# Parçacık Sürü Optimizasyonu Algoritması ile Yapay Sinir Ağı Eğitiminin FPGA Üzerinde Donanımsal Gerçeklenmesi

Mehmet Ali ÇAVUŞLU, Cihan KARAKUZU, Suhap ŞAHİN

**ÖZET**

Bu çalışmada YSAnın doğasına uygun olarak paralel işlemlerle, FPGA üzerinde, YSA eğitimi için yeni bir yaklaşım sunulmuştur. Eğitim türev bilgisine ihtiyaç duymaksızın, rastgele arama algoritması olan parçacık sürü optimizasyonu (PSO) kullanılarak FPGA üzerinde gerçeklenmiştir. FPGA'de ilgili tüm parametre değerleri ve işlemler IEEE 754 kayan noktalı sayı formatında tanımlanmıştır. Önerilen yaklaşım örnek bir YSA mimarisi baz alınarak VHDL dilinde kodlanıp Altera EP2C35F672C6 FPGA'sı üzerinde gerçeklenmiştir. Elde edilen sonuçlar önerilen yaklaşımın YSA eğitimini başarı ile gerçeklediğini göstermiştir.

**Anahtar Kelimeler:** YSA eğitimi, PSO, FPGA, Kayan noktalı sayı

# Hardware Implementation of Artificial Neural Network Training Using Particle Swarm Optimization on FPGA

**ABSTRACT**

In this study, a new ANN training approximation on FPGA is presented using parallel processes according to the nature of ANN. Training is implemented on FPGA using particle swarm optimization (PSO) stochastic search algorithm which does not need any derivative information. All related parameter values and processes are defined with IEEE 754 floating point numbers format. Proposed approach has been realized on Altera EP2C35F672C6 FPGA based on a sample ANN architecture using VHDL language. Obtained results show that proposed approach has successfully achieved ANN training.

**Key Words:** ANN training, PSO, FPGA, Floating point number

## 1. GİRİŞ

Doğasında paralel bilgi akışına sahip olan YSA'lar, modelleme, kontrol, görüntü işleme ve tıbbi teşhis gibi alanlarda oldukça zor problemleri çözebilecek yeteneğe sahiptirler. Gerçek başarımlarını paralel çalışan mimariler üzerinde gösterebilirler (1). YSA'ların en çok kullanılan türü çok katmanlı algılayıcılardır ve genellikle geriye yayılım algoritması ile eğitilirler. Fakat, türev tabanlı geriye yayılım algoritması yavaş, yerel minimuma kolay yakalanma, genellemede eksik ve başlangıç ağırlıklarına duyarlı olma gibi dezavantajlara sahiptir (2).

Yerel en iyi çözümler elde eden geriye yayılım algoritmasının aksine, global arama özelliğine sahip ve kuş sürülerinin davranışlarından esinlenerek geliştirilmiş, popülasyon tabanlı stokastik optimizasyon tekniği olan PSO (3), doğrusal olmayan problemlerin çözümü için uygundur. Eberhart ve Kennedy tarafından önerilen bu yöntem, fonksiyon optimizasyonu, bulanık sistem kontrolü, yapay sinir ağı eğitimi gibi birçok alanda başarıyla uygulanmıştır (4-7).

Algoritma rastgele çözümler içeren bir popülasyonla başlatılır ve nesilleri güncelleyerek en optimum çözümü araştırır. PSO'da parçacık olarak adlandırılan olası muhtemel çözümler, o andaki optimum parçacığı izleyerek problem uzayında dolaşırlar. PSO'nun klasik optimizasyon tekniklerinden en önemli farklılığı türev bilgisine ihtiyaç duymamasıdır. Diğer evrimsel algoritmalara nazaran PSO, algoritmasının gerçeklenmesi, ayarlanması gereken parametre sayısının az olması sebebiyle, kolaydır.

Donanımsal YSA uygulamaları hız, paralellik ve yoğun işlem yükü sağlamasından dolayı, yazılımsal YSA uygulamalara göre tercih edilmektedir (8-10). Yazılımsal YSA uygulamalarının sıralı işlem yapmaları sebebiyle, hız, paralellik ve yoğun işlem yükü gerektiren uygulamalar için özel amaçlı VLSI devreler tasarlanabilir (11). Özel amaçlı tasarlanan VLSI tümleşik devreler, gerçek zamanlı YSA uygulamalarında, işlem hızını ve yoğunluğunu artırma problemlerinin üstesinden gelmesine rağmen sadece bir YSA için kullanım (sınırlı

*Makale 26.04.2010 tarihinde gelmiş,28.09.2010 tarihinde yayınlanmak üzere kabul edilmiştir.*

*M. A. ÇAVUŞLU Y-Vizyon Sinyalizasyon Tic. Ltd. Şti, Hacettepe Teknokent 3. ARGE Binası, N0:13, 06800, Beytepe ANKARA*

*e-posta : alicavuslu@gmail.com*

*C. KARAKUZU, Bilecik Üniversitesi, Mühendislik Fakültesi, Bilgisayar Müh. Böl., Gülümbe Yerleşkesi, 11210, BİLECİK*

*e-posta : cihan.karakuzu@bilecik.edu.tr*

*S.ŞAHİN, Kocaeli Üniversitesi, Mühendislik Fakültesi, Bilgisayar Müh. Böl., Umuttepe Yerleşkesi, 41380, İZMİT*

*e-posta : suhapsahin@kocaeli.edu.tr*

*Digital Object Identifier 10.2339/2010.13.2, 83-92*

kullanım) ve üretim aşamalarının maliyetli olması, çok zaman almasından dolayı yaygın kullanım alanı bulamamışlardır (12). Field Programmable Gate Array (FPGA), esnek tasarım özelliklerine sahip olması nedeniyle, VLSI sistemlere alternatif olmuştur (13).

Programlanabilir ara bloklara sahip olan FPGA'larla, tasarım esnasında değişikliklerin kolayca yapılabilmesi ile zaman ve maliyetten önemli kazanımlar sağlanır. Paralel işlem yapabilme ve veri akışı özelliklerine sahip FPGA üzerinde, ağır işlem yükü gerektiren gerçek zamanlı uygulamalar gerçeklenebilmektedir (14). Bu özelliklerinden dolayı FPGA, son yıllarda birçok YSA uygulamasında, tercih edilen gömülü sistem platform olarak ortaya çıkmıştır (15-19).

Literatürde FPGA tabanlı değişik mimarili farklı YSA gerçeklemeleri ve eğitimleri sunulmuştur (8, 10, 20-24, 26-31). Ferreira ve arkadaşları, ileri beslemeli YSA yı, 32 bit kayan noktalı sayı formatında gerçeklemişler, aktivasyon fonksiyonu için 256 lineer bölümden oluşan hiperbolik tanjant fonksiyonu yaklaşımı kullanmışlar (20). Won, 5-6-1 yapısında ileri beslemeli YSA'yı 8 bit tamsayı formatında gerçeklemiş, aktivasyon fonksiyonu olarak logaritmik sigmoidal için, 16 bit genişliğinde 1024 uzunluğunda tek portlu blok RAM kullanarak bakma tablosu oluşturmuştur (21). Ferrer ve arkadaşları, çok katmanlı algılayıcıyı FPGA üzerinde sabit noktalı sayı formatı ile gerçeklemişler, aktivasyon fonksiyonu için $2^{-7}$ ile ölçeklendirilmiş hiperbolik tanjant fonksiyonu yaklaşımı kullanılmışlar (22). Chalhoub ve arkadaşları, çok katmanlı algılayıcıyı 18 bit sabit noktalı sayı formatı ile gerçeklemişler ve aktivasyon fonksiyonunu ROM tabanlı bakma tablosu ile işletmişler (23). Nedjah ve arkadaşları, logaritmik sigmoidal fonksiyonun parabolik yaklaşımlı aktivasyonunu kullanarak 32 bit kayan noktalı sayı formatı ile YSA gerçeklemişler (8). Çavuşlu ve arkadaşları, çok katmanlı algılayıcı gerçeklenmesi için 16 bit kayan noktalı sayı formatı ve logaritmik sigmoidal fonksiyonuna çok benzeyen (25)'de verilen matematiksel yaklaşım kullandılar (24).

Stepanova ve arkadaşları (10), DNA motiflerinin yapısının belirlenmesi için Hopfield ağı FPGA üzerinde gerçeklenmişler. Bu çalışmada 32 bit sabit noktalı sayı formatı ve tanjant hiperbolik fonksiyonunun parçalı doğrusal yaklaşımı kullanılmıştır. Lázaro ve arkadaşları (26), genel regresyon sinir ağı yapısını FPGA üzerinde gerçeklemişler. Gerçeklemede 32 ve 64 bit kayan noktalı sayı, 12 ve 64 bitlik sabit noktalı sayı formatı kullanılarak karşılaştırılması yapılmıştır. Hücre aktivasyonları için ise bakma tablosu kullanılmıştır. Boubaker ve arkadaşları (27), LVQ (Learning Vector Quantization) sinir ağının FPGA'de gerçeklenmesi üzerine çalışmışlardır.

Savich, Moussa ve Areibi YSA'nın geriye yayılım algoritması ile eğitimini FPGA ile gerçeklenmişler (28). Eğitim sabit ve kayan noktalı sayı formatları ile donanımsal gerçeklenmiştir. Aktivasyon fonksiyonu olarak ise logaritmik sigmoidal fonksiyonun parçalı doğrusal yaklaşımı kullanılmıştır. Çavuşlu ve arkadaşları (29)'daki çalışmada, YSA eğitimini geriye yayılım algoritması ile kayan noktalı sayı formatında FPGA üzerinde gerçeklemişler. Aktivasyon fonksiyonu olarak logaritmik sigmoidal fonksiyonunun (25)'de verilen matematiksel yaklaşımını kullanmışlar. Avcı ve arkadaşları (30), genetik algoritma ve sinir ağının karma kullanımı ile plaka yeri bulma işlemini gerçekledikleri çalışmada, YSA yı FPGA üzerinde 32 bit sabit noktalı sayı formatı ve logaritmik sigmoidal fonksiyonun (25)'de verilen matematiksel yaklaşımı ile gerçeklemişler. Farmahini-Farahani ve arkadaşları, PSO algoritması ile YSA eğitimini FPGA'de sabit noktalı sayı formatı kullanarak gerçeklemişler, gerçekledikleri ağda tanjant hiperbolik aktivasyon fonksiyonu için bakma tablosu kullanmışlardır (31).

Yukarıdaki kısaca özetlenen çalışmaları iki gruba ayırabiliriz: FPGA üzerinde eğitilmiş parametreleri bilinen bir ağı gerçekleme (8, 20, 21, 22, 24, 26, 27, 30) ve eğitimi ile birlikte bir ağı FPGA üzerinde gerçekleme (28, 29, 31). Bu çalışma, önceki çalışmalarımızın (24, 29) devamı niteliğinde olup, FPGA üzerinde YSA'nın ve PSO algoritması kullanılarak eğitiminin gerçeklenmesi yüksek duyarlılıklı sayı formatında yeni bir yaklaşımla sunmaktadır. Bu çalışmaya -kullanılan eğitim yöntemi bakımından- en yakın olanı (31)'dir. Fakat (31)'in en belirgin dezavantajı, 4 adet işlem birimi ile hücre hesaplamalarını yapması gerekmektedir. Eğer gerçeklenecek ağ 5 ve daha çok hücre içerdiğinde, hücre işlemlerinin 4'erli sıralı bir biçimde yapılmasıdır. Bu da YSA'da paralel işlem mantığına ters düşmektedir. Ayrıca sabit noktalı sayı formatının kullanılması ve hücre activasyon fonksiyonunun bakma tablosu ile gerçeklenmesi de diğer dezavantajlardır. Bu çalışmanın en belirgin katkısı/farkı, kayan noktalı sayı formatında toplama ve çarpma modüllerinin yanı sıra bölme modülünün kullanımı, bakma tablosu ve parçalı doğrusal yaklaşımı kullanılmaksızın fonksiyonel aktivasyon gerçeklemesinin kullanılmasına ilaveten -YSA'nın doğasına uygun olarak- veri akışını mümkün olduğunca paralel yapan yeni bir yaklaşımla YSA ve PSO tabanlı eğitiminin FPGA üzerinde gerçeklemesinin yapılmasıdır.

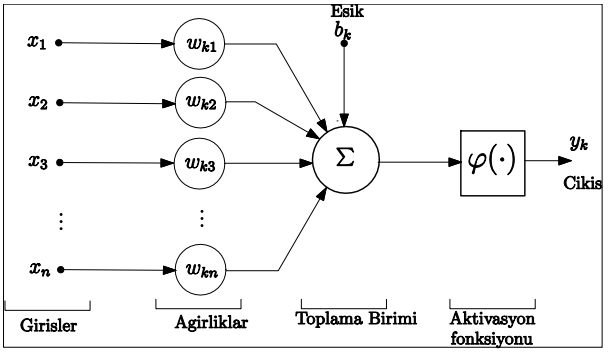
Sabit noktalı sayılar aritmetik işlemlerin donanımsal gerçeklenmesinde kolaylık ve düşük alan maliyeti sağlamalarına rağmen, kayan noktalı sayıların sağladığı hassaslık ve dinamikliği sağlayamazlar. YSA'nın eğitim aşamasında sayı duyarlılığının yüksek olması çok önemlidir. Hesaplamalarda sağladığı hassaslık ve dinamiklik özelliğinden dolayı uygulamada sayı formatı olarak IEEE 754 32 bit kayan noktalı sayı formatı kullanılmıştır. Aktivasyon fonksiyonu için ise bakma tablosu ve parçalı doğrusal yaklaşımlar yerine, logaritmik sigmoidal fonksiyon için (25)'de verilen matematiksel yaklaşım kullanılmıştır. Bu biçimde fonksiyonel aktivasyon kullanımı, fonksiyon çıkışında yüksek doğruluk elde etmek için, doğrusal parçalı yaklaşımdaki kontrol ifadelerinde sayıca artış, bakma tablosu yaklaşımlarında da çok sayıda örneklerin saklanması için gerekli olan

hafıza alanındaki ve tablodan fonksiyonun çıkış değerinin bulunması için gerekli olan süredeki artış gibi dezavantajları giderir. Bu çalışmada kullanılan yaklaşımın donanımsal gerçeklenmesinde, kontrol deyimlerine ve veri saklanması için hafızaya gerek yoktur. Uygulamanın benzetimi Modelsim'de, gerçeklenmesi ise Altera firmasına ait EP2C35F672C6 FPGA'sı üzerinde yapılmıştır.

Bu çalışmanın içeriği şu düzende aşağıda verilecektir: Bölüm 2'de YSA mimarisi, Bölüm 3'te parçacık sürücü optimizasyon algoritması, Bölüm 4'te YSA eğitiminin PSO ile FPGA tabanlı gerçeklenmesi ve 5. bölümde sonuçlar verilerek önerilen yaklaşımın başarısı gösterilmiştir.

## 2. YSA MİMARİSİ

YSA'lar, yapay sinir hücrelerinin bir araya gelmesi ile oluşmaktadır. Şekil 1'den de görüleceği üzere, YSA hücre modelinde, her yapay sinir hücresi (YSH) ağırlıklaştırılmış girişlerini toplar ve bu toplamı bir aktivasyon fonksiyonundan geçirerek çıkış verir. Her giriş ya bir başka YSA'nın çıkışıdır ya da dış dünyadan alınmış bir giriştir. Yapısal parametreler, örneğin katman sayısı, YSH sayısı ve hücre aktivasyon fonksiyonu uygulamadan uygulamaya farklılık gösterebilir.




Şekil 1. Yapay sinir hücresi modeli

Şekil 1'de verilen YSA'daki bir k. YSH çıkışı aşağıda verilen denklemlerle hesaplanır.

$$v_k = \sum_{j=1}^{m} w_{kj} x_j + b_k \tag{1}$$

$$y_k = \varphi(v_k) \tag{2}$$

YSA'nın gerçeklenmesini iki safhada inceleyebiliriz. Öğrenme aşaması, ağ parametrelerinin eğitim algoritmaları kullanarak uygun değere getirilmesinden oluşur. Test ve doğrulama aşaması, ağ eğitimi ile bulunan parametrelerle ağın test edilmesinden oluşur. Öğrenme aşamasındaki işlem ve hesaplamalar test etme aşamasındaki işlem ve hesaplamalara göre çok daha karmaşıktır. Donanımdaki ağ parametreleri öğrenme safhasında belirlenir. Öğrenme sonrası, bu parametreler, ağ eğitiminde gösterilmeyen veriler için test edilir.

## 3. PARÇACIK SÜRÜ OPTİMİZASYONU (PSO) ALGORİTMASI

Parçacık sürüsü algoritması hayvanlar arasındaki sosyal etkileşimden esinlenerek bulunan bir optimizasyon tekniğidir. PSO algoritması parçacık adı verilen bireyler arasındaki sosyal bir etkileşimi kullanarak, arama uzayında bireylerini, adaptif olarak en anlamlı bölgeye doğru yönlendirir.

PSO, bir grup rastgele çözümle (parçacık sürüsü) başlatılır ve güncellemelerle en uygun çözüm bulunmaya çalışılır. Her tekrarlamada (iterasyonda), parçacık konumları, iki en iyi parçacığa göre güncellenir. İlki; o ana kadar kullanılan aynı numaralı parçacıklar arasındaki en iyi uygunluk değerine sahip olan parçacıktır. Bu parçacık yerel en iyi ($p_{ye}$) olarak adlandırılır ve hafızada saklanmalıdır. Diğeri ise, popülasyonda o ana kadar tüm parçacıklar arasında elde edilen en iyi uygunluk değerini sağlayan parçacıktır. Bu parçacık global en iyidir ve *"*$g_{best}$*"* ile gösterilir. Örneğin D boyutlu N adet parçacık olduğunu varsayalım. Bu durumda popülasyon parçacık matrisi eşitlik 3'deki gibidir.

$$P = \begin{bmatrix} p_{11} & p_{12} & \cdots & \cdots & p_{1D} \\ p_{21} & p_{22} & \cdots & \cdots & p_{2D} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ p_{N1} & p_{N2} & \cdots & \cdots & p_{ND} \end{bmatrix}_{N\times D} \tag{3}$$

Yukarıdaki matriste, i'inci parçacık $p_i = [\, p_{i1}, p_{i2}, \ldots . p_{iD} \,]$ olarak ifade edilir. Önceki iterasyonlarda, en iyi uygunluk değerini veren i'ninci parçacığın pozisyonu $p_{ye_i} = [\, p_{ye_{i1}}, p_{ye_{i2}}, \ldots . p_{ye_{iD}} \,]$'dır ve yerel en iyi parçacık olarak adlandırılır. *gbest* ise her iterasyonda tüm parçacıklar için tektir ve $g_{best} = [\, p_{gb_1}, p_{gb_2}, \ldots . p_{gb_D} \,]$ şeklinde gösterilir. i'inci parçacığın hızı (her boyuttaki konumunun değişim miktarı) $v_i = [\, v_{i1}, v_{i2}, \ldots . v_{iD} \,]$ olarak ifade edilir. İki en iyi değerin bulunmasından sonra parçacık hızları ve konumları aşağıda verilen 4 ve 5 nolu denklemlere (32) göre güncellenir.

$$v_i^{k+1} = v_i^k + c_1 r_1^k \, (p_{ye_i}^{\,k} - p_i^k) + c_2 r_2^k \, (g_{best}^{\,k} - p_i^k) \tag{4}$$

$$p_i^{k+1} = p_i^k + v_i^{k+1} \tag{5}$$

Denklem 4'de, $c_1$ ve $c_2$ öğrenme faktörleridir. $c_1$ ve $c_2$, her parçacığı kendisinin yerel en iyisine ($p_{ye}$) ve sürünün en iyisine ($g_{best}$) doğru çeken, hızlanma terimlerini ifade eden sabitlerdir. $c_1$, parçacığın kendi tecrübelerine göre hareket etmesini, $c_2$ ise sürüdeki diğer parçacıkların tecrübelerine göre hareket etmesini sağlar. Düşük değerler seçilmesi parçacıkların hedef bölgeye doğru çekilmeden önce, bu bölgeden uzak yerlerde dolaşmalarına imkân verir. Ancak hedefe ulaşma süresi uzayabilir. Diğer yandan, yüksek değerler seçilmesi, hedefe ulaşmayı hızlandırırken, beklenmedik hareketlerin oluşmasına ve hedef bölgenin es geçilmesine sebep olabilir. Denklemdeki $r_1$ ve $r_2$, (0,1) arasında düzgün dağılımlı rasgele sayılardır. $k$ ise

iterasyon sayısını belirtmektedir. Şekil.2'de PSO algoritmasının genel kod yapısı özetlenmiştir.

```
For her parçacık için
      {başlangıç koşullamalar}
End
Do {
     For her parçacık için
          {uygunluk değerini hesapla, eğer uygunluk
          değeri, pye den daha iyi ise; bu parçacığı yeni pye
          olarak ata}
      End
     Tüm parçacıklar için bulunan pye parçacıklarının en
     iyisi uygunluk değererine sahip olanını, sürünün
     gbest'i olarak ata
       For her parçacık için
            { (4) denklemine göre parçacık hızını hesapla
            (5)denklemine göre parçacık pozisyonunu
            güncelle}
       End
    }
While {iterasyon sayısı ≤maksimum iterasyon sayısı
       veya eğitim ölçütü ≤ hedef eğitim ölçütü }
```

Şekil 2. PSO algoritmasının kod yapısı

## 4. YSA EĞİTİMİNİN PSO İLE FPGA TABANLI GERÇEKLENMESİ

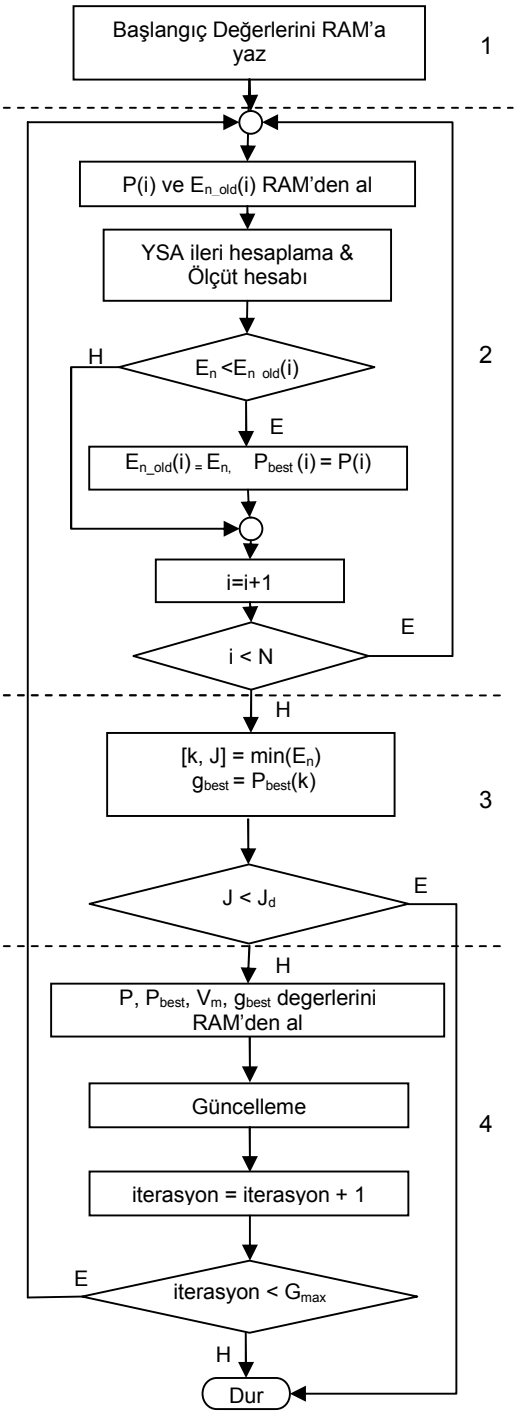
Bu çalışmada, FPGA tabanlı YSA'nın eğitiminde ağ parametrelerinin belirlenmesi işlemi PSO algoritması ile gerçeklenmiştir. Şekil.3'te de görüldüğü gibi FPGA tabanlı gerçekleme ve eğitim 4 aşamada yapılmaktadır. Birinci aşamada RAM bloklarının oluşturulup, rastgele üretilen başlangıç değerlerinin RAM'a yazılma işlemi, 2. aşamada YSA'nın FPGA tabanlı gerçeklenmesi, uygunluk değerinin hesaplanması ve yerel en iyilerin belirlenmesi, 3. aşamada global en iyi değerlerinin belirlenmesi, 4. aşamada ise güncelleme işlemleri yapılmaktadır. Kısaca bu 4 aşamalı işlemleri aşağıdaki gibi tanımlayabiliriz.

### 4.1. RAM Bloklarının ve Başlangıç Değerlerinin Oluşturulması

#### 4.1.1 RAM bloklarının oluşturulması

Uygulamada parametrelerin okuma/yazma işlemlerini paralel yapabilmek amacıyla kullanılan matris ve vektörlere ($P$, $P_{best}$, $V_m$, $g_{best}$ ve $E_n$) ait farklı blok RAM'lar oluşturulmuştur. RAM blokları, $g_{best}$ vektörü için 1 x D, $E_n$ (uygunluk değeri) vektörü için 1xN, diğer RAM blokları için NxD uzunluğundadır. N, parçacık sayısını (sürü büyüklüğü) ve D, optimize edilecek parametre sayısını gösterir. Blok RAM'larin derinlikleri kullanılacak sayı formatının bit uzunluğuna göre tasarlanmıştır.

Blok RAM'lar, Modelsim'de simülasyon yapılırken Şekil. 4'teki örnek koddan (33) görüleceği üzere, adres (ADRES_WIDTH) ve data (DATA_WIDTH) genişliklerine bağlı olarak VHDL dilinde kodlanmıştır. Veri yazma işlemi, *we* portu 1 olduğunda verinin RAM bloğunda ilgili adrese yazılması ile gerçekleşmektedir. Okuma işlemi ise sürekli olarak her adımda yapılmaktadır ve RAM bloğundaki ilgili adresteki verinin *q* portuna aktarılmasıyla gerçeklenmektedir. FPGA üzerinde blok RAM'lar, Altera Mega Wizard tools ile VHDL dilinde N, D ve sayı formatı bit uzunluğuna bağlı olarak oluşturulmuştur.




Şekil 3. PSO ile FPGA üzerinde YSA eğitimi akış şeması

#### 4.1.2 Başlangıç değerlerinin oluşturulması

PSO'daki parçacık ve hızlarının başlangıç değerleri 32 bit kayan noktalı sayı formatında (0-1) aralığında eşitlik 6 (34) ile verilen denklemle belirlenmiştir.

$$X_{n+1} = (aX_n + c) \bmod m \qquad (6)$$

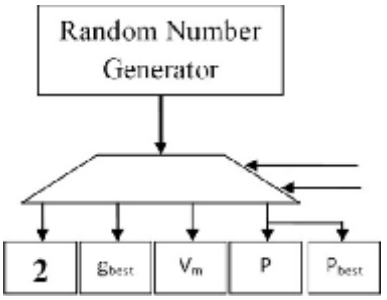
```
TYPE RAM IS ARRAY(0 TO 2 ** ADDRESS_WIDTH - 1)
OF std_logic_vector(DATA_WIDTH - 1 DOWNTO 0);
SIGNAL ram_block : RAM;

IF (we = '1') THEN
        ram_block(to_integer(unsigned(address))) <= data;
END IF;
q <= ram_block(to_integer(unsigned(address)));
```

Şekil 4. VHDL dilinde blok RAM oluşturma örnek kodu (33)

Sayı üretimi $X_0$ tohum değeri ile başlar. Eşitlikte, a ve c sabit sayılar olup m ise modüldür. Başlangıç parametrelerinin her eğitimde farklı olması için FPGA üzerinde gerçekleşecek olan eğitim, harici bir girişin set edilmesi ile başlatılmakta ve sabit değerler, harici girişin set edilmesine kadar geçen darbe sayısı ve kullanıcının belirlediği parametrelerle belirlenmektedir.

Başlangıç değerlerinin rastgele üretilip RAM'a yazılması işlemi, kontrol girişlerine bağlı olarak Şekil 5'de verildiği gibi yapılmaktadır. Sırasıyla üretilen değerler ilk olarak $P$ ve $P_{best}$ değerlerine (bu değerler başlangıçta aynı olmalıdır), daha sonra sırasıyla $V_m$ ve $g_{best}$ için oluşturulan bloklara yazılmaktadır. $P$, $P_{best}$, $V_m$, $g_{best}$ ve $E_n$ sırasıyla sürü matrisi, sürünün yerel en iyi parçacıklarını içeren matris, sürünün hız matrisi, global en iyi parçacık vektörü ve parçacıkların uygunluk değerlerini içeren ölçüt vektörünü temsil ederler. Yazma işlemi bittikten sonra 2. aşamaya geçilmektedir.



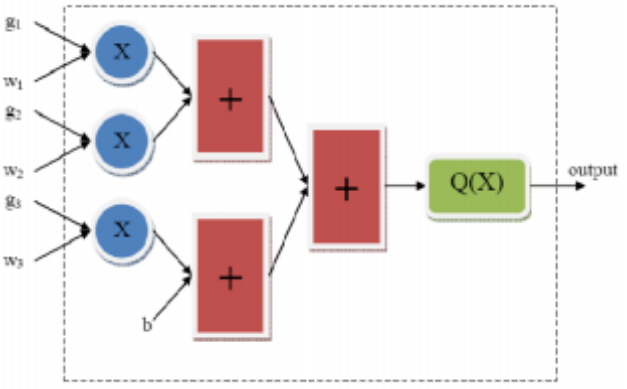

Şekil 5. Başlangıç değerlerinin RAM'e yazılmasını gösteren blok şema

### 4.2 YSA'nın Donanımsal Gerçeklenmesi ve Yerel En İyilerin Belirlenmesi

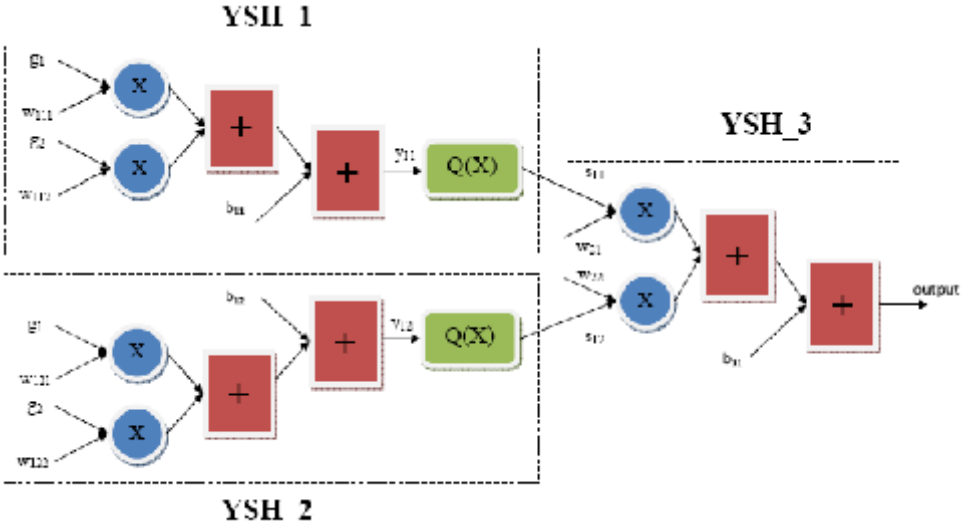
Parçacık elemanları RAM'dan okunduktan sonra YSA parametreleri olarak atanır ve bu parametrelerle her giriş için YSA çıkışındaki hata bulunur. Bu hatalar kullanılarak her iterasyonda parçacığın uygunluk değeri hesaplanır. Her parçacık için elde edilen uygunluk değerleri kendine ait eski değerden daha iyi olması durumunda ilgili parçacık yerel en iyi olarak atanır. Bu işlemler tüm parçacıklar için yapılmakta ve daha sonra 3. aşamaya geçilmektedir.

#### 4.2.1 YSA'nın donanımsal gerçeklenmesi

YSA'nin donanımsal gerçeklenmesinde aritmetik işlemler FPGA üzerinde paralel olarak koşturulur. Şekil.6'da 3 girişli bir YSH'ya ait blok şema verilmektedir. Şekilde $g_i$ YSH'nin girişlerini, $w_i$ girişlere ait ağırlıkları, b hücreye ait eşik değerini, Q(x) ise aktivasyon fonksiyonunu göstermektedir. Şekil 7'de ise 2 giriş, gizli katmanında 2 hücre ve çıkış katmanında 1 hücreden oluşan YSA'ya ait blok yapı gösterilmektedir. Şekilden de görüldüğü üzere 3 YSH kullanılmıştır. YSH_3'te doğrusal aktivasyon fonksiyonu kullanılmıştır.




Şekil 6. YSH için paralel veri işleme blok şeması




Şekil 7. {2-2-1} yapısındaki YSAnın FPGA'de gerçekleme blok yapıs

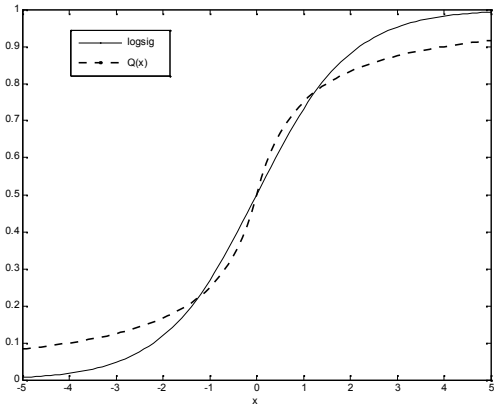
YSH için önemli birimlerden birisi de aktivasyon fonksiyonudur. Aktivasyon fonksiyonu, hücreye ait girişler ve ağırlıkların çarpımı ile eşik değerinin toplamını işleyerek çıkışı belirler. Üstel fonksiyon ifadesini FPGA'de doğrudan gerçekleyemediğimizden, eşitlik 8 ile verilen matematiksel yaklaşım (25) kullanılmıştır. Şekil 8'de de görüldüğü gibi kullandığımız fonksiyon logaritmik sigmoidal fonksiyonuna oldukça benzeyen bir fonksiyondur.

$$Q(x) = \frac{1}{2}\left[1 + \frac{x}{1+|x|}\right] \tag{8}$$

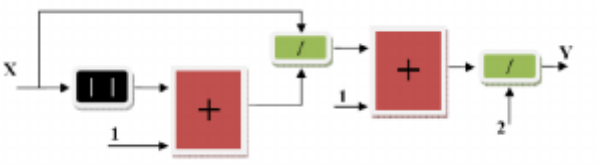
Şekil 9'da ise aktivasyon fonksiyonun FPGA'da gerçeklenme blok şeması verilmiştir. Şemadan görüleceği üzere aktivasyon fonksiyonu için 2 toplama, 2 bölme ve bir mutlak değer alma işlem birimleri gerekmektedir. Kayan noktalı sayılarda mutlak değer alma işlemi verilen sayının işaret bitinin sıfır yapılmasıyla, 2'ye bölme işlemi de exponent ifadesi 1 azaltılmasıyla (*e-1*) yapılır. Bu şekilde aktivasyon fonksiyonumuz 2 toplama ve 1 bölme işlem birimi ile gerçeklenebilir.

Kayan noktalı sayılar, sıradan bir uygulama için bile aşırı donanım kaynağı tüketmesine rağmen, sayı gösterimindeki dinamiklik ve hassas işlem yapma kapasitesi sebebiyle bu çalışmada tercih edilmiştir. Şekil 10'da kayan noktalı sayı formatına ait blok şema verilmiştir. Eşitlik 9'da bir sayının kayan noktalı sayı formatına dönüştürülmesi için gerekli formül verilmiştir.

Kayan noktalı sayılar, sıradan bir uygulama için bile aşırı donanım kaynağı tüketmesine rağmen, sayı gösterimindeki dinamiklik ve hassas işlem yapma kapasitesi sebebiyle bu çalışmada tercih edilmiştir. Şekil 10'da kayan noktalı sayı formatının blok yapısı verilmiştir. Eşitlik 9'da bir sayının kayan noktalı sayı formatına dönüştürülmesi için gerekli formül verilmiştir.




Şekil 8 Aktivasyon fonsiyonu




Şekil 9. Aktivasyon fonksiyonu gerçekleme blok yapısı

FPGA üzerinde aritmetik işlemleri yapabilmek için VHDL dilinde kodlanan ve daha önceki çalışmalarımızda oluşturulan (24,29,35,36) add.lib, mul.lib, div.lib kütüphaneleri kullanılmıştır. Bu kütüphanelere ait sentez sonuçları Tablo 1'de verilmiştir. Sentez sonuçları 32 bit (1-8-23) kayan noktalı sayı formatında Altera firmasına ait EP2C35F672C6 entegresi üzerinde, Quartus II 8.0 programından alınmıştır.


Şekil 10. IEEE 754 32 bit kayan sayı formatı

$$Number = (-1)^s (1.f) 2^{e-bias} \tag{9}$$

Tablo.1 Kullanılan aritmetik işlemler için gerekli lojik eleman sayıları

| | Toplama | Çarpma | Bölme |
|---|---|---|---|
| LEs | 758 | 108 | 329 |

### 4.2.2 Uygunluk değerlerinin hesaplanması ve yerel en iyilerin belirlenmesi

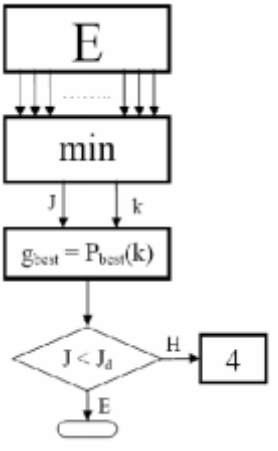
Uygunluk değerinin hesaplanması için eşitlik 10'da verilen toplam karesel hata ölçütü kullanılmıştır. Eşitlik 10'daki hata ($e_j$) eşitlik 11 ile tanımlanır. j örnek sayısını, $yd_j$ istenen çıkış değerini göstermektedir. Elde edilen $E_n$ değeri daha önceki elde edilen $E_{n\_old}$ değeri ile karşılaştırılır. Eğer yeni değer eski değerden küçük ise o parçacık $P_{best}$'deki yerine atanmaktadır. Aynı şekilde $E_n$ değeri de $E_{n\_old}$ değerine atanır. Tüm parçacıklar için aynı işlemler tekrarlanır.

$$E_n = \frac{1}{2}\sum_j e_j^2 \tag{10}$$

$$e_j = yd_j - y_j \tag{11}$$

### 4.3 Küresel En İyilerin Belirlenmesi

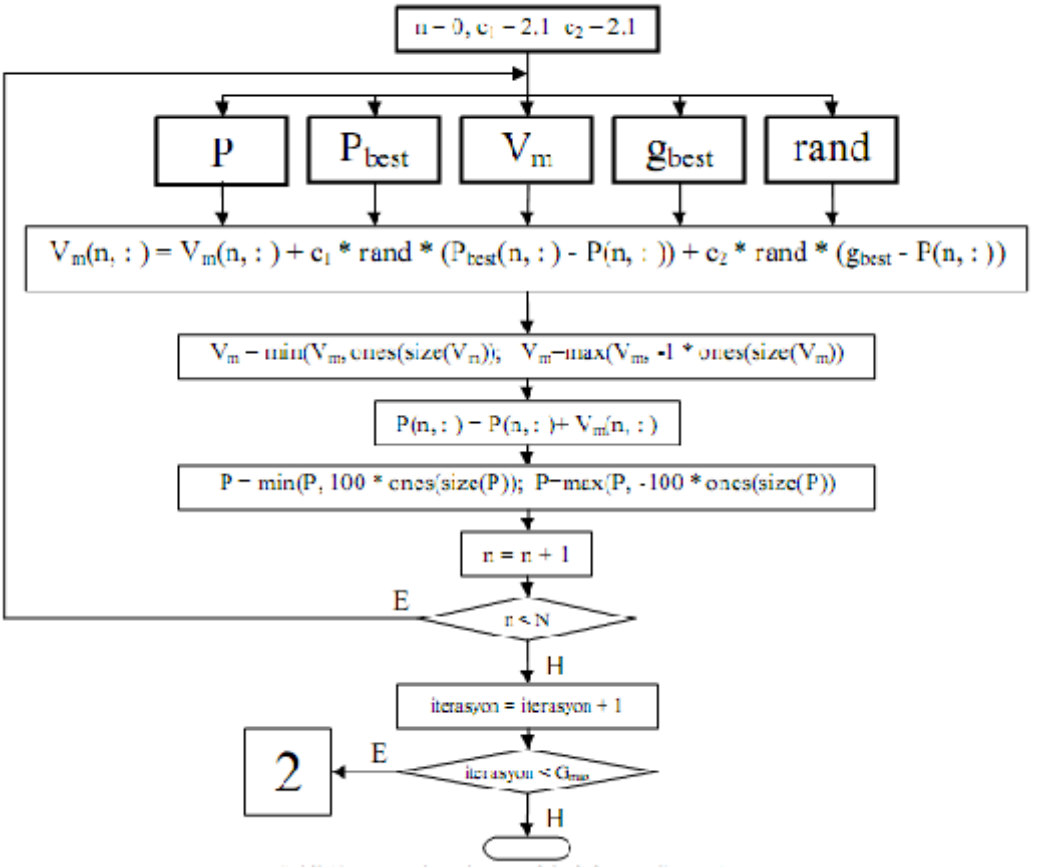
Bu bölümde uygunluk değerleri arasından minimum olanının indeksinin işaret ettiği $P_{best}$'in ilgili satır vektörü küresel en iyi ($g_{best}$) olarak atanır. Eğer minimum ölçüt değeri sonlandırma kriteri ($J_d$) değerinin altında ise eğitim sonlandırılır, aksi takdirde 4. aşamaya geçilir (Şekil 11).




Şekil 11. Küresel en iyi parçacığı bulma akış şeması

### 4.4 Güncelleme

Güncelleme işleminde RAM'den aynı anda alınan P, $P_{best}$, $V_m$ ve $g_{best}$ değerleri eşitlik 4 ve 5'te verilen işlemlere tabi tutularak $V_m$ ve P matrislerinin güncelleme işlemleri yapılır. $c_1$ ve $c_2$ değerleri sabit değerler olup 2.1 olarak alınmıştır. Güncelleme hesaplamaları yapılırken hız değerleri [-1 1] aralığında, parçacıkların eleman değerleri ise [-100 100] aralığında sınırlandırılmıştır. P ve $V_m$ parametre güncellemeleri yapıldıktan sonra tekrar aynı anda RAM'e yazılmaktadır. Tüm parçacıklara ait parametrelerin güncelleme işlemi yapıldıktan sonra iterasyon sayısı bir artırılır ve 2 nolu işlem bloğuna geri dönülür. Eğer iterasyon sayısı maksimum nesil sayısı ($G_{max}$) değerinden fazla olursa işlem sonlandırılır. Şekil 12 güncelleme işlemlerinin akışını göstermektedir.




Şekil 12. Parçacık ve hız matrislerini güncelleme akış şeması

## 5. SONUÇLAR

FPGA üzerinde yapılan eğitimde, öncelikle parçacık ve parametre sayısının belirlenmesi gerekmektedir. λ bitlik hafıza için, FPGA üzerinde $\alpha$ girişli, gizli katmanda $\beta$ hücreye sahip ve çıkış katmanında ise θ hücreye sahip bir YSA'nın γ bit uzunluğunda veri gösterimi için kullanılabilecek maksimum parçacık sayısı (ξ) eşitlik 12'de verilmiştir. *D* parçacık elemanlarının sayısını göstermektedir.

$$\left.\begin{aligned} D &= \alpha.\beta + \beta\theta + \beta + \theta \\ \xi &= \frac{\lambda - D\gamma}{\gamma(3D+1)} \end{aligned}\right\} \qquad (12)$$

Çalışmada PSO ile YSA eğitimin FPGA üzerinde test edilme işleminde EXOR problemi baz alınmıştır. Problem çözümü için 2 girişli, gizli katmanında 2 hücresi ve çıkış katmanında 1 hücresi olan YSA kullanılmıştır. YSAnın parametre sayısı (ağırlıklar ve eşikler) 9'dur. Belirtilen yapıda bir ağ ve kullanılan FPGA için eşitlik 12'den hesapla en fazla 539 parçacık kullanılabilir. Üzerinde çalışılan problemin parametre sayısından çok fazla sayıda parçacık kullanımının eğitim başarımını çok fazla etkilemediği göz önüne alınarak, bu çalışmada PSO'daki sürü büyüklüğü (N) 30 olarak belirlenmiştir. Parçacık sayısının belirlenmesinden sonra verilerin saklanması için gerekli hafıza Tablo 2'de gösterilmiştir.

Tablo 2. PSO parametre matris ve vektörlerin saklanması için gerekli hafıza

| | Bit |
|---|---|
| ***P*** | 8640 |
| $P_{best}$ | 8640 |
| $V_m$ | 8640 |
| $g_{best}$ | 288 |
| $E_n$ | 960 |
| ***Kullanılan*** | 27168 |
| ***Toplam*** | 483840 |
| ***Yüzde*** | 5.6 |

YSA'nın gerçeklenmesi, uygunluk değerinin ve parametrelerin hesaplanması için kullanılan toplama, çarpma ve bölme modülleri sayısı Tablo 3'te gösterilmiştir. Tabloda da görüleceği üzere 17 toplama, 11 çarpma ve 2 bölme modülü kullanılmıştır.

Tablo 3. Kullanılan aritmetik işlem modül sayısı

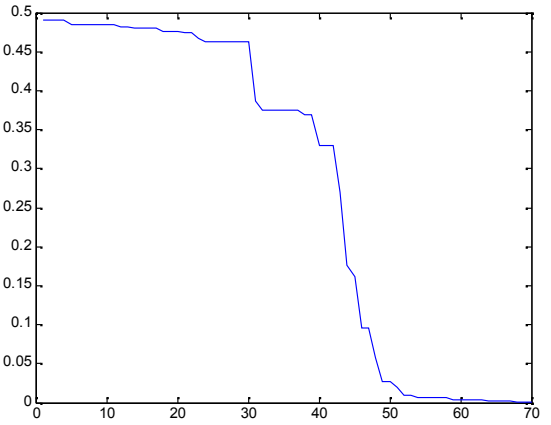
| | Sum | Mul | Div |
|---|---|---|---|
| ***MLP*** | 10 | 6 | 2 |
| ***Uygunluk Değeri*** | 2 | 1 | - |
| ***Güncelleme*** | 5 | 4 | - |
| ***Toplam*** | 17 | 11 | 2 |

FPGA üzerinde PSO ile yapılan eğitimde, 32 bit kayan noktalı sayı formatı kullanıldı. Eğitim Altera EP2C35F672C5 FPGA'sı üzerinde gerçeklendi. Sentez sonuçları Quartus II 8.0 programı kullanılarak elde edildi (Tablo 4). Sonlandırma kriteri olarak ($J_d$) ise $1.10^{-3}$ seçilmiştir. Bu değer başlangıç değerlerine bağlı olarak genellikle 50-90 jenerasyonda elde edilmektedir. Tablo 5'te 70. nesilde sonlandırma değerinin altına düşen bir eğitime ait değerler gösterilmiştir. Şekil 13'de eğitim boyunca ölçüt fonksiyonunun değişimi verilmiştir.

Sonuç olarak bu çalışmada paralel çalışan bir YSA'nın ve PSO tabanlı eğitiminin FPGA üzerinde gerçeklenmesi detaylı olarak gösterilmiş ve EXOR problemi baz alınarak önerilen yapı ile elde edilen sonuçlar verilmiştir.

Tablo 4. Önerilen yapı için kullanılan lojik eleman sayısı

| | Kullanılan | Toplam | Yüzde |
|---|---|---|---|
| ***LEs*** | 23952 | 33216 | %72 |




Şekil 13. Tablo 5'te verilen eğitim boyunca ölçüt fonksiyonu değişim grafiği

Tablo 5. PSO ile eğitimi gerçeklenen YSA'ya ait bazı Epoch (Ep)lara ait ölçüt ($E_n$) değerleri

| Ep | $E_n$ | Ep | $E_n$ |
|---|---|---|---|
| 1 | 0,49122757 | 53 | 0,010081206 |
| 5 | 0,48509708 | 54 | 0,007147521 |
| 10 | 0,48509708 | 57 | 0,007147521 |
| 15 | 0,48092192 | 59 | 0,003416425 |
| 20 | 0,47537833 | 61 | 0,003416425 |
| 25 | 0,4631243 | 63 | 0,003416425 |
| 30 | 0,4631243 | 64 | 0,001513072 |
| 35 | 0,37521678 | 67 | 0,001513072 |
| 40 | 0,3291635 | 68 | 0.0013873889 |
| 43 | 0,27027592 | 69 | 0.0013873889 |
| 45 | 0,16215982 | 70 | 9,96E-04 |
| 47 | 0,09637376 | | |
| 48 | 0,05789694 | | |
| 50 | 0,026483823 | | |
| 51 | 0,019987877 | | |

## 6. KAYNAKLAR

1. Szabó, T., Fehér, B., Horváth, G., Neural network implementation using distributed arithmetic, in Proceedings of the International Conference on Knowledge-Based Electronic Systems , Adelaide, Australia, vol. 3, Pages. 511-520, 1998.
2. Chau, K. W., A split-step PSO algorithm in predicting construction litigation outcome, Lecture Notes in Artificial Intelligence, Vol. 4099, Pages.1211-1215, 2006.
3. Kennedy, J., Eberhart, R. C., Particle swarm optimization, Proc. IEEE int'l conf. on neural networks,

Vol. IV, IEEE service center, Piscataway, NJ, Pages. 1942-1948, 1995.
4. Guerra, F.A., Coelho, L. dos S., Multi-step ahead nonlinear identification of Lorenz's chaotic system using radial basis neural network with learning by clustering and particle swarm optimization, Chaos, Solitons & Fractals, Volume 35, Issue 5, Pages 967-979, March 2008.
5. Zhou, J., Duan, Z., Li, Y., Deng, J.,Yu, D., PSO-based neural network optimization and its utilization in a boring machine, Journal of Materials Processing Technology, Volume 178, Issues 1-3, 14 Pages 19-23, September 2006.
6. Bocaniala, C.D., Costa, J.S., Application of a novel fuzzy classifier to fault detection and isolation of the DAMADICS benchmark problem, Control Engineering Practice, Volume 14, Issue 6, Pages 653-669, June 2006,
7. Ghoshal, S.P., Optimizations of PID gains by particle swarm optimizations in fuzzy based automatic generation control, Electric Power Systems Research, Volume 72, Issue 3, Pages 203-212,15 December 2004.
8. Nedjah, N., Silva, R.M.D., Mourelle, L.M.M., Silva, M.V.C.D., Dynamic MAC-based architecture of artificial neural networks suitable for hardware implementation on FPGAs, Neurocomputing, Volume 72, Issues 10-12, Pages.2171-2179, 2009.
9. Atencia, M, Boumeridja, H., Joya, G., García-Lagos, F., Sandoval, F., FPGA implementation of a systems identification module based upon Hopfield Networks, Neurocomputing, Volume 70, Issues 16-18, Pages. 2828-2835, 2009.
10. Stepanova, M., Lin, F., Lin, V.C.L., A Hopfield Neural Classifier and Its FPGA Implementation for Identification of Symmetrically Structured DNA Motifs, Journal of VLSI Signal Processing Systems, Volume 48, Issue 3, Pages 239-254, September 2007.
11. Rucket, U, Funke, A., Pintake, C., Acceleratorboard for Neural Associative Memories, Neurocomputing, Volume 5, Issue 1, Pages 39-49, 1993.
12. Cox, C., Blanz, W., GABGLION- a Fast Field Programmable Gate Array Implementation of a Connectionist Classifier, IEEE Journal of Solid-satate Circuits, Volume 27, Issue 3, Pages 288-299, 1992.
13. Morgan, P. , Ferguson, A., Bolouri, H., Cost-performance analysis of FPGA, VLSI and WSI implementations of a RAM-based neural network, Proceedings of the Fourth International Conference on Microelectronics for Neural Networks and Fuzzy Systems, Turin, Italy, Pages. 235-243, Sep. 26-28, 1994.
14. Martínez, J.J., Toledo, F.J., Fernández, E., Ferrández, J.M., A retinomorphic architecture based on discrete-time cellular neural networks using reconfigurable computing, Neurocomputing, Volume 71, Issues 4-6, Pages 766-775, January 2008.
15. Krips, M., Lammert, T., Kummert, A., FPGA implementation of a neural network for a real-time handtracking system, Proceedings of The First IEEE International Workshop on Electronic Design, Test and Applications, Pages 313 – 317, 2002.
16. Ossoinig, H., Reisinger, E., Steger, C., Weiss, R., Design and FPGA-Implementation of a Neural Networkwork, Proceedings of the 7th International Conference on Signal Processing Applications & Technology, Boston, USA, Pages. 939-943, 1996.
17. Sahin, S., Becerikli, Y., Yazici, S., Neural Network Implementation in Hardware Using FPGAs, Lecture Notes in Computer Science, Vlome 4234, Pages 1105-112, 2006.
18. Zhu, J., Milne, G.Jç, Gunther, B.K., Towards an FPGA Based Reconfigurable Computing Environment for Neural Network Implementations, Proceedings of the Ninth International Conference on Artificial Neural Networks (ICANN'99) IEE Conference Proceedings 470,Edinburgh, UK , Pages.661-666, 1999.
19. Mousa,M., Areibi, S., Nichols, K., On the Arithmetic Precision for Implementing Back-Propagation Networks on FPGA: A Case Study, in FPGA Implementations of Neural Networks, eds. Amos R. Omondi and Jagath C. Rajapakse,Springer, US, Pages 37-61, 2006.
20. Ferreira, P., Ribeiro, P., Antunes, A., Dias, F.M. A high bit resolution FPGA implementation of a FNN with a new algorithm for the activation function, Neurocomputing, Volume 71, Issue (1-3), Pages. 71-77, 2006.
21. Won, E., A hardware implementation of articial neural nertworks using field programmable gate arrays, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment Volume 581, Issue 3, Pages 816-820, 1 November 2007.
22. Ferrer, D., Gonzalez, R., Fleitas, R., Acle, J.P., Canetti, R., NeuroFPGA - Implementing Artificial Neural Networks on Programmable Logic Devices, Proceedings of Design, Automation and Test in Europe Conference and Exhibition, Volume 3, Pages.218-223, 2004.
23. Chalhoub, N., Muller,F., Auguin, M., FPGA-based generic neural network architecture, Industrial Embedded Systems, International Symposium on; Antibes Juan-Les-Pins,France, Pages:1-4, 18-20 Oct. 2006.
24. Çavuşlu, M.A., Karakuzu, C., Şahin, S., Neural Networkwork Hardware Implementation Using FPGA, in ISEECE 2006 3rd International Symposium on Electrical, Electronic and Computer Engineering Symposium Proceedings, Nicosia, TRNC,, Pages. 287-290, 2006.
25. Elliot, D. L., A Better Activation Function for Artificial Neural Networks, Technical Research Report T.R. 93-8, Institute for Systems Research, University of Maryland, 1993.
26. Lázaro, J., Arias, J., Astarloa, A., Bidarte, U., Zuloaga, A., Hardware architecture for a general regression neural network coprocessor, Neurocomputing, Volume 71, Issues 1-3, Pages 78-87, December 2007.
27. Boubaker, M., Akil, M. Khalifa, K.B., Grandpierre,T., Bedoui, M., Implementation of an LVQ neural network with a variable size: algorithmic specification, architectural exploration and optimized implementation on FPGA devices, Neural Computing & Applications, Volume 19, Number 2, Pages: 283-297, September 15, 2009
28. Savich, A.W., Moussa, M., Areibi,S., The Impact of Arithmetic Representation on Implementing MLP-BP on FPGAs: A Study, IEEE Transactions on Neural Networks, Volume 18, Issue 1, Pages:240 – 252, 2007.
29. Çavuşlu, M. A. , Karakuzu, C., Şahin, S., Yakut, M., Neural Network Training Based on FPGA with Floating Point Number Format and It's Performance, Neural Computing & Applications , DOI 10.1007/s00521-010-0423-3.

30. Avcı, G. , Kösten, M.M. , Altun, H. , Karakaya, F., Çavuşlu, M. A, Implementation of an Hybrid Approach on FPGA for License Plate Detection Using Genetic Algorithm and Neural Networks, International Symposium on INnovations in Intelligent SysTems and Applications, Trabzon, Pages. 392-396, June 29-July 1, 2009.
31. Farmahini-Farahani, A., Fakhraie, S. M., Safari, S., Scalable Architecture for on-Chip Neural Network Training using Swarm Intelligence, Proc. of the Design, Automation and Test in Europe Conf. (DATE'08), Munich, Germany, Pages.1340-1345, 2008.
32. Shi Y., Eberhart R. A Modified Particle Swarm Optimizer, Proceedings of the 1998 IEEE International Conference on Evolutionary Computation, Pages 69-73, 1998.
33. http://www.altera.com/support/examples/vhdl/vhd-single-clock-syncram.html (erişim 13 Eyl 2010)
34. Brysbaert, M., Algorithms for randomness in the behavioral sciences: A tutorial. Behavior Research Methods, Instruments, & Computers, Volume 23, Pages 45-60,1991.
35. Az, I., Şahin, S., Karakuzu, C. , Çavuşlu, M. A., Implementatıon of FFT and IFFT Algorithms in FPGA, in ISEECE 2006 3rd International Symposium on Electrical, Electronic and Computer Engineering Symposium Proceedings, Nicosia, TRNC, Pages. 7 – 10 ,2006.
36. Çavuslu, M. A., Dikmese, S., Şahin, S., Küçük K. ve Kavak, A., Akıllı Anten Algoritmalarının IEEE 754 Kayan Sayı Formatı ile FPGA Tabanlı Gerçeklenmesi ve Performans Analizi, in Proc. URSI-TÜRKİYE' 2006 3. Bilimsel Kongresi, Ankara, Turkey, Pages. 610-612. 2006.