\ 0\}$ ---> $X = \{1\ 1\ 1\ 1\ 1\ 1\ 1\ 0\}$ $Y = \{\ 0\ 0\ 0\ 0\ \}$

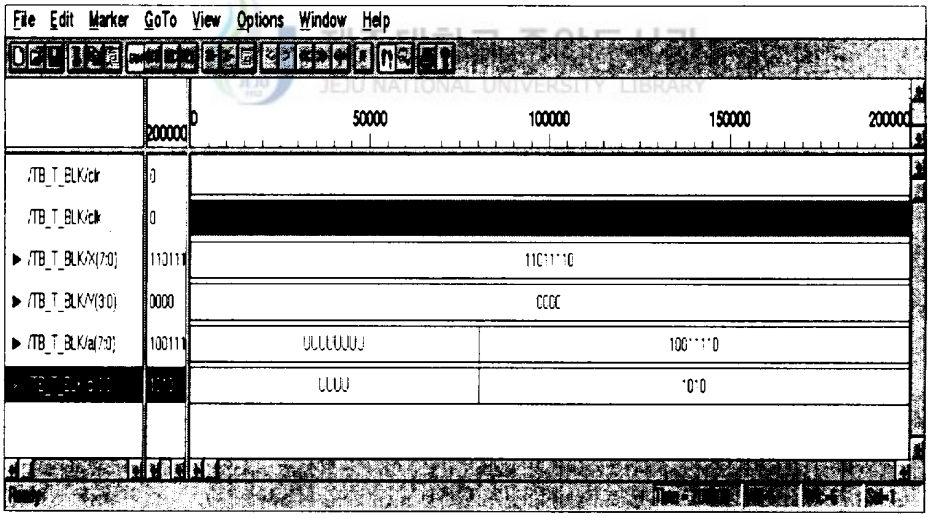


Fig. 32. Output of total block using a $x^{(1)}$ pattern with noise

$X^{(2)}$ = { 0 0 1 0 0 1 1 1 } ---> X = {1 1 1 0 0 1 1 1} Y = { 0 0 0 0 }

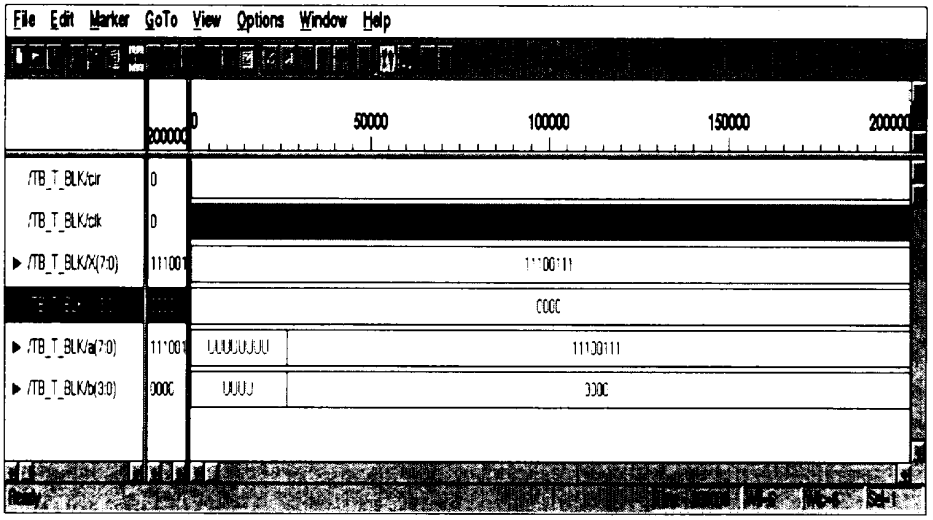


Fig. 33. Output of total block using a $x^{(2)}$ pattern with noise

$X^{(2)}$ = { 0 0 1 0 0 1 1 1 } ---> X = {1 1 1 0 0 1 1 1} Y = { 0 0 1 1 }

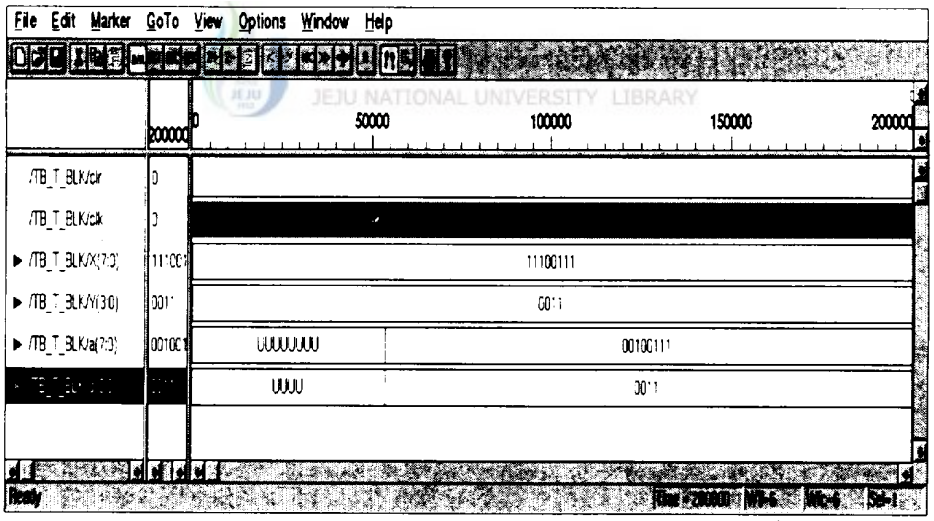


Fig. 34. Output of total block using a $x^{(2)}$ pattern with noise

그림 33의 결과를 살펴보면 오류가 있는 입력패턴이 저장된 대표패턴을 찾아내지 못함을 보여주고 있다. 이것은 연상해내는 복호화 과정에서 구하려는 뉴론의 벡터합의 값이 0이 되었을 경우 연상되기 전의 값을 갖기 때문이다. 예를 들어 설명하면 오류가 있는 X패턴이 입력되었을 때 구해진 값이 { 0 4 −4 0 }이면 Y'값은 { 0 1 0 0 }이 아니라 { y_1 1 0 y_4 }이 된다. 그림 33의 패턴의 경우 Y'값은 { y_1 y_2 y_3 y_4 }인 경우가 되고, Y패턴을 { 0 0 0 0 }로 입력했기 때문에 Y'값은 { 0 0 0 0 }이 되어 정확히 연상하지 못한다. 이러한 경우 Y패턴을 오류가 없는 값으로 입력하면 그림 34와 같이 저장된 대표패턴을 정확히 연상해 내는 것을 알 수 있다.

V. 결 론

본 논문에서는 궤환성을 갖는 단층신경망인 Hopfield 아날로그 신경회로망의 계산에너지함수를 분석하여 계산에너지함수에 존재하는 제 3항을 제거하는 새로운 방법을 제안하고, 디지털 신경회로망인 양방향 연상기억장치를 ASIC 설계기술 중에서 하드웨어 설계언어를 이용하여 논리적으로 구현하였다.

우선, 종전의 Hopfield 아날로그 신경회로망의 경우 시그모이드 이득률을 작게 하였을 경우 제 3항의 에너지가 커지면서 그에 따른 시스템이 오류를 범하는 경우가 증가하게 되며, 시그모이드 이득률을 큰 값으로 하였을 경우 뉴론의 출력값이 너무 급히 변하여 단위함수에 가깝게 되어 시스템이 수렴하는 데 불안해지는 문제점이 있었다. 본 논문에서는 뉴론의 입력단에 종속전류원을 연결하여 시스템의 계산에너지함수에서 제 3항을 제거하였다. 그리하여, 시스템의 에너지함수를 최적화하려는 비용함수에 보다 근접하게 함으로써, 시스템의 성능을 개선시켰으며, 시뮬레이션을 통하여 성능이 향상되었음을 보였다.

디지털 신경회로망인 양방향 연상 기억장치를 하드웨어로 구현하기 위하여 ASIC 설계하였다. 양방향 연상 기억장치는 VHDL을 이용하여 다섯 개의 블록으로 설계하였는데 각각의 블록들은 구조적 시뮬레이션을 통하여 검증하였다. 검증된 블록들을 전체 합성하고 게이트레벨로 시뮬레이션하여 재검증하였다. 이러한 과정을 통하여 각 블록들은 개별적으로 유사한 동작을 하는 다른 시스템에서 이용될 수 있고, 더 큰 시스템으로의 확장도 쉽게 이루어 질 수 있다.

앞으로 연구해야 할 점은 성능개선된 Hopfield 아날로그 신경회로망을 ASIC 설계기술을 이용하여 구현하고, 병렬성이 크며 부호화가 가능한 양방향 신경회로망을 구현하고, 다양한 여러 가지 디지털 시스템들을 ASIC 설계기술로 구현하는 것 등이다.

참고문헌

ALTERA, 1992, "MAX+PLUS Ⅱ Getting started," ALTERA corporation.

Anthony N. Michel, Jay A. Farrell, 1990, "Associative Memories via Artifical Neural Networks" IEEE Control System Magazine pp 6-17.

Bart Kosko. 1988, "Bidirectional Associative Memories" IEEE. pp. 49-60.

Bart Kosko. sept. 1987, "Constructing an Associative Memory" byte. pp. 136-144.

Bernard C, Levy 1987, "Global optimization with Stochastic Neural Networks", IEEE, Int. Conf. on NN. Sandiego, CA. USA.

Chin-Teng Lin, C. S. George Lee. 1996, "Neural Fussy Systems," Prentice Hall P T R Prentice-Hall, Inc.

최 명 렬, 1996, "주문형 반도체 설계 ASIC DESIGN" 하이테크정보

Dae Soo Kim, 1995, "Neural Networks Theory and Applications Ⅰ,Ⅱ" Hiteck Information company.

Fausett, Lausett, Laurene V. 1994, "Fundamentals of neural networks : architectures, algorithms," Englewood Cliffs, Prentice-Hall.

Hopfield, J. J., and D. W. Tank. 1986. "Computing with Neural Circuits: A Model," Science 233: 625-633.

Hopfield, J. J. 1984. "Neurons with Graded Response Have Collective Computational Properties Like Those of Two State Neurons," Proc. National Academy of Sciences 81: 3088-3092.

Hopfield, J. J., and D. W. Tank. 1985. "Neural" Computation of Decisions in Optimization Problems," Biolog. Cybern. 52: 141-154.

Jacek M. Zurada. 1992, "Introduction to Artificial Neural Systems," West Publishing Company.

조 중 휘, 홍 윤 식, 정 연 모, 김 영 철, 1996, "VHDL 기초와 응용," 반도체설계 교육센터.

Kim Tae-Hyoung, Park Jae-Young, Lee Cheon-Woo, Park In-Jung, "The Implementation of Neural Network Chip," ICNN pp. 395-399.

고 경 희, 강 민 제, 1996 "계산에너지함수 분석을 통한 Hopfield 신경회로망의 최적화" 제 5회 인공지능, 신경망 및 퍼지시스템 학술대회.

고 경 희, 강 민 제, 1997 "계산에너지함수 분석을 통한 궤환성을 갖는 단층신경회로망의 성능개선" 전자공학회. pp. 54-60

Park, S. 1989. "Signal Space Interpretation of Hopfield Circuit and a Linear Programming Circuit," IEEE Trans. Circuits and Systems CAS-33(5): 533-541.

박 현 철, 1995, "VHDL 회로설계와 응용" 한성출판사.

Roger Lipsett, Carl F. Schaefer, Cary Ussery, 1993, "VHDL : Hardware Description and Design," Intermetrics Inc.

SYNOPSYS, "Design Analyzer Reference" Synopsys Inc.

Zurada, J. M., and M. J. Kang. 1988. "Summing Networks Using Neural Optimization Concept," Electron. Lett. 24(10): 616-617.

Zurada, J. M., and M. J. Kang. 1988. "Computational Circuits Using Neural Optimization Concept," Int. J. Electron. 67(3): 311-320.

부 록

A. 주문형 반도체(ASIC)

1) 주문형 반도체의 개요

ASIC(Application Specific Integrated Circuit)은 원어를 그대로 해석하면 특정 용도 집적회로이나 우리 나라에서는 주문형 반도체라 일컬어지고 있다. 일반적으로 널리 사용되는 표준형 IC 공정기술과는 달리 특정 용도로 칩을 설계하여 주거나 쉽고 빠르게 설계할 수 있는 환경을 제공해주는 기술이다.

ASIC을 위한 기술은 공정기술과 설계기술의 두 가지로 나눌 수 있다.

공정기술은 ASIC 칩을 제조하기 위한 반도체 제조 공정기술을 의미하며, 보통 기억소자(memory)용 공정기술보다 한 세대 뒤떨어지는 공정기술을 사용하는데 TTL로 널리 사용되어온 Bipolar기술, 기억소자 등의 디지털 초고집적회로(VLSI)용 공정기술인 MOS기술, Bipolar기술과 CMOS기술의 장점을 취한 BiCMOS기술, 초고속용 화합물 반도체 기술이 GaAs기술이 있다. 설계기술은 PC나 워크스테이션에서 ASIC용 CAD(Computer Aided Design) 소프트웨어(CAD tool)를 사용하여 시스템 설계, 모듈 설계, 회로 설계, 단위 셀 설계 등의 특정용도 집적회로를 설계하는 기술로 VHDL 등의 고 수준 언어를 이용한 시스템/모듈 설계, 여러 가지 모의 실험(simulation) 툴을 이용한 설계의 검증 및 회로 설계, 칩 제조에 사용될 마스크 패턴을 위한 레이아웃 설계(layout design) 등 전문적인 지식을 요구한다.

2) 주문형 반도체의 분류

ASIC은 시대와 함께 변할 수 있으나 일반적으로 그림 9과 같이 분류한다.

넓은 의미의 ASIC은 단일 고객을 위한 ASCP(Application Specific Custom

Products : 좁은 의미의 ASIC)와 복수 고객을 위한 ASSP(Application Specific Standard Products)로 분류되며, ASCP는 완전주문형 집적회로(full-custom IC), 일부주문형 집적회로(semi-custom IC), 프로그래머블 집적회로 (programmable IC)로 분류된다. 일부주문형 집적회로는 표준셀 및 게이트 어레이가 대표적으로 열거된다.

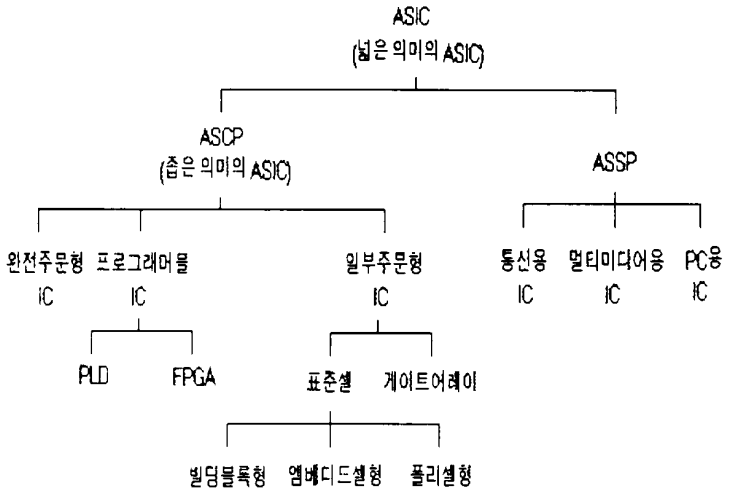


Fig. 9. Classification of ASIC

ASCP는 사용자 한 명의 주문에 따라 제조되는 집적회로로 좁은 의미의 ASIC으로 완전주문형 집적회로의 설계는 회로 설계자가 요구되는 기능을 가진 회로의 하나 하나를 설계하고, 캐드 툴을 사용하여 칩 제조 시 사용되는 마스크 패턴인 집적 회로의 레이아웃을 일일이 수 작업을 통하여 그리고, 설계된 회로들을 다시 수 작업을 통하여 배치함으로써 이루어지므로 집적회로를 설계하는데 오랜 시간이 걸리고, 오류를 범할 가능성도 높지만 가장 적은 면적의 실리콘 상에서 회로 설계가 가능하므로 대량생산을 할 경우 단가가 가장 저렴하게 된다.

일부주문형 집적회로의 설계는 표준화된 일부의 설계와 반도체 공정기술을 사용하는 방법으로 반도체 양산 공정기술에 맞게 설계된 표준셀(standard cell)들이

나, 트랜지스터 혹은 게이트들을 규칙적으로 배열하여 범용적으로 사용하는 어레이(array)등을 이용하여, 시스템 업체가 요구하는 기능의 시스템이나 회로를 단일 칩(one-chip)화하는 방법이다. 일부주문형 집적회로는 게이트 어레이(Gate Array)와 표준셀의 두 가지로 크게 나뉜다.

표준셀은 동작과 성능이 입증된 논리기능과 아날로그 회로기능을 표준셀 라이브러리에 보관, 회로설계 시 적합한 회로를 이용하여, 레이아웃 설계자동화(배치와 배선)로 반도체 전 공정을 수행하여 제작한다. 셀의 크기에 따라 빌딩블록형과 엠베디드셀형, 폴리셀형으로 분류한다. 셀은 수동 배치, 배선으로 고집적이 가능하며, 칩 크기는 완전주문형에 비해 10~15% 크지만, 게이트어레이보다 20% 이상 작다. 표준셀의 특징은 대용량 메모리를 사용할 경우에 칩의 크기가 작으며, 매가 셀이라는 이미 설계된 모듈을 이용하여 설계의 생산성을 높인다. 표준셀은 마스크를 제품마다 다르게 하는 것으로서 게이트어레이에 비해서 레이아웃의 자유도가 높고, 여러 가지의 기능을 칩 상에 실현하는 것이 원칙적으로 가능하다.

게이트어레이는 NAND, NOR와 같은 기본 논리게이트나 표준 논리소자를 규칙적으로 배열한 배선 이전까지를 미리 제조한 상태에서, 시스템업체가 요구하는 기능을 기본 블록들 사이의 금속배선용 주문형 마스크공정을 통하여 완성한다. 집적회로 제작기간이 짧으며(4~6주), 마스터슬라이스 구조개선으로 회로의 기능 및 특성에 따라 선택범위가 확대된다. 배선만을 개별로 설계함으로서 회로가 구성되기 때문에 사전에 마스터어레이를 제조(확산공정까지)하여 두는 것이 가능하다. 설계종료 후 미리 만들어진 웨이퍼를 가지고 배선공정만으로서 웨이퍼공정을 마친다. 이것은 설계종료 후 짧은 시간에 견품을 받을 수 있는 장점을 가진다.

프로그래머블 집적회로는 PLD(Programmable Logic Device)와 FPGA(Field Programmable Gate Array)로 구분된다. PLD는 수요자가 내부의 배선을 프로그램에 의해 접속시켜서 필요한 논리회로를 작성한다. AND와 OR 게이트의 어레이로 구성되어 있기 때문에 논리회로는 합과 곱의 형태로 표시되며 설계자가 개발한 마이크로 코드(micro code)등의 웨어(firmware)나 논리회로를 반도체 제조업체가 마스크 패턴을 제작하여 칩으로 제조해 주는 재 프로그램이 불가능한

PLD으로 PROM, PLA, PAL 등이 있다.

FPGA는 PLD와 게이트어레이의 장점을 이용한 것으로, PLD의 특징인 사용자 프로그램성을 유지하면서, 게이트어레이 구조를 하고 있어 고속, 고집적화가 가능하다. 이는 사용자가 개발장소에서 직접 구워냄으로써 편이성이 우수하고, 낮은 시제품 제작비용, 짧은 개발기간, 소량제품의 개발 등의 장점을 가지고 있다.

ASSP(Application Specific Standard Product)는 반도체 제조업체 측에서 특정 용도에 사용하기 위해 사용자의 사용 용도에 관계없이 시스템의 용도에 따라 독자적으로 개발, 제작한 칩으로 복수의 주문자를 대상으로 하며, 각 종류의 칩 생산량이 적은 편이다. 통신용 칩, PC용 칩, 멀티미디어용 칩, 가전제품용 칩 등의 ASSP들이 많이 판매되고 있다.

3) 주문형 반도체 칩의 설계 과정

80년대까지의 설계방법은 반도체 제조기술에 의존적인 설계로서, 게이트설계(논리설계)와 레이아웃설계가 주류였으나 90년대 초반부터는 반도체기술과 무관한 하드웨어 기술언어(HDL: Hardware Description Language)를 이용하여 회로를 설계할 수 있게 되었다. 그림 10은 ASIC 설계흐름을 보여주고 있다.

1) Front-end design 단계

하드웨어 기술 언어로 원하는 회로의 동작을 기술하고 시뮬레이션(simulation)과 합성(synthesis)을 통해 netlist를 생성한다.

① 설계 사양의 결정(Design Specification)

시스템의 응용, 응용에 요구되는 성능, 시스템 구조, 외부와의 인터페이스 및 프로토콜, 시장 경쟁력, 상품 단가, 제조 기술, 설계 기술 및 툴 등 여러 가지 중요한 요소를 고려한다.

② 동작레벨설계

시스템 기기를 만드는 사람이 ASIC으로 실현하고자 하는 ASIC의 기능, 성능, 외부 핀의 기능이나 동작타이밍을 정의하는 단계이다. 이 사양 설계 과정에서 요구하는 기능의 실현방법 등의 방식검토를 행한다. 가장 추상도가 높은 동작레벨의 설계데이터는 회로의 외부 입출력단자(신호)의 정의와 회로 전체로서 실행하는 처리의 기술로 이루어진다.

③ 기능설계

회로의 기능을 보통 하드웨어기술언어(HDL, VHDL)를 사용하여 설계하고, 기능시뮬레이션(Functional simulation)을 통하여 검증한다. 이 단계에서는 특정기술이나 공정에 대한 이해 없이도 설계가 가능하다. 여기에서는 기능구성요소의 결정, 기능블록도 작성, 동작을 나타내는 상태천이도 작성, 제어방식결정, 기능동작확인 등을 설계한다. 현재의 하드웨어기술언어는 주로 RTL 설계데이터를 기술하는 데 사용된다.

④ 논리설계

기능설계 결과를 게이트레벨(Gate level)까지 전개하는 것이 논리설계이다. 여기에서는 기능설계에서 결정되어진 논리사양을 만족하는 논리구성 및 게이트간의 접속관계를 규정한다. 검증된 기능기술을 이용하여 게이트레벨의 회로로 변환하는데 주로 자동합성 툴을 이용하여 합성하고, 게이트레벨 시뮬레이션을 통해 합성한 결과 netlist의 기능을 확인한다. 변환에는 목표로 하는 기술과 셀 라이브러리가 사용된다. 논리설계단계에서는 설계결과가 필요한 논리기능을 만족하고 있는가, 지연 및 타이밍제약이나 게이트 수 제약의 범위에 맞는가, 테스트 용이화의 배려가 되어 있는가 등이 체크된다. 합성을 사용하지 않을 경우 ASIC 공급업체로부터 제공되는 셀 라이브러리를 사용하여 사용자가 직접 논리회로를 설계한다.

셀을 사용하여 설계된 게이트어레이회로의 검증을 하는데 기능과 타이밍의 위반이 없는가가 확인된다. 타이밍검증에서는 셀 고유의 지연과 배선에 의한 지연을 고려한 지연 시뮬레이션이 행해진다. 배선지연은 레이아웃 전에는 배선이 확정되지 않기 때문에 통계적으로 구한 가상배선장을 사용한다. 논리검증과는 달리 논리설계결과를 근거로 진단데이터의 생성, 즉 테스트패턴을 생성하고 그 패턴의

고장검출율을 고장시뮬레이터에 의하여 검증하는 것도 이 단계에서 행해진다.

2) Back-end design 단계

front-end design에서 생성된 netlist를 물리적 레이아웃(physical layout)으로 만든다.

① 레이아웃 설계

front-end design에서 예상배선장에 기인한 지연시뮬레이션결과가 기대치와 일치하면 레이아웃설계를 한다. 레이아웃설계는 논리시뮬레이션에서 검증된 네트리스트를 근간으로 셀을 배치하는 배치설계와 셀 간을 배선하는 배선설계로 나누어진다. 레이아웃설계의 중점은 칩 면적을 작게 하는 것과 배선장을 짧게 하는 것인데, 칩 면적은 가격을 좌우하고, 배선장은 특성(타이밍)을 좌우한다. 게이트어레이는 선택한 마스터어레이에서 셀의 배치 및 배선이 가능해야 하며, 만약 불가능하면 마스터어레이를 변경하여야 한다. 고속동작을 위해서는 클록신호의 스큐가 작게 되도록 배선하는 것이 중요하다.

레이아웃에 의한 배선이 확정된 후 실배선이 지연시간을 사용한 최종 논리회로의 동작 검증을 행한다. 프로세스 기술의 미세화에 따라서, 배선지연이 셀의 지연보다도 상대적으로 크게 되었다. 또, 이전의 프로세스 기술에서는 무시 가능했던 배선의 저항에 의한 지연시간이 스큐의 검사에서 무시할 수 없다.

3) 검증(Verification) 단계

물리적 레이아웃의 설계 규칙 점검기(DRC: Design Rule Check)와 LVS(Layout vs. Schematic)를 수행하며 물리적 레이아웃에서 지연(delay) 정보 추출하여 게이트레벨 레이아웃(backannotation)을 수행한다.

4) 칩 제조공정

5) 테스트

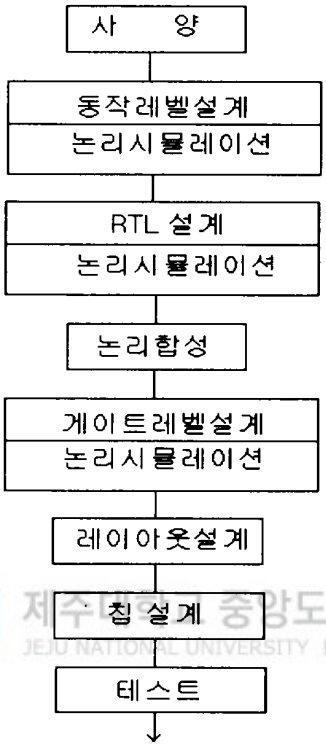


Fig. 10. Flow diagram of ASIC design

B. 양방향 연상 기억장치의 VHDL 소스 파일

1) 상태발생기 블록(bam_st)

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
USE IEEE.std_logic_unsigned.ALL;

ENTITY bam_st IS
    PORT( clr, clk                    : IN std_logic;
          en_x, en_y, en_c, en_r      : OUT std_logic);
END bam_st;
ARCHITECTURE a_bam_st OF bam_st IS
BEGIN
    PROCESS(clr,clk)
        variable cnt : std_logic_vector(6 downto 0);
    begin
        if clr='1' then
            en_x<='0';  en_y<='0';  en_r<='0';  en_c<='0';
            cnt:="0000000";
        else
            if clk'event and clk='1' then
                cnt:=cnt+"0000001";
                if cnt="0000001" then
                    en_x<='1';  en_y<='0';  en_c<='0';
                end if;
                if cnt="0100010"then
                    en_x<='0';  en_y<='1';  en_c<='0';
                end if;
                if cnt="1000011"then
                    cnt:="0000000";
                    en_x<='0';  en_y<='0'; en_r<='1   en_c<='1';
                end if;
            end if;
        end if;
```

```
    end process;
end a_bam_st;

CONFIGURATION cfg_bam_st of bam_st is
    for a_bam_st
    end for;
end cfg_bam_st;
```

※ 다음 소스 파일부터는 library구문과 configuration구문이 생략된다.

2) 입력블록(in_reg, b_input)

```
ENTITY in_reg IS
    PORT (   clr, en_c, en_r, ren_c   : IN      std_logic;
             x, out_x       : IN   std_logic_vector(7 downto 0);
             y, out_y       : IN   std_logic_vector(3 downto 0);
             in_x       : OUT   std_logic_vector(7 downto 0);
             in_y       : OUT   std_logic_vector(3 downto 0) );
END in_reg;
ARCHITECTURE a_in_reg OF in_reg IS
BEGIN
    PROCESS(clr, en_c, en_r, ren_c, x, y, out_y, out_x)
    begin
        IF clr='1' then
            in_x <= "00000000";
            in_y <= "0000";
        ELSE
            IF en_r='0' THEN
                    in_x <= x;   in_y <= y;
            elsif en_c'event and en_c='0' then
                    IF ren_c='1' THEN
                        in_x <= out_x;   in_y <= out_y;
                    END IF;
            END IF;
        END IF;
```

```
    END PROCESS;
END a_in_reg;


ENTITY b_input IS
    PORT( clr, clk, en_x, en_y    : IN  std_logic;
          in_x          : IN  std_logic_vector(7 downto 0);
          in_y          : IN  std_logic_vector(3 downto 0);
          x_t, y_t, o_x, o_y      : out std_logic );
END b_input;
ARCHITECTURE a_b_input OF b_input IS
BEGIN
    PROCESS(clr,en_x,clk,en_y,in_x,in_y)
        variable c:std_logic_vector(2 downto 0);
        variable t:std_logic_vector(1 downto 0);
    BEGIN
        if clr = '1' then
            c:="000";
            t:="00";
            x_t <= '0';  y_t <= '0';  o_x <= '0';  o_y <= '0';
        else
                if clk'event and clk='1' then
              if en_x='0' then
                    c:="000";
                    x_t<='0';  o_x<='0';
               else
                    if c="000" then
                        x_t<='0';  o_x<=in_x(7);
                    end if;
                    if c="001" then
                        x_t<='0';  o_x<=in_x(6);
                    end if;
                    if c="010" then
                        x_t<='0';  o_x<=in_x(5);
                    end if;
                    if c="011" then
```

```
                x_t<='0';   o_x<=in_x(4);
            end if;
            if c="100" then
                x_t<='0';   o_x<=in_x(3);
            end if;
            if c="101" then
                x_t<='0';   o_x<=in_x(2);
            end if;
            if c="110" then
                x_t<='0';   o_x<=in_x(1);
            end if;
            if c="111" then
                x_t<='1';   o_x<=in_x(0);
            end if;
            c:=c+"001";
        end if;
        if en_y='0' then
            t:="00";
            y_t<='0';   o_y<='0' ;
        else
            if t="00" then
                y_t<='0';   o_y<=in_y(3);
            end if;
            if t="01" then
                y_t<='0';   o_y<=in_y(2);
            end if;
            if t="10" then
                y_t<='0';   o_y<=in_y(1);
            end if;
            if t="11" then
                y_t<='1';   o_y<=in_y(0);
            end if;
            t:=t+"01";
        end if;
    end if;
end if;
```

```
    end process;
end a_b_input;
```

3) 기억블록(b_rom, b_ad)

```
ENTITY b_rom IS
    PORT( clr   : IN      std_logic;
          addr          : IN   std_logic_vector(4 downto 0);
          data      : OUT   std_logic_vector(3 downto 0 ));
END b_rom;
ARCHITECTURE a_b_rom OF b_rom IS
BEGIN
    PROCESS(clr, addr)
    begin
        if (clr='0') then
            case addr is
                when "00000" => data <="0011";
                when "00001" => data <="0001";
                when "00010" => data <="1111";
                when "00011" => data <="0001";
                when "00100" => data <="0011";
                when "00101" => data <="1111";
                when "00110" => data <="1111";
                when "00111" => data <="1111";
                when "01000" => data <="0001";
                when "01001" => data <="0011";
                when "01010" => data <="0001";
                when "01011" => data <="1111";
                when "01100" => data <="0001";
                when "01101" => data <="1101";
                when "01110" => data <="1101";
                when "01111" => data <="0001";
                when "10000" => data <="1111";
                when "10001" => data <="1101";
                when "10010" => data <="1111";
```

```
                when "10011" => data <="0001";
                when "10100" => data <="1111";
                when "10101" => data <="0011";
                when "10110" => data <="0011";
                when "10111" => data <="1111";
                when "11000" => data <="1101";
                when "11001" => data <="1111";
                when "11010" => data <="0001";
                when "11011" => data <="1111";
                when "11100" => data <="1101";
                when "11101" => data <="0001";
                when "11110" => data <="0001";
                when "11111" => data <="0001";
                when others => data <="0000";
            end case;
        end if;
    end process;
end a_b_rom;


ENTITY b_ad IS
    PORT( clr, clk, en_x, en_y    : IN  std_logic;
          addr                    : OUT std_logic_vector(4 downto 0));
END b_ad;
ARCHITECTURE a_b_ad OF b_ad IS
BEGIN
    PROCESS(clk, clr, en_x, en_y)
        VARIABLE temp_x:std_logic_vector(4 downto 0);
        VARIABLE temp_y:std_logic_vector(4 downto 0);
    BEGIN
    IF clr ='1' then
        temp_x:="00000";
        temp_y:="00000";
        addr<="00000";
    ELSE
        IF clk'event AND clk='1' then
```

```
            IF en_x='1' THEN
                addr <=temp_x;
                temp_x:=temp_x+"00001";
            ELSE
                temp_x:="00000";
            END IF;
            IF en_y='1' THEN
                addr<=temp_y;
                IF temp_y<"11000" then
                    temp_y:=temp_y+"01000";
                ELSE
                    temp_y:=temp_y-"10111";
                END IF;
            ELSE
                temp_y:="00000";
            END IF;
        end if;
    end if;
    END PROCESS;
END a_b_ad;
```


4) 연산블록(bam_alu)

```
ENTITY bam_alu IS
    PORT ( clr, clk, en_x, en_y, x_t, y_t, o_x, o_y  : IN std_logic;
           in_x : in std_logic_vector(7 downto 0);
           in_y  : in std_logic_vector(3 downto 0);
           data : IN std_logic_vector(3 downto 0);
           out_x : OUT std_logic_vector(7 downto 0);
           out_y : OUT std_logic_vector(3 downto 0) );
END bam_alu;
ARCHITECTURE a_bam_alu OF bam_alu IS
BEGIN
    PROCESS(clr, clk, en_x, en_y, x_t, y_t, o_x, o_y, data, in_x, in_y)
        variable st, mt, x_st, T_st:std_logic_vector(3 downto 0);
```

```
    variable m_c:std_logic_vector(2 downto 0);
    variable n_c:std_logic_vector(1 downto 0);
    variable y_st, V_st:std_logic_vector(7 downto 0);
begin
    if clr = '1' then
        out_x <="00000000";
        out_y <="0000";
    else
        if clk'event and clk='1' then
            if en_x='0' then
                n_c:="11";  st:="0000";  x_st:="0000";  T_st:="0000";
            else
                if o_x='1' then
                    st := st + data;
                else
                    st := st;
                end if;
                if x_t='1' then
                    if n_c="11" then
                        if st(3)='0' then
                            if st(2 downto 0)="000" then
                                x_st(3):=in_y(3);
                            else
                                x_st:="1000";
                            end if;
                        else
                            x_st:="0000";
                        end if;
                        T_st:=T_st or x_st;
                    end if;
                    if n_c="10" then
                        if st(3)='0' then
                            if st(2 downto 0)="000" then
                                x_st(2):=in_y(2);
                            else
                                x_st:="0100";
```

```
                end if;
            else
                x_st:="0000";
            end if;
            T_st:=T_st or x_st;
        end if;
        if n_c="01" then
            if st(3)='0' then
                if st(2 downto 0)="000" then
                    x_st(1):=in_y(1);
                else
                    x_st:="0010";
                end if;
            else
                x_st:="0000";
            end if;
            T_st:=T_st or x_st;
        end if;
        if n_c="00" then
            if st(3)='0' then
                if st(2 downto 0)="000" then
                    x_st(0):=in_y(0);
                else
                    x_st:="0001";
                end if;
            else
                x_st:="0000";
            end if;
            T_st:=T_st or x_st;
            out_y<=T_st;
        end if;
        st:="0000";
        n_c:=n_c-"01";
    end if;
end if;
if en_y='0' then
```

```
        m_c:="111";    mt:="0000";
        y_st:="00000000";   V_st:="00000000";
    else
        if o_y='1' then
            mt := mt + data;
        else
            mt := mt;
        end if;
        if y_t='1' then
            if m_c="111" then
                if mt(3)='0' then
                    if mt(2 downto 0)="000" then
                        y_st(7):=in_x(7);
                    else
                        y_st:="10000000";
                    end if;
                else
                    y_st:="00000000";
                end if;
                V_st:=V_st or y_st;
            end if;
            if m_c="110" then
                if mt(3)='0' then
                    if mt(2 downto 0)="000" then
                        y_st(6):=in_x(6);
                    else
                        y_st:="01000000";
                    end if;
                else
                    y_st:="00000000";
                end if;
                V_st:=V_st or y_st;
            end if;
            if m_c="101" then
                if mt(3)='0' then
                    if mt(2 downto 0)="000" then
```

```
                y_st(5):=in_x(5);
            else
                y_st:="00100000";
            end if;
        else
            y_st:="00000000";
        end if;
        V_st:=V_st or y_st;
    end if;
    if m_c="100" then
        if mt(3)='0' then
            if mt(2 downto 0)="000" then
                y_st(4):=in_x(4);
            else
                y_st:="00010000";
            end if;
        else
            y_st:="00000000";
        end if;
        V_st:=V_st or y_st;
    end if;
    if m_c="011" then
        if mt(3)='0' then
            if mt(2 downto 0)="000" then
                y_st(3):=in_x(3);
            else
                y_st:="00001000";
            end if;
        else
            y_st:="00000000";
        end if;
        V_st:=V_st or y_st;
    end if;
    if m_c="010" then
        if mt(3)='0' then
            if mt(2 downto 0)="000" then
```

```
                y_st(2):=in_x(2);
            else
                y_st:="00000100";
            end if;
        else
            y_st:="00000000";
        end if;
        V_st:=V_st or y_st;
    end if;
    if m_c="001" then
        if mt(3)='0' then
            if mt(2 downto 0)="000" then
                y_st(1):=in_x(1);
            else
                y_st:="00000010";
            end if;
        else
            y_st:="00000000";
        end if;
        V_st:=V_st or y_st;
    end if;
    if m_c="000" then
        if mt(3)='0' then
            if mt(2 downto 0)="000" then
                y_st(0):=in_x(0);
            else
                y_st:="00000001";
            end if;
        else
            y_st:="00000000";
        end if;
        V_st:=V_st or y_st;
        out_x<=V_st;
    end if;
    mt:="0000";
    m_c:=m_c-"001";
```

```
                    end if;
                end if;
            end if;
        end if;
    END PROCESS;
END a_bam_alu;
```

5) 출력블록(b_comp, out_reg)

```
ENTITY b_comp IS
    PORT( clr, en_c     : in       std_logic;
          out_x, in_x   : IN       std_logic_vector(7 downto 0);
          out_y, in_y   : IN std_logic_vector(3 downto 0);
          ren_c    : out      std_logic );
END b_comp;
ARCHITECTURE a_b_comp OF b_comp IS
BEGIN
    PROCESS(clr,en_c,in_x,in_y,out_x,out_y)
      VARIABLE u_x, tin_x  :   std_logic_vector(7 downto 0);
      VARIABLE u_y, tin_y  :   std_logic_vector(3 downto 0);

    begin
        if clr ='0' then
            tin_x:=in_x;   tin_y:=in_y;
            if en_c='1' and en_c'event  then
                u_x := tin_x xor out_x;
                u_y := tin_y xor out_y;
                IF u_x="0000000" THEN
                    IF u_y="0000" THEN
                        ren_c <='0';
                    ELSE
                        ren_c <='1';
                    END IF;
                ELSE
                    ren_c <='1';
```

```
                END IF;
            end if;
        end if;
    END PROCESS;
END a_b_comp;


ENTITY out_reg IS
    PORT (   clr, ren_c, en_c    : IN  std_logic;
             out_x          : IN  std_logic_vector(7 downto 0);
             out_y          : IN  std_logic_vector(3 downto 0);
             A        : OUT  std_logic_vector(7 downto 0);
             B        : OUT  std_logic_vector(3 downto 0));
END out_reg;
ARCHITECTURE a_out_reg OF out_reg IS
BEGIN
    process(clr, ren_c, out_x, out_y, en_c)
    begin
        if clr = '0' then
            if en_c'event  and en_c='0' then
                if ren_c ='0' then
                    A <= out_x;
                    B <= out_y;
                end if;
            end if;
        end if;
    end process;
end a_out_reg;
```

감사의 글

이 논문이 완성되기까지 끊임없는 지도와 편달을 하여주신 강민제 교수님께 존경과 감사의 마음을 드립니다. 그리고 바쁘신 가운데서도 본 눈문을 자상하게 심사해주신 이광만 교수님, 고성택 교수님, 학위과정중 배움의 길목이 되어주신 김경식 교수님, 도양회 교수님, 김경연 교수님, 논문의 실험에 도움을 주신 김성백 교수님께 깊이 감사드립니다.

대학원을 다니는 동안 도움을 아끼지 않았던 선배님들(윤성보, 오데레사, 장경훈, 김용규, 김영민, 백원석), 같이 고민하고 웃으며 지냈던 동기들(김창일, 권성숙, 고회선), 후배들과 김미정 조교선생님, 김경미 조교선생님, 전자공학과 학우들에게도 감사를 드립니다.

끝으로 저에 대한 믿음을 버리지 않으시고 늘 저의 배경이 되어주신 부모님과 동생들, 친구들에게 이 작은 논문을 바칩니다.