ÖNSÖZ

Çalışmamın başından sonuna kadar yardımlarını esirgemeyen değerli danışman hocam **Öğr. Gör. Deniz BALKAN**'a, konunun teorik kısmında baştan sona destek olan **Doç. Dr. Neslihan Şengör**'e ve uygulama kısmındaki katkılarından dolayı **Yük. Müh. Levent AKSOY**'a teşekkür ederim.

Tez süreci boyunca maddi manevi her türlü desteği veren ve bu çalışmamda bana çok yardımcı olan PAVO Tasarım A.Ş.'den başta **Yük. Müh. Kutsal ANIL** olmak üzere **Müh. Servet AYOK** ve **Müh. Ilgaz AZ**'a teşekkürü bir borç bilirim.

Bu günlere gelmemde en büyük pay sahibi olan aileme çok teşekkür ederim.

Mayıs 2009 Özgür ÖZDEN

İÇİNDEKİLER

Sayfa

ŞEKİL LİSTESİ

Sayfa

ÖZET

Çok katmanlı algılayıcı yapısının FPGA ile gerçeklenmesi konulu bu çalışmada, öncelikle bir yapay sinir ağı yapısı olan çok katmanlı algılayıcılar ile ilgili genel bilgi verilmiştir. Sonrasında ağın oluşum basamakları, sinir hücrelerinin oluşturduğu giriş, çıkış ve gizli katmanların konumları ile bu katmanlar arasındaki mantıksal ve matematiksel ilişkiler ve işlemler; örnek bir çok katmanlı algılayıcı yapısı ayrıntılı olarak gösterilerek incelenmiştir. Bu aşamada uygun bir eğitim fonksiyonu seçilmiş, bu fonksiyon sonucunda ulaşılan değerler ve oluşan hatalara göre ağın eğitimi yapılmıştır.

Sonrasında, oluşturulmuş çok katmanlı algılayıcı yapısını benzetim ortamında gerçekleme ve test etme amacıyla, örnek bir problem belirlenmiştir. 9 piksel boyutundan oluşan bir gösterim ortamında T ve L harflerinin çok katmanlı algılayıcı yapısının eğitimi yoluyla birbirinden ayırt edilmesi olarak seçilen problem uyarınca, MATLAB ortamında ağın eğitimi ile benzetim, hata hesaplamaları ve testler yapılmıştır.

Bir sonraki aşamada, MATLAB ile eğitimi ve benzetimi yapılan çok katmanlı algılayıcı yapısının üzerinde gerçekleneceği FPGA yongasının yapısı, çalışması ve uygulamaları hakkında genel bilgi verilmiştir.

Sonrasında, çok katmanlı algılayıcı yapısının FPGA yongasında çalışması için gerekli olan VHDL kodunun tasarlanmasına geçilmiştir. Bu aşamada takip edilecek yöntemler, sebepleri ve sonuçları ile açıklanmıştır. Takip eden süreçte çalıştırılan VHDL kodunun benzetim ortamında elde edilen sonuçları gösterilmiş ve yorumlanmıştır.

Son aşamada, amaca uygun biçimde çalışan VHDL kodu FPGA yongasına gömülerek çok katmanlı algılayıcı örneği gerçeklenmiştir.

ABSTRACT

In this thesis, primarily, the general information about a multilayer perceptron which is a neural network algorithm is given. Then, the organization of the network, input, output and hidden layers that are made up of neurons and the logical relations and mathematical equations between these layers are analyzed through an example of multilayer perceptron network. In this stage, the network is educated by the values and errors that are reached by proper learning function chosen before.

Later, a sample problem is chosen to simulate and test the multilayer perceptron system. The problem is to differentiate the letters T and L on 9 pixel area by educating the multilayer perceptron network. The education of the network, simulation, tests and error calculation are made on MATLAB.

In the next stage, the general information about the construction, usage and applications of FPGA are given, which the multilayer perceptron system that is simulated and tested by MATLAB works on it.

Then, the VHDL code is designed, which is needed to implement multilayer perceptron algorithm on FPGA. The methods in this stage are explained with their reasons and results. Then the results of the VHDL code are shown and discussed by simulating it.

Finally, the VHDL code which is working smoothly and convenient for the aim, is embedded in the FPGA.

1. GİRİŞ

Yapay sinir ağları (YSA), insan beyninin bilgi işleme teknolojisinden esinlenerek geliştirilmiş bir bilgiişlem teknolojisidir. YSA ile basit biyolojik sinir sisteminin çalışma şekli simüle edilir (benzetilir). Simüle edilen sinir hücreleri nöronlar içerirler ve bu nöronlar çeşitli şekillerde birbirlerine bağlanarak ağı oluştururlar. Bu ağlar öğrenme, hafızaya alma ve veriler arasındaki ilişkiyi ortaya çıkarma kapasitesine sahiptirler. Diğer bir ifadeyle, YSA'lar, normalde bir insanın düşünme ve gözlemlemeye yönelik doğal yeteneklerini gerektiren problemlere çözüm üretmektedir. Bir insanın, düşünme ve gözlemleme yeteneklerini gerektiren problemlere yönelik çözümler üretebilmesinin temel sebebi ise insan beyninin ve dolayısıyla insanın sahip olduğu yaşayarak veya deneyerek öğrenme yeteneğidir.

Biyolojik sistemlerde öğrenme, nöronlar arasındaki sinaptik bağlantıların ayarlanması ile olur. Yani, insanlar doğumlarından itibaren bir yaşayarak öğrenme süreci içerisine girerler. Bu süreç içinde beyin sürekli bir gelişme göstermektedir. Yaşayıp tecrübe ettikçe sinaptik bağlantılar ayarlanır ve hatta yeni bağlantılar oluşur. Bu sayede öğrenme gerçekleşir. Bu durum YSA için de geçerlidir. Öğrenme, eğitme yoluyla örnekler kullanarak olur; başka bir deyişle, gerçekleşme girdi/çıktı verilerinin işlenmesiyle, yani eğitme algoritmasının bu verileri kullanarak bağlantı ağırlıklarını bir yakınsama sağlanana kadar, tekrar tekrar ayarlamasıyla olur.

YSA'lar, ağırlıklandırılmış şekilde birbirlerine bağlanmış birçok işlem biriminden (nöronlar) oluşan matematiksel sistemlerdir. Bir işlem birimi, aslında sık sık transfer fonksiyonu olarak anılan bir denklemdir. Bu işlem birimi, diğer nöronlardan sinyalleri alır; bunları birleştirir, dönüştürür ve sayısal bir sonuç ortaya çıkartır. Genelde, işlem birimleri kabaca gerçek nöronlara karşılık gelirler ve bir ağ içinde birbirlerine bağlanırlar; bu yapı da sinir ağlarını oluşturmaktadır [1].

Bu çalışmada, ilk olarak bir yapay sinir ağı yapısı olan çok katmanlı algılayıcı yapısına odaklanılmıştır. Çok katmanlı algılayıcı yapısının temel taşları nöronlar ve bunların oluşturdukları katman yapıları incelenmiştir. Katmanlar arasındaki ağırlık

hesaplamaları ve bunlar sonucunda oluşan çıkış değerleri ayrıntılı bir biçimde irdelenmiştir. Çıkış değerleri ile öğrenilmesi istenen değerlerin karşılaştırılması sonucu oluşturulan öğrenme algoritması incelenmiş ve sonuç olarak bir çok katmanlı algılayıcı genel yapısı ve ağ eğitimi ortaya konmuştur.

Çalışmanın ikinci aşamasında, ilk aşamada ayrıntılı bir biçimde açıklanmış olan çok katmanlı algılayıcı yapısı, bir örnek problem üzerinde benzetim yoluyla gerçeklenmiştir. 9 piksel boyutlarındaki T ve L harflerinin ayırt edilmesi olarak belirlenmiş örnek, MATLAB ortamında oluşturulmuş ve benzetilmiştir. Benzetim ile elde edilen sonuçlar farklı test verileri kullanılarak test edilmiş, sonuçta ağın doğru eğitilip eğitilmediği yorumlarla açıklanmıştır.

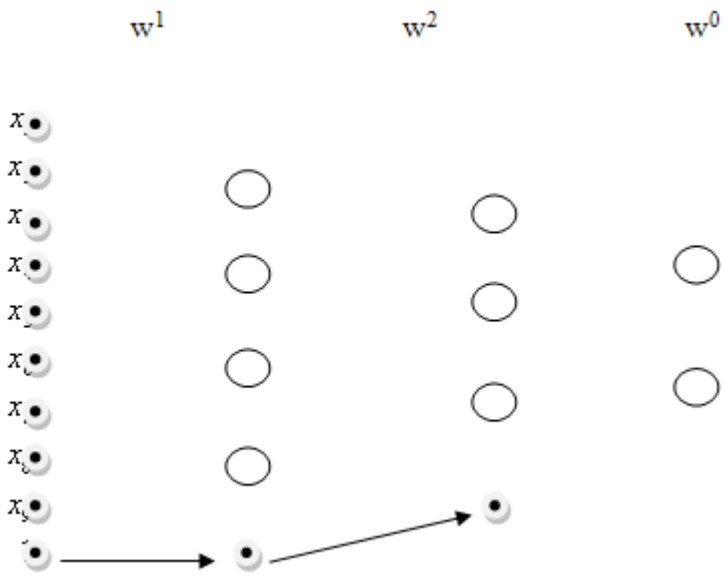
Çalışmanın üçüncü aşamasında, çok katmanlı algılayıcı yapısının üzerinde gerçekleneceği FPGA yapısı hakkında genel bilgiler verilmiştir. Bu çalışmada FPGA kullanılmasındaki amaç, son dönemde büyük gelişmelere imza atılmış olan gömülü sistemler ve özellikle FPGA teknolojisinde YSA algoritmaları gerçekleyerek bu işlemlerin sayısal kapılar seviyesinde ve çok hızlı bir biçimde gerçekleşmesini sağlamaktır. Bu amaçla kullanılacak olan FPGA'in iç mimarisi ve değişik uygulamaları, bu bölümde anlatılmıştır.

Çalışmanın dördüncü aşamasında, MATLAB ile benzetimi ve testleri yapılarak eğitimi tamamlanmış örnek ağın FPGA'de gerçeklenmesi için gerekli olan kodun yazımına geçilmiştir. Bu aşamada kullanılan tasarım dili VHDL'dir. VHDL ile yapılacak tasarımda izlenecek yöntemler, sebepleriyle açıklanmıştır. Kodun tasarımını takiben benzetimine geçilmiş, benzetim sonuçları ayrıntılı biçimde incelenerek yorumlanmıştır.

2. ÇOK KATMANLI ALGILAYICI YAPISI

Çok katmanlı algılayıcı yapısı, giriş nöron katmanı, çıkış nöron katmanı ve sayısı ihtiyaca göre belirlenmiş gizli nöron katmanlarından oluşmaktadır. Her katmanın çıkışı, bir sonraki katmanın girişi olmakta ve böylelikle çıkışa ulaşılmaktadır. Eğitilmiş bir çok katmanlı algılayıcı yapısında, nöronlar arasındaki bağlantı ağırlıkları eğitim sürecinde hesaplanmış belirli değerlerdedirler. Yani, bir çok katmanlı algılayıcı yapısını eğitmek için arzu edilen çıkışla elde edilen çıkış arasındaki hata hesabından yararlanılarak aşağıda açıklanacak yöntem ve formüller yardımıyla nöronlar arasındaki ağırlıkları hesaplamak gerekmektedir [1].

Örnek olarak, 10 nöronlu giriş katmanına, 4 ve 3 nöronlu 2 gizli katmana ve 2 nöronlu çıkış katmanına sahip bir çok katmanlı algılayıcı yapısını inceleyelim.



Şekil 2.1 : Örnek bir çok katmanlı algılayıcı katman yapısı

Şekil 2.1'de görüldüğü gibi, 9 girişimiz olmasına rağmen, girişe ve gizli katmanlara BIAS (1) değeri de giriş olarak verilmektedir.

$$\begin{bmatrix} V_1^1 \\ V_2^1 \\ V_3^1 \\ V_4^1 \end{bmatrix} = \begin{bmatrix} w_{11}^1 & w_{12}^1 & w_{13}^1 & w_{14}^1 & w_{15}^1 & w_{16}^1 & w_{17}^1 & w_{18}^1 & w_{19}^1 & w_{110}^1 \\ w_{21}^1 & w_{22}^1 & w_{23}^1 & w_{24}^1 & w_{25}^1 & w_{26}^1 & w_{27}^1 & w_{28}^1 & w_{29}^1 & w_{210}^1 \\ w_{31}^1 & w_{32}^1 & w_{33}^1 & w_{34}^1 & w_{35}^1 & w_{36}^1 & w_{37}^1 & w_{38}^1 & w_{39}^1 & w_{310}^1 \\ w_{41}^1 & w_{42}^1 & w_{43}^1 & w_{44}^1 & w_{45}^1 & w_{46}^1 & w_{47}^1 & w_{48}^1 & w_{49}^1 & w_{410}^1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ 1 \end{bmatrix} \quad \textbf{(2.1)}$$

Giriş katmanıyla 1. gizli katman arasındaki ağırlıkları gösteren w^1, 4*10'luk bir matrisle ifade edilir. Başlangıçta bu matrisin elemanlarına -0,5 ile 0,5 arasında rastgele değerler verilmiştir. Bu ağırlık matrisiyle girişler çarpılarak da, 4*1'lik V^1 matrisi elde edilir **(2.1)**. Bu matris, giriş-çıkış grafiğinin dikliğini ifade eder ve 1. katmanın çıkışını elde etmede kullanılır **(2.2)** .

$$y = f(V) = 1 \Big/ 1 + e^{-aV} \quad \textbf{(2.2)}$$

$$\begin{bmatrix} y_1^1 \\ y_2^1 \\ y_3^1 \\ y_4^1 \end{bmatrix} = \begin{bmatrix} f(V_1^1) \\ f(V_2^1) \\ f(V_3^1) \\ f(V_4^1) \end{bmatrix} \quad \textbf{(2.3)}$$

Bizim örneğimiz için de $y^1=f(V^1)$ olmaktadır **(2.3)**. Elde edilen 4*1'lik y^1 matrisi diğer katman için yapılacak hesaplamalarda giriş olarak kullanılacaktır.

1. gizli katmanla 2. gizli katman arasındaki w^2 ağırlık matrisimiz, 5 giriş 3 çıkış nöronu olduğu için 3*5'lik bir matris olacaktır **(2.4)**. Yukarıda yapılan hesaplamalara benzer şekilde;

$$\begin{bmatrix} V_1^2 \\ V_2^2 \\ V_3^2 \end{bmatrix} = \begin{bmatrix} w_{11}^2 & w_{12}^2 & w_{13}^2 & w_{14}^2 & w_{15}^2 \\ w_{21}^2 & w_{22}^2 & w_{23}^2 & w_{24}^2 & w_{25}^2 \\ w_{31}^2 & w_{32}^2 & w_{33}^2 & w_{34}^2 & w_{35}^2 \end{bmatrix} \begin{bmatrix} y_1^1 \\ y_2^1 \\ y_3^1 \\ y_4^1 \\ 1 \end{bmatrix} \quad \textbf{(2.4)}$$

$$V^2_{3*1} = w^2_{3*5} * y^1_{5*1} \quad \textbf{(2.5)}$$

V^2 matrisi elde edilir **(2.5)**. Bu matris de $y^2_{3*1} = f(V^2)$ formülünde yerine konarak 1. gizli katmanın çıkış değerleri elde edilir **(2.6)**.

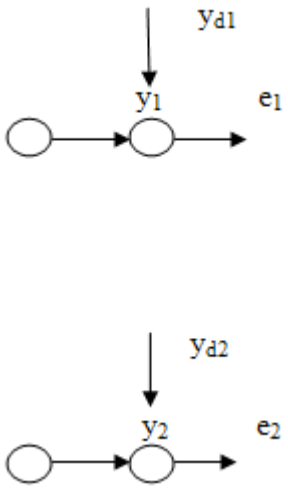
$$\begin{bmatrix} y_1^2 \\ y_2^2 \\ y_3^2 \end{bmatrix} = \begin{bmatrix} f(V_1^2) \\ f(V_2^2) \\ f(V_3^2) \end{bmatrix} \quad \textbf{(2.6)}$$

3. gizli katmanla çıkış katmanı arasındaki ağırlık hesabı da benzer şekildedir. 2*4'lük w^0 matrisi ve 4*1'lik y^2 matrisi çarpılarak V^0_{2*1} matrisi elde edilir **(2.7)**. Bu matris de $y^0 = f(V^0)$ formülünde yerine konarak çok katmanlı algılayıcımızın çıkış değerleri elde edilmiş olur **(2.8)**.

$$\begin{bmatrix} V_1^0 \\ V_2^0 \end{bmatrix} = \begin{bmatrix} w_{11}^0 & w_{12}^0 & w_{13}^0 & w_{14}^0 \\ w_{21}^0 & w_{22}^0 & w_{23}^0 & w_{24}^0 \end{bmatrix} \begin{bmatrix} y_1^2 \\ y_2^2 \\ y_3^2 \\ 1 \end{bmatrix} \quad \textbf{(2.7)}$$

$$\begin{bmatrix} y_1^0 \\ y_2^0 \end{bmatrix} = \begin{bmatrix} f(V_1^0) \\ f(V_2^0) \end{bmatrix} \quad \textbf{(2.8)}$$

Elde ettiğimiz çıkış değerileri, -0,5 ile 0.5 değerleri arasında rastgele olarak seçilmiş w değerleri kullanılarak hesaplandığından, elde ettiğimiz çıkış arzulanan çıkış olmayacaktır. Bizim, arzulanan çıkış değerleriyle elde edilen çıkış değerleri arasındaki hata hesabından yararlanarak ve geri yönde ilerleyerek istenilen çıkışı verecek uygun ağırlıkları hesaplamamız gerekmektedir. Bu aşama, çok katmanlı algılayıcımızın "öğrenme" aşamasıdır.



Şekil 2.2 : Çıkış katmanındaki hata gösterimi

Şekil 2.2'ye göre;

$e_1=(y_{d1}-y_1)$, $e_2=(y_{d2}-y_2)$ **(2.9)**

$\varepsilon = ½ (e_1^2 + e_2^2)$ **(2.10)**

Bu hata hesabından yeni ağırlık değerlerine geçiş ise;

$W(k+1)=w(k)- \eta \nabla \varepsilon$ **(2.11)**

ile olmaktadır.

w^0_{11} için $\nabla \varepsilon$ hesabı yaparsak;

$$\frac{\partial \varepsilon}{\partial w_{11}^0} = \frac{\partial \varepsilon}{\partial e_1} \cdot \frac{\partial e_1}{\partial y_1^0} \cdot \frac{\partial y_1^0}{\partial V_1^0} \cdot \frac{\partial V_1^0}{\partial w_{11}^0} = e_1 . (-1) . f'(V^0_1) . y^2_1 \quad \textbf{(2.12)}$$

elde ederiz **(2.12)**. 2*4'lük w^0 matrisinin her elemanı için yukarıdaki işlemi tekrarlarsak **(2.13) – (2.20)** arasındaki sonuçları elde ederiz.

w^0_{11} için $\nabla \varepsilon = e_1 . (-1) . f'(V^0_1) . y^2_1$ **(2.13)**

w^0_{12} için $\nabla \varepsilon = e_1 . (-1) . f'(V^0_1) . y^2_2$ **(2.14)**

w^0_{13} için $\nabla \varepsilon = e_1 . (-1) . f'(V^0_1) . y^2_3$ **(2.15)**

w^0_{14} için $\nabla \varepsilon = e_1 . (-1) . f'(V^0_1) . 1$ **(2.16)**

w^0_{21} için $\nabla \varepsilon = e_2 . (-1) . f'(V^0_2) . y^2_1$ **(2.17)**

w^0_{22} için $\nabla \varepsilon = e_2 . (-1) . f'(V^0_2) . y^2_2$ **(2.18)**

w^0_{23} için $\nabla\varepsilon$ = $e_2 . (-1) . f'(V^0_2) . y^2_3$ **(2.19)**

w^0_{24} için $\nabla\varepsilon$ = $e_2 . (-1) . f'(V^0_2) . 1$ **(2.20)**

Dikkat edersek, w^0 matrisinin 1. satırındaki elemanların hepsinin ilk 3 çarpanı aynı olmakta, diğer çarpan da o katmanın giriş değeri olmaktadır. Burada (-1) çarpanını gradient formülünün başına alarak, aynı olan diğer iki çarpanı yerel gradient adı altında bir ifadede birleştirmek mümkündür. Bu ifadenin matris gösterimi **(2.21)**'deki gibidir.

$$\begin{bmatrix}\delta_1^0\\ \delta_2^0\end{bmatrix} = \begin{bmatrix}e_1\\ e_2\end{bmatrix} \cdot * \begin{bmatrix}f'(V_1^0)\\ f'(V_2^0)\end{bmatrix} \qquad \textbf{(2.21)}$$

Sonuç olarak çıkış katmanıyla 2. gizli katman arasındaki ağırlık hesaplama ifadesi;

$$w^0_{2\times4}(k+1) = w^0_{2\times4}(k) + \eta \begin{bmatrix}\delta_1^0\\ \delta_2^0\end{bmatrix}_{2\times1} \begin{bmatrix}y_1^2 & y_2^2 & y_3^2 & 1\end{bmatrix}_{1\times4} \qquad \textbf{(2.22)}$$

halini almaktadır.

2. gizli katmanla 1. gizli katman arasındaki ağırlıkları hesaplamak istediğimizde ise, öncekine benzer bir yol izlenir, ancak gradient hesabında bazı farklılıklar ortaya çıkmaktadır. Bu iki katman arasındaki ağırlık hesabı için örnek olarak w^2_{12} ağırlığının hesaplanışını inceleyelim.

w^2_{12} için $\nabla\varepsilon$ hesabı **(2.23a-b-c)** :

$$\frac{\partial\varepsilon}{\partial w_{12}^2} = \frac{\partial\varepsilon}{\partial e_1}\frac{\partial e_1}{\partial y_1^0}\frac{\partial y_1^0}{\partial V_1^0}\frac{\partial V_1^0}{\partial y_1^2}\frac{\partial y_1^2}{\partial V_1^2}\frac{\partial V_1^2}{\partial w_{12}^2} + \frac{\partial\varepsilon}{\partial e_2}\frac{\partial e_2}{\partial y_2^0}\frac{\partial y_2^0}{\partial V_2^0}\frac{\partial V_2^0}{\partial y_1^2}\frac{\partial y_1^2}{\partial V_1^2}\frac{\partial V_1^2}{\partial w_{12}^2} \qquad \textbf{(2.23a)}$$

$$\frac{\partial\varepsilon}{\partial w_{12}^2} = \frac{1}{2}2e_1 \,.(-1).f'(V_1^0).w_{11}^0.f'(V_1^2).y_2^1 + \frac{1}{2}2e_2 \,.(-1).f'(V_2^0).w_{21}^0.f'(V_1^2).y_2^1 \qquad \textbf{(2.23b)}$$

$$\frac{\partial\varepsilon}{\partial w_{12}^2} = \delta_1^0.w_{11}^0.f'(V_1^2).y_2^1 + \delta_2^0.w_{21}^0.f'(V_1^2).y_2^1 \qquad \textbf{(2.23c)}$$

Bu hesaplardan sonra bir önceki katmanda yaptığımız gibi hesaplamayı genelleştirdiğimizde yerel gradient ifadesi;

$$\begin{bmatrix}\delta_1^2\\ \delta_2^2\\ \delta_3^2\end{bmatrix}_{3\times1} = \left[(w^o)^T{}_{3\times2}\begin{bmatrix}\delta_1^0\\ \delta_2^0\end{bmatrix}_{2\times1}\right]_{3\times1} \cdot * \begin{bmatrix}f'(V_1^2)\\ f'(V_2^2)\\ f'(V_3^2)\end{bmatrix}_{3\times1} \quad \textbf{(2.24)}$$

şeklini alır. Burada dikkat edilmesi gereken husus, 2 $\times$4'lük w^0 matrisinin transpozesini alırken, bias (1) ile çarpılan 4. sütunun çıkarılmış olmasıdır **(2.25)**.

$$\begin{bmatrix}V_1^0\\ V_2^0\end{bmatrix} = \begin{bmatrix}w_{11}^0 & w_{12}^0 & w_{13}^0 & \cancel{w_{14}^0}\\ w_{21}^0 & w_{22}^0 & w_{23}^0 & \cancel{w_{24}^0}\end{bmatrix}\begin{bmatrix}y_1^2\\ y_2^2\\ y_3^2\\ 1\end{bmatrix} \quad \textbf{(2.25)}$$

Sonuç olarak 2. gizli katman ile 1. gizli katman arasındaki ağırlık hesaplama ifadesi;

$$w^2{}_{3\times5}(k+1) = w^2(k) + \eta.\begin{bmatrix}\delta_1^2\\ \delta_2^2\\ \delta_3^2\end{bmatrix}_{3\times1}\begin{bmatrix}y_1^1 & y_2^1 & y_3^1 & y_4^1 & 1\end{bmatrix}_{1\times5} \quad \textbf{(2.26)}$$

şeklini alır.

1. gizli katmanla giriş katmanı arasındaki ağırlık güncelleme yöntemi, 2. gizli katman ile 1. gizli katman arasındaki ağırlık güncellemesi sırasında uygulanan yöntemle aynıdır. Benzer yaklaşımlarla δ^1 yerel gradienti;

$$\begin{bmatrix}\delta_1^1\\ \delta_2^1\\ \delta_3^1\\ \delta_4^1\end{bmatrix} = \left[(w^2)_{4\times3}^T\begin{bmatrix}\delta_1^2\\ \delta_2^2\\ \delta_3^2\end{bmatrix}\right] \cdot * \begin{bmatrix}f'(V_1^1)\\ f'(V_2^1)\\ f'(V_3^1)\\ f'(V_4^1)\end{bmatrix} \quad \textbf{(2.27)}$$

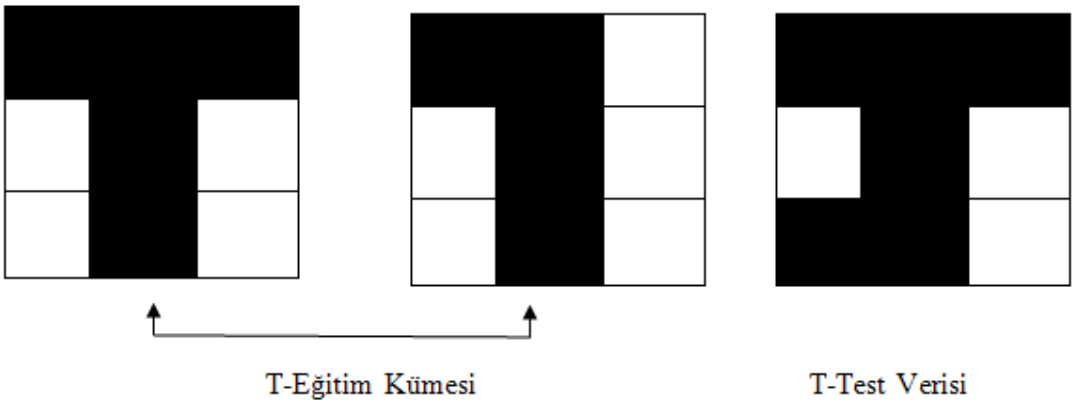
olarak bulunur. 1. gizli katmanla giriş katmanı arasındaki ağırlık hesaplama ifadesi ise;

$$w_{4\times10}^1(k+1) = w_{4\times10}^1(k) + \eta \cdot \begin{bmatrix}\delta_1^1\\ \delta_2^1\\ \delta_3^1\\ \delta_4^1\end{bmatrix}\begin{bmatrix}x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 & 1\end{bmatrix} \quad \textbf{(2.28)}$$

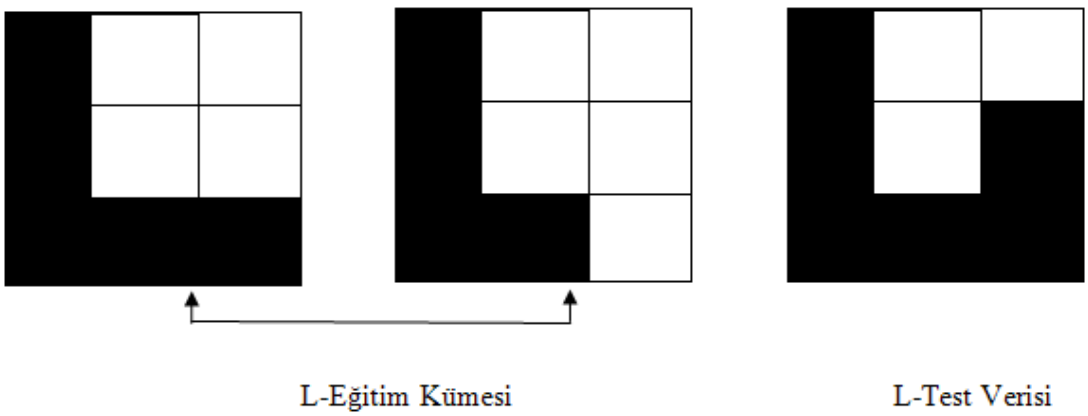
şeklinde hesaplanır.

3. T VE L HARFLERİNİ BİRBİRİNDEN AYIRT EDEN ÇOK KATMANLI ALGILAYICI YAPISI VE MATLAB ORTAMINDA BENZETİMİ

Önceki bölümde açıklanan çok katmanlı algılayıcı yapısına örnek bir uygulama olarak, 3*3 lük 9 pikselle tanımlanmış T ve L harflerini birbirinden ayıralım. Bu amaçla belirlenmiş olan T harfi için eğitim ve test kümeleri Şekil 3.1'de, L harfi için eğitim ve test kümeleri ise Şekil 3.2'de gösterilmiştir:



Şekil 3.1 : T harfi eğitim ve test kümesi



Şekil 3.2 : L harfi eğitim ve test kümesi

Kodun MATLAB ortamında çalıştırılması sonucu; sinir ağı yapımız, eğitim kümeleri aracılığı ile T ve L harflerini öğrenecektir. Bu aşamada, bir önceki bölümde

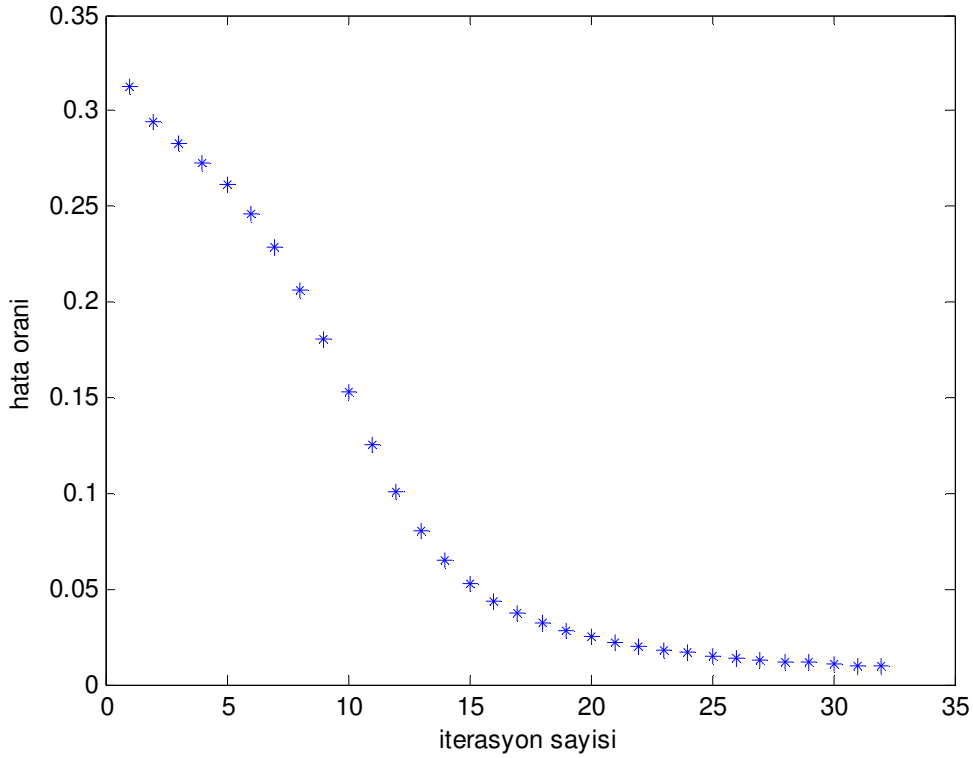
açıklanan teorik yaklaşımlar adım adım uygulanacak, rastgele alınan girişler sonucu yapılan hesaplamalar ve bunlar sonucu elde edilen hata değerleri kullanılarak ağırlıkların güncellenmesi suretiyle ağ eğitimi gerçeklenecektir. Program, hata oranı belli bir sınırın altına düştüğünde duracak ve test kümesini giriş değeri olarak alarak eğitimin doğru olup olmadığını gösterecektir.

3.1 Program Kodu ve Çıktıları

```
%%%%%cok katmanli%%%%%
clear all, clc;
%%%%parametreler%%%%%
mu1=0.2;
mu2=0.2;
a=3;
%%%%%%%%%%
ng=9;
nn1=4;
nn2=3;
nn0=2;
iterasyon_s=100;
egitim_s=4;
%%%%ilk degerler%%%
agirlik_1=0.25*randn(nn1,ng+1);
agirlik_2=0.25*randn(nn2,nn1+1);
agirlik_o=0.25*randn(nn0,nn2+1);
%%%%%%%%%%%%%%%%%%%%%%


egitim_kumesi_g=[0.9 0.9 0.9 0.1 0.9 0.1 0.1 0.9 0.1;
                 0.9 0.1 0.1 0.9 0.1 0.1 0.9 0.9 0.9;
                 0.9 0.9 0.1 0.1 0.9 0.1 0.1 0.9 0.1;
                 0.9 0.1 0.1 0.9 0.1 0.1 0.9 0.9 0.1];
egitim_kumesi_c=[1 0;
                 0 1;
                 1 0;
                 0 1];

   for l=1:iterasyon_s
   for k=1:egitim_s

   giris_1=[egitim_kumesi_g(k,:)';1];
   cikis=egitim_kumesi_c(k,:)';

   v1=agirlik_1*giris_1;
   y1=1./(1+exp((-1)*a*v1));
   giris_2=[y1;1];
   v2=agirlik_2*giris_2;
   y2=1./(1+exp((-1)*a*v2));
   giris_o=[y2;1];
   vo=agirlik_o*giris_o;
   yo=1./(1+exp((-1)*a*vo));
```

```
    e=cikis-yo;
    hata=0.5*e'*e;

    hata_kon(k,1)=hata(:);

    %fdrv_o=a*yo.*(1-yo);
    %%%egitim cikis katmani%%%

    trvo = a*yo.*(1-yo); %logsig hata fonksiyonunun cikis katmani icin turevi
    yerel_grao=e.*trvo; %cikis katmani yerel gradienti

    %%%egitim 2. gizli katman%%%

    trv2 = a*y2.*(1-y2); %logsig hata fonksiyonunun 2. gizli katman icin turevi
    indo = [agirlik_o(:,1:nn2)]; %indirgenmis agirlik matrisi
    yerel_gra2 = indo'*yerel_grao.*trv2; %2. gizli katman yerel gradienti

    %%%egitim 1. gizli katman%%%

    trv1 = a*y1.*(1-y1); %logsig hata fonksiyonunun 1. gizli katman icin turevi
    ind2 = [agirlik_2(:,1:nn1)]; %indirgenmis agirlik matrisi
    yerel_gra1 = ind2'*yerel_gra2.*trv1; %1. gizli katman yerel gradienti

    agirlik_o = agirlik_o+mu2.*yerel_grao*giris_o';  %cikis katmani parametresi olarak mu2 kullanildi
    agirlik_2 = agirlik_2+mu2.*yerel_gra2*giris_2';  %2.gizli katman parametresi olarak mu2
kullanildi
    agirlik_1 = agirlik_1+mu1.*yerel_gra1*giris_1'; %1.gizli katman parametresi olarak mu1
kullanildi

    end %egitim

    hata_top=0.25*sum(hata_kon');
    hata_top_v(l,1)=hata_top(:);

    hata_lim=0.01;

    if hata_top<hata_lim
    break;
    end %if
    end %iterasyon

    plot(hata_top_v,'*')

    %%%%%%test%%%%%%%%

    test_kumesi_g=[0.9 0.1 0.1 0.9 0.1 0.9 0.9 0.9 0.9];

    tgiris_1=[test_kumesi_g(1,:)';1];
    v1=agirlik_1*tgiris_1;
    y1=1./(1+exp((-1)*a*v1));
    giris_2=[y1;1];
    v2=agirlik_2*giris_2;
    y2=1./(1+exp((-1)*a*v2));
    giris_o=[y2;1];
    vo=agirlik_o*giris_o;
    yot=1./(1+exp((-1)*a*vo))
```

Programımız, sinir ağımızı giriş verilerimize göre hata oranı 0.01'e ulaşıncaya kadar eğitmekte, sonra da test verileriyle ağı kontrol etmektedir.



Şekil 3.3 : L harfi için hata oranı – iterasyon sayısı grafiği

Şekil 3.3, eğitim sürecinde hata oranıyla iterasyon sayısı arasındaki ilişkiyi vermektedir. Hata oranı, 32. iterasyonda istenilen değere (0.01) ulaşmış ve program sinir ağı eğitimi tamamlanarak test verisi giriş olarak eğitilmiş ağa verilmiştir. Şekil 3.2'de belirtilen L harfinin test verisi sonucu elde edilen çıkış sonucu (yot) aşağıdadır:

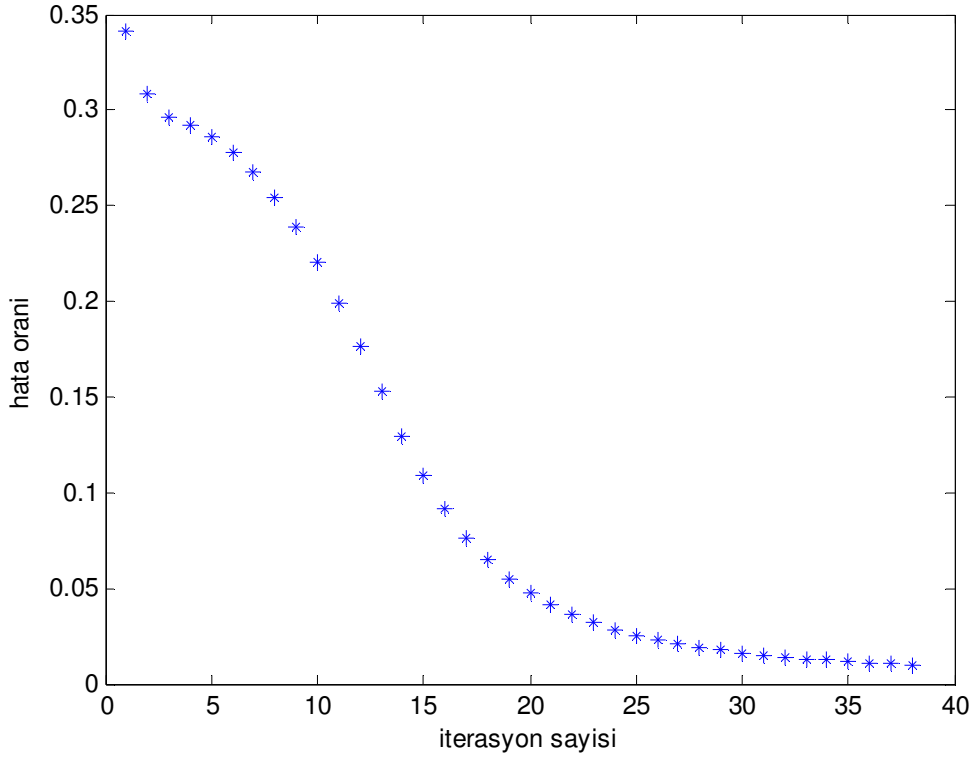
yot =

0.0815

0.9225

Görüldüğü gibi elde edilen sonuç, arzu edilen sonuç olan [0,1] değerine yakınsamaktadır. Sinir ağının L harfi için eğitimi başarılıdır.

T harfinin eğitimi ve testi için aynı program, test giriş kümesi; test_kumesi_g=[0.9 0.9 0.9 0.1 0.9 0.1 0.9 0.9 0.1] olarak düzenlenip tekrar çalıştırıldığında, Şekil 3.4'teki hata grafiği elde edilmektedir.



Şekil 3.4 : T harfi için hata oranı – iterasyon sayısı grafiği

Program, hata oranının 0.01 değerine ulaşması sonucu 38. iterasyonda son bulmuştur. Şekil 3.1'de verilen T harfi için test verisinin programa giriş olarak verilmesiyle de aşağıdaki sonuç (yot) elde edilmiştir:

yot =

0.9103

0.1156

Elde edilen sonuç, arzu edilen değer olan [1,0]'a yakınsamaktadır. Ağın T harfi için eğitimi başarılıdır. Unutulmamalıdır ki, bu değerler rastgele girişlere göre elde edilmiştir. Yani, program her çalıştığında elde edilen sayısal sonuçlar ve iterasyon

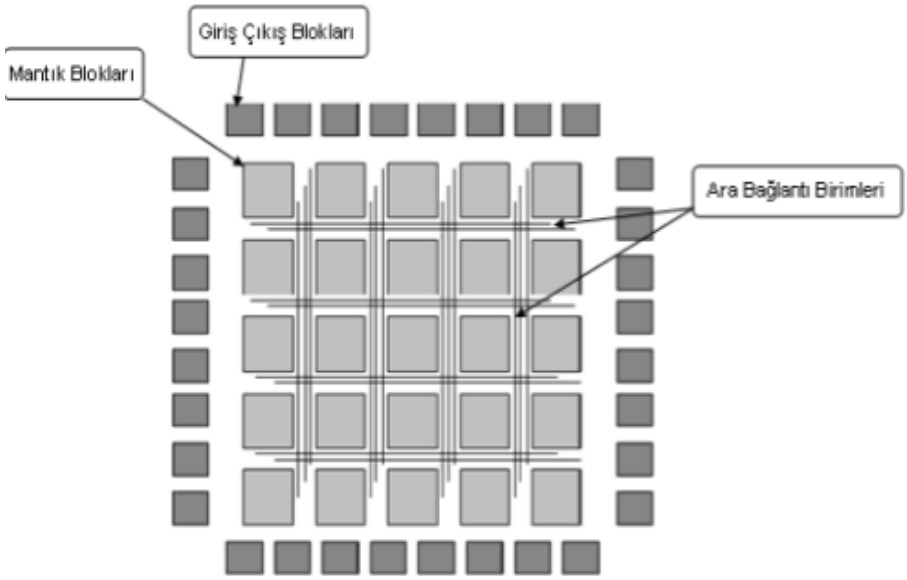
sayıları farklı olabilir. Ancak bu yakınsamayı, yani ağın eğitim başarısını değiştirmemektedir.

4. FPGA

4.1 Giriş

FPGA (Field Programmable Gate Array - Alanda Programlanabilir Kapı Dizileri), programlanabilir mantıksal bloklar ve bağlantılar içeren yarı iletken tümleşik devrelerdir. Mantıksal bloklar basit mantıksal kapıların (AND, XOR) işlemlerini yerine getirmek için programlanabildikleri gibi, daha karmaşık fonksiyonların da (şifre çözücüler, basit matematiksel fonksiyonlar vb.) işlemlerini yerine getirebilmektedirler. Birçok FPGA ayrıca hafıza yapısına sahiptir. Bunlar basit Flip-Floplar olabileceği gibi daha bütünsel hafıza blokları da olabilmektedir [2].

FPGA, Şekil 4.1'de görüldüğü gibi, programlanabilir mantık blokları, bu blok dizisini çevreleyen giriş-çıkış blokları ve ara bağlantılar olmak üzere düzenlenebilir üç ana bölümden oluşur. Programlanabilir mantık blokları, ara bağlantılar içerisine gömülü şekilde bulunur. Programlanabilir mantık bloklarının yapılandırılması ve bu bloklar arasındaki iletişim ara bağlantılar sayesinde gerçekleşir. Giriş çıkış blokları, ara bağlantılar ile bütünleşmiş devrenin paket bacakları arasındaki ilişkiyi sağlar.



Şekil 4.1 : FPGA genel yapısı

Sistem tasarımcısı ihtiyaç duyduğunda mantıksal blokların birbirleri ile bağlantı kurmasını sağlayabilir. Bağlantı hiyerarşisi bu tasarıma izin verir. FPGA'lerin hem mantıksal kapıları hem de bağlantıları tasarımcı tarafından programlanabilmektedirler. Bu nedenle isimleri field-programmable (alanda programlanabilir) dir.

FPGA'ler genellikle benzerleri ASIC'lere (application-specific integrated circuit) göre daha yavaştırlar. Ancak ASIC'ler karmaşık tasarımların üstesinden gelemezler, aynı zamanda da daha fazla güç tüketirler. Bunların yanında ASIC'lerin geliştirme zamanlarının kısa olması, çıkan hataların sahada düzeltilebilmesi ve daha az tekrarlamayan mühendislik maliyetine sahip olması gibi avantajları vardır. FPGA'lerin daha az esnek olan ve bir kere tasarlandıktan sonra değiştirilemeyen versiyonlarını satıcılar daha ucuza satabilmektedirler. Tasarımlar standart FPGA'ler üzerinde gerçekleştirilip ASIC'lere daha çok benzeyen sabit versiyonlarına aktarılır. FPGA'lerin diğer bir alternatifi de CPLD'lerdir (complex programmable logic devices).

4.2 Tarihçe

FPGA'lerin tarihsel kökleri 1980'lerin ortasından önceki CPLD'lere dayanmaktadır. Ross Freeman, Xilinx 1984 yılında FPGA'i birlikte bulmuşlardır. CPLD'ler ve FPGA'ler oldukça fazla sayıda programlanabilir mantıksal element içermektedirler. CPLD'in mantık kapısı sayısı birkaç binlerden on binler düzeyine kadar olabilmektedir. FPGA'lerde ise bu sayı on binlerden milyonlarcaya kadar çıkabilmektedir.

CPLD'ler ile FPGA'ler arasındaki başlıca fark mimari ile ilgilidir. Bir CPLD oldukça sınırlı bir yapıya sahiptir. Çarpımların toplamı şeklinde ifade edilen bir ya da daha fazla mantık dizisi nispeten daha az sayıda zaman yazmacını (clock register) beslemektedir. Bunun sonucu daha az esnekliktir (öngörülebilir zamanlama gecikmesi ve daha yüksek oranda bağlantı oranı avantajı ile birlikte). Diğer taraftan FPGA mimarisi, bağlantı konusunda baskındır. Bu durum FPGA'leri daha esnek (tasarım aralığı açısından uygulaması daha elverişli) ve daha karmaşık tasarımlara hitap eden bir hale getirmektedir.

CPLD'ler ile FPGA'ler arasındaki dikkate değer bir başka fark, bir çok FPGA üst düzey gömülü fonksiyonlara (toplama,çarpma) ve gömülü hafızalara sahiptir. Bununla ilişkili olan önemli fark, birçok modern FPGA'de bütün ya da kısmi sistem yeniden konfigürasyonu ve anında (on the fly) tasarım değişimi ve system yükseltmesi normal sistem operasyonunun bir parçasıdır. Bazı FPGA'lerde parçalı yeniden konfigürasyon yapılabilmektedir. Bu şekilde sistemin bir parçası yeniden yapılandırılırken diğer kısımları çalışmaya devam etmektedir.

4.3 Çağdaş Geliştirmeler

Son zamanlardaki iri taneli (coarse-grained) mimari yaklaşım, geleneksel FPGA'lerin içindeki bağlantıları ve mantıksal blokları dahili mikroişlemciler ve ilişkili çevresel aygıtlarla birleştirerek hazır bir forma getirmede (Programlanabilir bir yongadaki sistem-system on a programmable chip) bir adım ileriye götürmüştür. Bu tarz melez örnekler Xilinx Virtex-II PRO ve Virtex-4 cihazlarında bulunabilir. Bu cihazlarda bir ya da birden fazla FPGA mantıksal yapısı ile yerleştirilmiş PowerPC işlemcileri vardır. Diğer benzeri bir cihaz, Atmel FPSLIC dir. İçerisinde Atmel'in programlanabilir lojik yapısının kombinasyonu bulunan bir AVR işlemci kullanır.

Bir alternatif yaklaşım, FPGA mantığıyla gerçekleştirilmiş Soft işlemci çekirdeklerini gerçekleştirmek için Hard-macro işlemcileri kullanmaktır.

Önceden bahsedildiği gibi, Birçok modern FPGA tekrar programlanabilmektedir.

Bu önde gelen düşünce (Yeniden programlanabilir sistemler) işlemcilerin kendilerini yeniden yapılandırmalarına yardım etmektedir. Mitrionics firmasına ait Mitrion Virtual Processor FPGA ile gerçekleştirilmiş yeniden yapılandırılabilir soft processor bu sistemlere bir örnektir. Bu örnek dinamik olarak yeniden yapılandırmayı desteklememektedir fakat bunun yerine kendini özel bir programa adapte edebilmektedir.

4.4 Uygulamalar

FPGA uygulamaları; sayısal işaret işleme, radyo haberleşme sistemi, uzay, savunma sistemleri, ASIC prototiplendirme, medikal resimleme, robotik, ses tanıma, şifreleme, biyoinformatik, bilgisayar donanım emülasyon gibi ve genişleyen diğer

alanlarda kullanılmaktadır. FPGA başlangıçta CPLD'lerin rakibi olarak başladı ve benzeri bir alanda mücadale etti. Büyüklükleri, kapasiteleri ve hızları attıkça pazarda SOC (full systems on chips) yaklaşımı ile daha da fazla yer almaya başladı.

FPGA'ler özellikle uygulama ve paralellik kullanılan algoritmaları mimarisinde sunmaktadır. Şifreleme algoritmalarında özellikle kod kırma işlemlerinde bütün olası senaryoları deneme durumlarında kullanılmaktadır.

FFT (Hızlı Fourier Dönüşümü) ve konvolüsyon işlemlerinin bir mikroişlemci yerine FPGA de gerçekleştirildiği yüksek performanslı hesaplamalarda FPGA'ler artarak kullanılmaktadır.

FPGA'lerin hesaplama işlemlerinde kullanımı reconfigurable computing (yeniden tasarlanabilir hesaplama) olarak bilinir.

FPGA üzerindeki mantıksal kaynakların özünde olan paralellik 500MHz'in altındaki saat frekanslarında bile hatırı sayılır ölçüde hesaplama çıktısı verebilmektedir. Örneğin, 2007 nesli FPGA'ler her bir saat darbesinde aşağı yukarı 100 hassas kayan noktalı üniteyi yerine getirebilmektedirler. FPGA'lerin esnekliği kesinliği azaltan alternatif maliyetiyle daha çok performans ve paralel birim artışına izin vermektedir. Bu yeni işleme yaklaşımı yeniden ayarlanabilir hesaplama olarak adlandırılır. Zaman açısından yoğun görevler yerini yazılımlar yerine FPGA'lere bırakmaktadır.

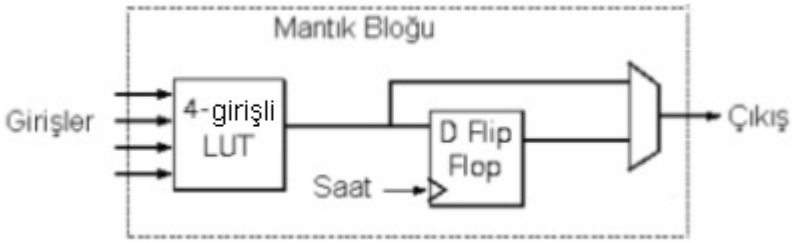
FPGA'lerin yüksek performans hesaplamalarında kabul görmeleri FPGA'lerin tasarım zorluğundan dolayı halen belli bir limittedir. Geleneksel yazılımlarla aşırı derece geriye dönüşlü tasarım araçları karşılaştırıldığında küçük bir değişiklik bile 4-8 saat zaman almaktadır.

4.5 Mimari

Tipik temel mimari yapılandırılabilir mantıksal blok dizisi ve yönlendirme kanallarını içerir. Giriş çıkış noktaları satır ve boyuncadır. Genel olarak, bütün yönlendirme kanalları eşit genişliktedir (tellerin uzunluğu).

Bir uygulama çevrimi bir FPGA' de gerçekleştirilebilecek şekilde tasarlanmalıdır.

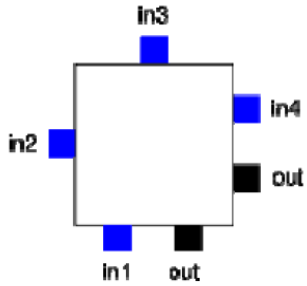
Klasik bir FPGA mantık bloğu Şekil 4.2'de görülebileceği gibi dört giriş lookup table ve bir flip-flop içerir. Son yıllarda üreticiler altı girişli olan versiyonlarını yüksek performanslı olduğunu iddia ettikleri ürünlerde kullanmaktadır.



Şekil 4.2 : Mantık bloğu yapısı

Lookup tablosuna kayıtlı ya da kayıtsız tek bir çıkış vardır. Mantıksal blok LUT(Look up table) için dört giriş ve bir saat girişine sahiptir. Saat sinyalleri özel bir amaç için ticari FPGA'ler içindeki atanmış yönlendirme ağlarına yönlendirilirken, diğer sinyaller ayrı olarak yönetilirler.

Şekil 4.3'te FPGA mantıksal blok kapıları görülebilmektedir.



Şekil 4.3 : Mantıksal blok kapıları

Çıkış kapısına mantıksal bloğun sağından ya da altından ulaşılabilirken, her bir girişe mantıksal bloğun tek bir kenarından ulaşılabilir.

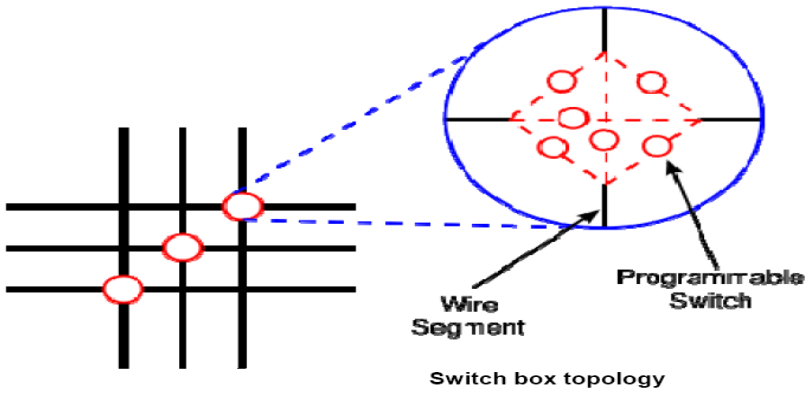
Her bir çıkış kapısı kendisine komşu olan kanallardaki bağlantı parçasıyla (wiring segment) bağlantı kurabilir.

Benzer olarak, bir giriş-çıkış kapısı kendisine komşu olan kanallardaki herhangi bir bağlantı parçası ile bağlantı kurabilir. Örnek olarak, entegrenin en üstündeki giriş-çıkış kapısı hemen altındaki herhangi bir yatay bağlantı paçası ile bağlantı kurabilir.

Genellikle, FPGA'lerde yönlendirme bölümlenmemiştir. Bu nedenle her bağlantı parçası sonlanmadan önce tek bir mantıksal blok içerir. Daha hızlı bağlantılar için, bazı FPGA mimarileri daha uzun yönlendirme yollarına sahiptir, bunlar birden fazla mantıksal blok içerirler.

Yatay ve dikey kanalların kesiştiği yer anahtarlama kutusu (switch box)'tur. Bu mimaride bir taraftan anahtarlama kutusu'na giriş yapıldığı zaman, kendisine komşu olan diğer üç tarafa yönlendirilebilmesini sağlayacak üç adet programlanabilir anahtar vardır. Anahtarların bu mimaride kullandığı bu şablon ya da topoloji düzlemsel ya da alan bazlı mimaridir. Bu anahtarlama kutusu topolojisinde, birinci izdeki tel sadece kendisine komşu olan birinci izdeki diğer tellere; ikinci izde, sadece kendisine komu olan ikinci izdeki diğer tellere bağlanabilir.

Şekil 4.4 anahtarlama kutusu bağlantılarını resimlemektedir.



Şekil 4.4 : Anahtarlama kutusu bağlantıları

Modern FPGA aileleri sabitlenmiş üst seviye işlevsellikleri içermek için silikon içinde tasarlanarak kapasitelerinin de üstüne çıkmışlardır. Silikon içinde gömülü bu yaygın işlevselliklere sahip olmak, alan ihtiyacını azaltmış ve ilkel tipleriyle karşılaştırıldığında hız artışı sağlamıştır.

FPGAler ayrıca geniş ölçüde silikon öncesi doğrulama, silikon sonrası doğrulama ve aygıt yazılımını da içeren sistem doğrulamasında kullanılmaktadır. Bu şekilde entegre üreticileri entegre fabrika da üretilmeden önce tasarımlarını tasdik edebilmekte bu da entegrenin pazara sürülme zamanını kısaltmaktadır.

4.6 FPGA Tasarım ve Programlama

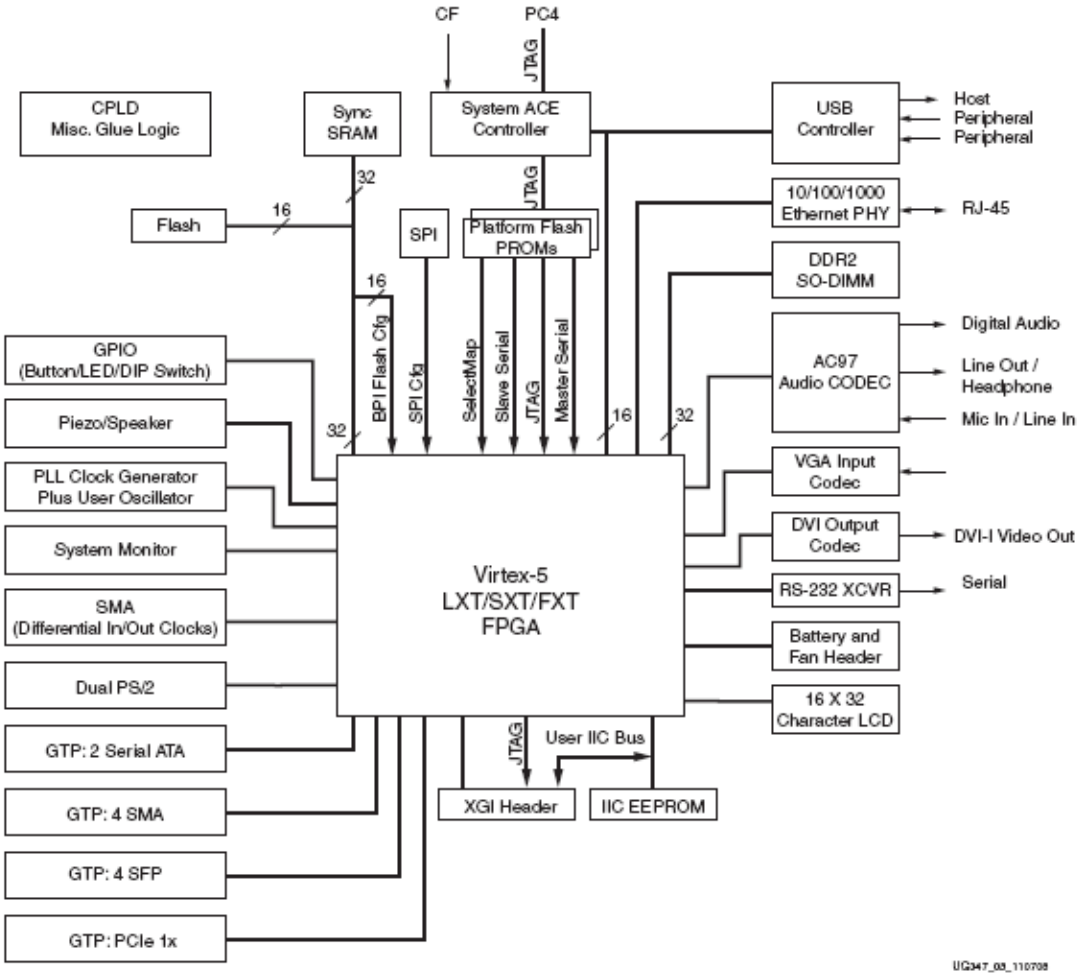
FPGA'in davranışını tasarlamak için kullanıcı HDL (Hardware Description Language-Donanım Tanımlama Dili) ya da şematik dizayn sağlamalıdır.Yaygın HDL'ler VHDL ve Verilog tur. EDA (Electronic design automation) kullanılarak netlist (bağlantı listesi) yaratılır. Netlist "place-and-route"(yerleştir-yolu çiz) yöntemi kullanılarak temel FPGA mimarisine uygun düşmektedir. "place-and-route" FPGA şirketinin ürünü olan yazılımlarla gerçekleştirilmektedir. Kullanıcı planı, place – route sonuçları üzerinden zaman analizi, benzetim ve diğer doğrulama metodolojileri kullanarak onaylayacaktır. Tasarım ve doğrulama işlemleri tamamlandıktan sonra yine FPGA şirketinin ürünü olan yazılımlarla binary dosyası yaratılılır ve bu dosya FPGA'i yeniden ayarlama amaçlı kullanılır. HDL'lerin assembly dili ile karşılaştırıldığında görülen tasarım güçlüklerini azaltmak için tasarım soyutlamayı yükseltmeye yönelik yönelimler vardır.

Karmaşık sistemlerin FPGA'lerde tasarlanmasını basitleştirmek için, önceden tanımlı karmaşık fonksiyon ve süreçler test ve optimize edilerek dizayn sürecine hız katılmıştır. Önceden tanımlı sürecler genelde "IP Cores" olarak adlandırılır.

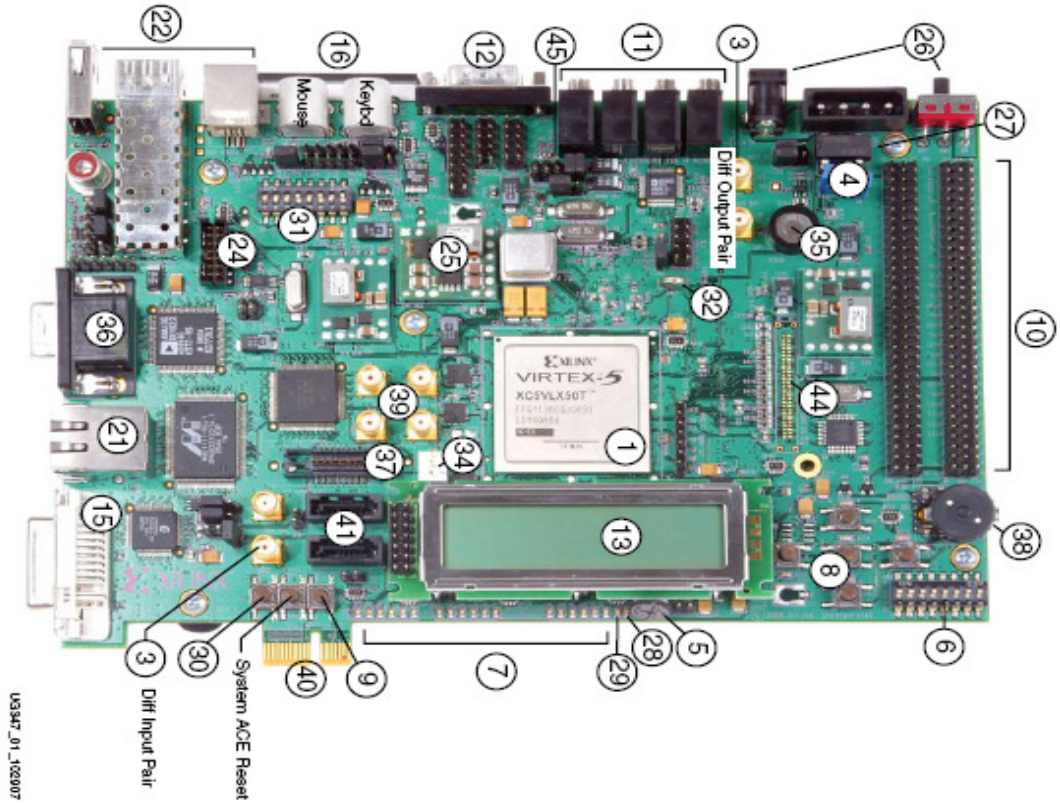
Tipik tasarım akışında, FPGA uygulama geliştirici tasarımı birden çok evrelerde baştan sona simule edecektir. Sistemi simüle etmek ve sonuçları gözlemlemek için VHDL ve Verilog'taki tanımları (Register Transfer Level description) test sıraları yaratılarak simüle edilir. Ardından, birleştirme motoru ile tasarım netlist için planlandığında (mapping), netlist kapı düzeyi tanımlamaya (gate level description) çevrilir. Simülasyon burada tekrarlanarak birleştirmenin hatasız olarak yapıldığı doğrulanır. Son olarak, tasarım FPGA'de hazırlanır. Bu aşamada yayılma ertelemesi eklenebilir ve simülasyon netliste not düşerek tekrar yapılabilir [3].

4.7 Virtex-5

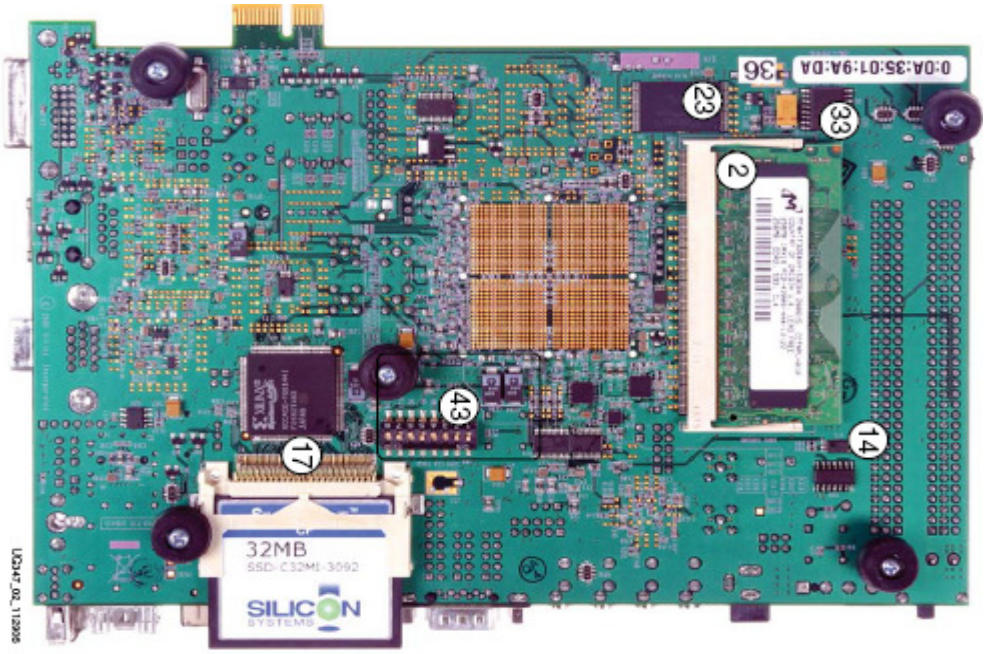
Bu çalışmada, hazırlanan tasarımın üzerinde koşturulması için, Xilinx firmasının ürettiği VIRTEX-5 FPGA tümleşik devresi ve onun üzerinde bulunduğu ML506 tasarım kartı kullanılmıştır. ML506'nın blok şeması Şekil 4.5'te, üst yüz ve alt yüz görünümü Şekil 4.6 ve Şekil 4.7'de gösterilmektedir.



Şekil 4.5 : VIRTEX-5 FPGA ML506 blok şeması



Şekil 4.6 : VIRTEX-5 FPGA ML506 tasarım kartı üst yüzü



Şekil 4.7 : VIRTEX-5 FPGA ML506 tasarım kartı alt yüzü

4.7.1 Virtex-5 detaylı tanıtım

Bu bölümde, Şekil 4.6 ve Şekil 4.7'de numaralandırılmış elemanların ne oldukları kısaca belirtilmiştir.

1. Virtex-5 FPGA
2. DDR2 SODIMM Ram hafıza modülü
3. Diferansiyel saat giriş ve çıkışı
4. Osilatörler
5. LCD parlaklık kontrast ayarı
6. Genel amaçlı anahtarlar
7. Kullanıcı ve hata LED'leri
8. Kullanıcı butonları
9. CPU reset butonu
10. Genişleme başlığı
11. Stereo AC97 ses kodeği
12. RS-232 seri kanalı
13. 16 karakter-2 satırlı LCD ekran
14. 8 kb EEPROM'lu IIC veri yolu
15. DVI bağlantısı
16. Fare ve klavye bağlantısı
17. Compact Flash kart bağlantısı
18. SRAM
19. 32 Mb flash hafıza
20. Xilinx XC95144XL CPLD
21. 3 hızlı Ethernet girişi
22. USB denetleyicileri
23. Xilinx XCF32P PROM konfigürasyon hafızaları

24. JTAG konfigürasyon girişi

25. Güç kaynağı

26. AC adaptörü ve güç girişi

27. Güç belirtme LED'i

28. FPGA'in başarıyla konfigüre edildiğini belirten DONE LED'i

29. FPGA'e gücün başarıyla geldiğini gösteren INIT LED'i

30. Programlama anahtarı

31. Adres ve durum konfigürasyon anahtarları

32. Şifreleme anahtar pili

33. SPI Flash hafızası

34. Fan denetleyicisi ve sıcaklık/gerilim ekranı

35. Basit ses tonları çıkartan piezo ses dönüştürücüsü

36. VGA görüntü kodek girişi

37. JTAG izleyici

38. Çevirici kodlayıcısı

39. GTP/GTX giriş ve çıkışları

40. PCI arayüzü

41. SATA bağlantısı

42. SFP bağlantısı

43. GTP/GTX saat devresi

44. Mantıksal analiz dokunma noktaları

45. Sistem ekranı [8]

5. VHDL KODU, BENZETİMİ VE SONUÇLARI

Bu aşamada, MATLAB aracılığı ile oluşturulmuş eğitilmiş çok katmanlı algılayıcı yapısının testinin sayısal bir donanım tanımlama dili olan VHDL ile gerçeklenmesine çalışılmıştır. Bu aşamada yapılacak devre tasarımında izlenebilecek yöntemler aşağıda açıklanmıştır:

5.1 Yöntem 1

- MATLAB benzetiminde 0,1 ve 0,9 arasında alınan giriş değerleri, kuvantalanarak 10 ve 90 olarak alınmıştır.

-MATLAB'dan elde edilen eğitilmiş ağın ağırlık değerleri, 100 ile kuvantalanmıştır.

-Aktivasyon fonksiyonunun payı 100 ile kuvantalanarak normalde -1 ile 1 arasında değişmesi gereken foksiyon değerinin -100 ile 100 arasında değişmesi sağlanmıştır.

-Aktivasyon fonksiyonunun gerçeklenmesi için look up table oluşturulmuştur. Bunun amacı, VHDL ile üstel fonksiyonların türevinin gerçekleştirme imkanı bulunmaması, fonksiyonun Taylor serisine açılımının ise tamsayı sınırı olan 32 bit (−2.147.483.648 ile +2.147.483.647) sınırını aşmasıdır. MATLAB ile elde edilen fonksiyon kuvantalanmış değerleri look up table ile VHDL'e aktarılmaktadr.

5.1.1 Yöntemin avantajları

Yöntem 1, tamsayı çalışmaya olanak vermesi, bunun sonucunda toplama ve çarpma işlemlerinin kolaylıkla yapılabilmesi açısından avantajlıdır. Tasarım sonucu elde edilecek deve şematiğinin FPGA tümleşik devresine sığma gibi bir problemi bulunmamaktadır.

5.1.2 Yöntemin dezavantajları

Yöntem 1, tamsayı çalışıldığı için işlem sonuçlarının tamsayı olarak yuvarlanması sonucunu doğurmaktadır. Ancak bu dezavantaj, sadece istenilen değere yakınsama

hatasını bir miktar arttıracağı ve sonucu değiştirmeyeceği için, kabul edilebilir bir dezavantajdır.

5.2 Yöntem 2

-Gerek giriş, gerekse ağırlık değerleri, MATLAB benzetiminde bulunan reel değerlerin IEEE 754 32 bit standardına göre elde edilmesiyle kullanılmıştır.

-32 bit kayan noktalı çarpma ve toplama fonksiyonları kullanılarak ağırlık hesaplamaları yuvarlamasız ve kuvantasız olarak reel değerler üzerinden yapılmıştır.

-Kayan noktalı işlemlerde sınır problemi olmadığından, aktivasyon fonksiyonu look up table olmaksızın tasarım içinde gerçeklenmiştir.

5.2.1 Yöntemin avantajları

Giriş ve ağırlık sayılarının reel değerleri kullanıldığı için, yuvarlamaya bağlı hata en az oranda çıkmaktadır. MATLAB'da elde edilen değerlere tam uyum göstermektedir.

5.2.2 Yöntemin dezavantajları:

32 bit kayan noktalı sayılarla çalışıldığı ve bu veri tipiyle çalışan toplama ve çarpma fonksiyonları bir çok kez çağırıldığı için, tasarlanan devre çok çok büyük ve karmaşık çıkmakta, sentezleme süresi kabul edilemez şekilde uzamakta, sonuçta sentezlenen devrede kullanılmakta olan VIRTEX 5 yongasına sığmamaktadır.

Çok katmanlı algılayıcı yapısının VHDL dili ile sentezi ve FPGA ile gerçeklenmesi konusunda, Yöntem 2'nin kabul edilemez sentez ve gerçekleme dezavantajları olduğu için, Yöntem 1 tercih edilmiştir [4], [5], [6], [7].

5.3 VHDL Kodu ve Açıklamalar

VHDL kodu, yeterince uzun olduğu için buraya konmamıştır. Kod, EK A'da mevcuttur. Burada sadece açıklamalar yapılacaktır.

-MATLAB benzetiminde olduğu gibi, sistemin 9 girişi vardır.Girişler "1" olduğunda 90, "0" olduğunda "10" değeri test girişi olarak atanmaktadır.

-MATLAB'dan elde edilmiş olan eğitilmiş ağın ağırlık değerleri sabit olarak kodun başında belirtilmiştir.

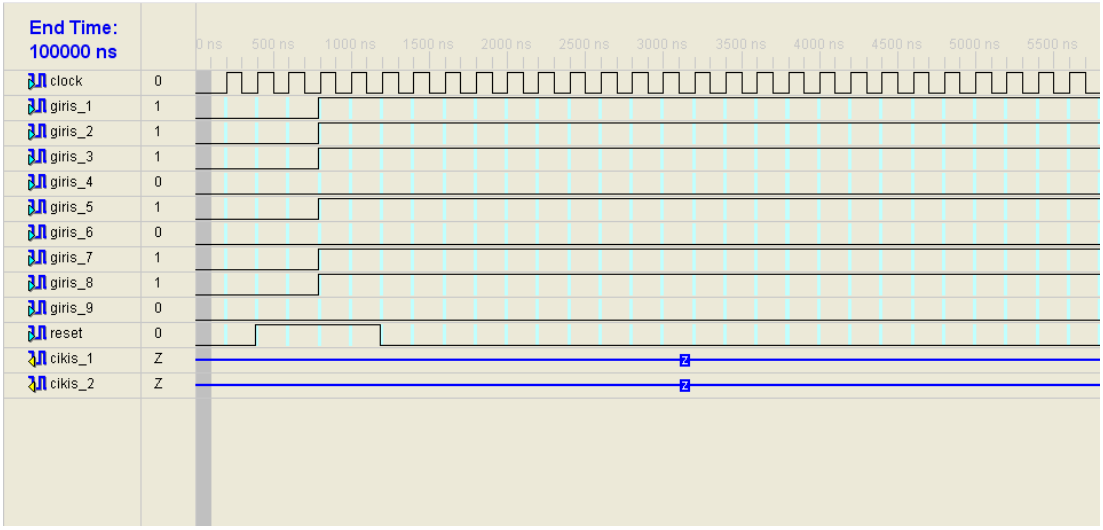
-Reset değeri “1” olduğunda, bütün değişken ilk değerleri “0” olarak atanmıştır.

-Durum makinesi kullanılarak, kodun daha rahat çalışması ve anlaşılması sağlanmıştır. Öncelikle o katmanın ara değeri v, 2. bölümde anlatıldığı gibi hesaplanmış, sonra bu değer kullanılarak aktivasyon fonksiyonu çıkış değeri y, look up table’dan çağırılarak elde edilmiştir. Her katman için, bu aşamalar durum makinesinin farklı durumlarında hesaplanmıştır.

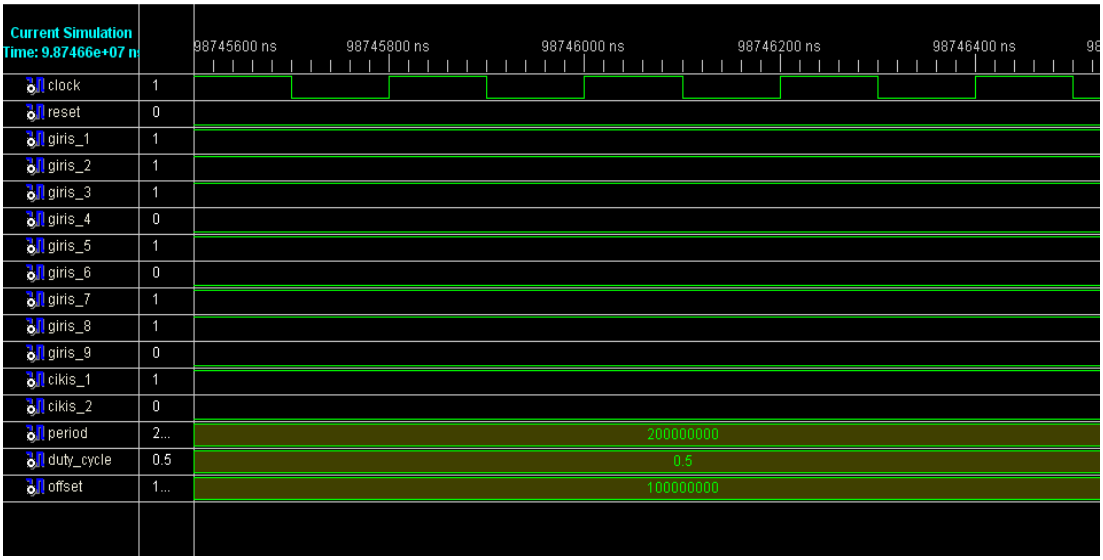
-Son durum için elde edilen test çıkışı değerleri uyarınca çıkış bitleri ayarlanmıştır.

5.4 VHDL Kod Benzetimi ve Sonuçları

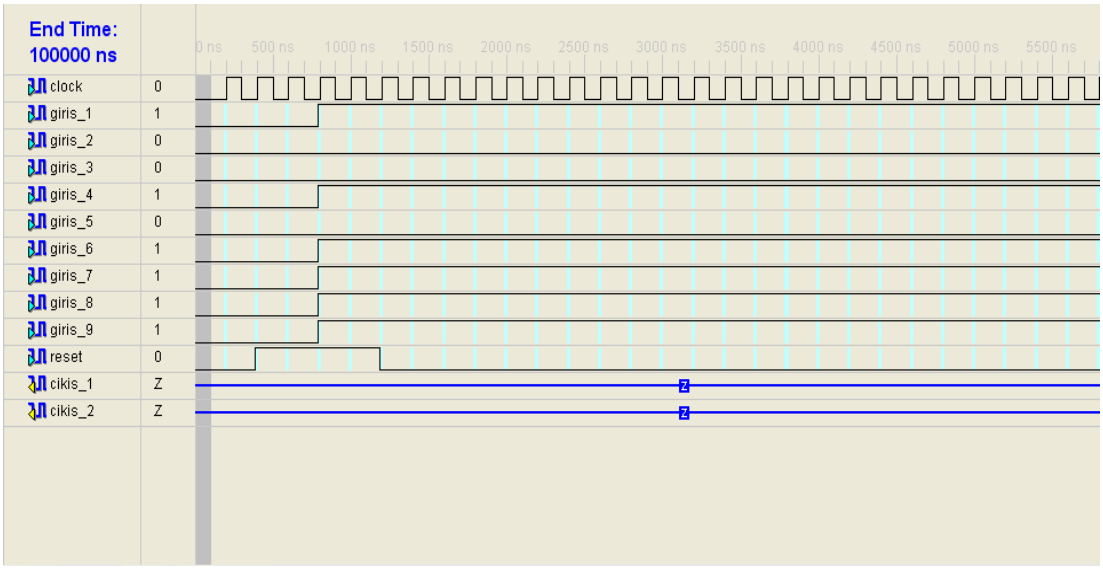
XILINX ISE 10.1 ortamında Kodun benzetimi için; giriş değerlerinin, benzetim süresinin ve saat ile reset durumunun belirtilmesi için bir test ortamı oluşturulmasına ihtiyaç vardır. Çok katmanlı algılayıcı yapımızın T ve L harfi testleri için 2 farklı test ortamı oluşturulmuştur. T harfinin testi için oluşturulan test ortamı Şekil 5.1’de, benzetim sonucu Şekil 5.2’de, L harfinin testi için oluşturulan test ortamı Şekil 5.3’de ve benzetim sonucu Şekil 5.4’te gösterilmektedir.



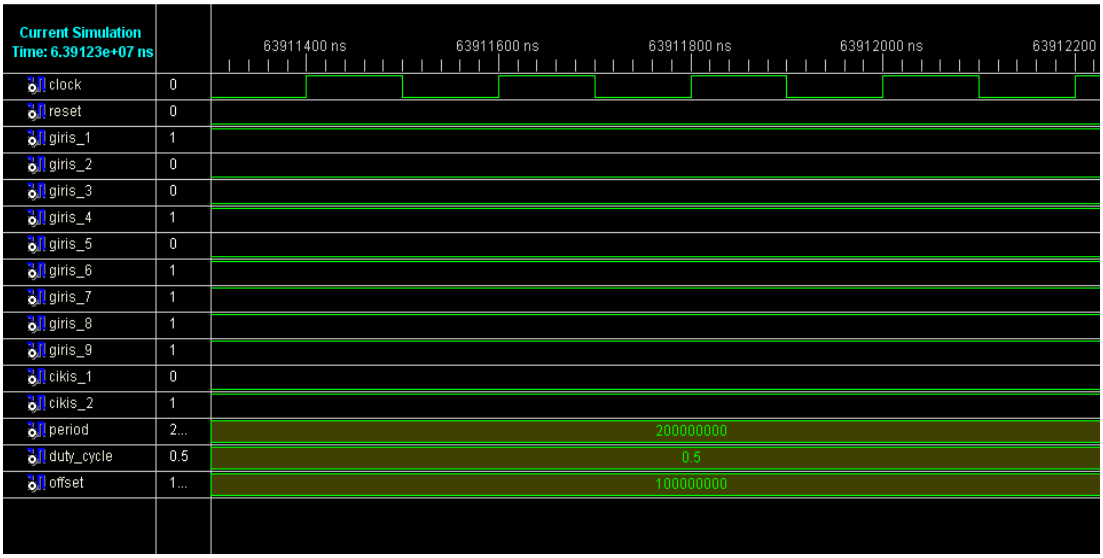
Şekil 5.1 : T harfi test ortamı



Şekil 5.2 : T harfi benzetim sonucu



Şekil 5.3 : L harfi test ortamı



Şekil 5.4 : L harfi benzetim sonucu

Benzetim sonuçlarından görüldüğü üzere, T harfinin testi çıkışı için öngörülen [1,0] değeri ve L harfinin test çıkışı için öngörülen [0,1] değeri elde edilmiştir. Çok katmanlı algılayıcı yapısının VHDL ortamında tasarımı ve benzetimi başarılıdır.

6. TASARIMIN VIRTEX-5 FPGA ÜZERİNDE GERÇEKLENMESİ

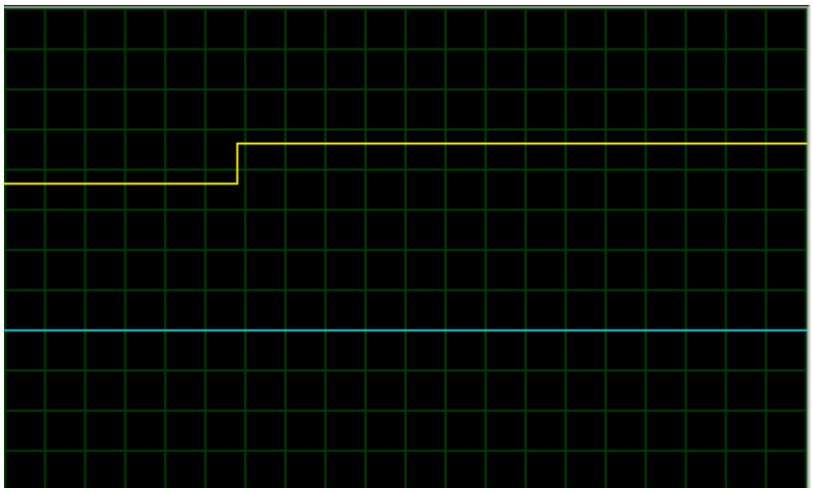
XILINX ISE 10.1 ortamında benzetimi tamamlanan tasarımın VIRTEX-5 FPGA yongasına gönderilmesi için, tasarımda belirtilen giriş ve çıkış kapılarının tasarım kartında nereye bağlanacağına karar verilmesi gerekir. Bu tasarımda saat girişi kartın kendi üzerinde bulunan 100 MHz frekansında darbe üreten saat pinine bağlanmıştır. 9 adet test girişinden 1-8 arası girişler 8 adet anahtara, 9. giriş de butona bağlanmıştır. Bunun sebebi tasarım kartı üzerinde 8 adet kontrol edilebilir anahtar bulunmasıdır. Reset girişi de diğer bir butona bağlanmıştır. 2 adet olan çıkış da istenirse LED'lerden gözlenebilir, istenirse tasarım kartının dışarıyla bağlantı noktalarına verilerek çıkış sinyalleri osiloskoptan izlenebilir. Bu tasarımda sinyallerin daha rahat incelenebilmesi için 2. yöntem tercih edilmiştir. Bu bağlantıların belirtilmesi için pin bağlantı (UCF) dosyası oluşturulması gerekmektedir. Tasarım kartının bilgi dosyasından elde edilen bağlantı numaralarına göre oluşturulan dosya içeriği aşağıdadır [8]:

```
NET "giris_1" LOC = U4 | PULLUP;

NET "giris_2" LOC = V3 | PULLUP;

NET "giris_3" LOC = T4 | PULLUP;

NET "giris_4" LOC = T5 | PULLUP;

NET "giris_5" LOC = U6 | PULLUP;

NET "giris_6" LOC = U5 | PULLUP;

NET "giris_7" LOC = U7 | PULLUP;

NET "giris_8" LOC = T7 | PULLUP;

NET "clock" LOC = AD8;

NET "cikis_1" LOC = H33;

NET "cikis_2" LOC = AN33;
```

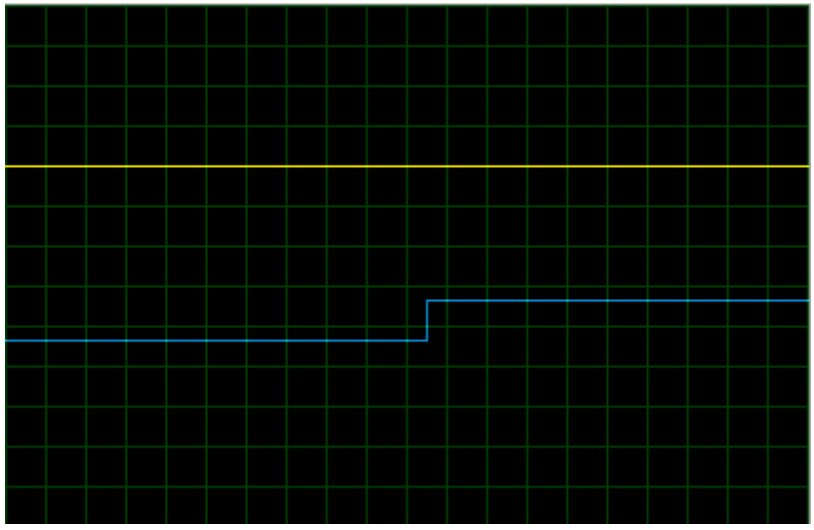
NET "giris_9" LOC = B22 ;

NET "reset" LOC = B21;

Pin bağlantı dosyası oluşturulduktan sonra programlama dosyası (BIT file) oluşturma aşamasına geçilmiştir. XILINX ISE 10.1'in otomatik olarak oluşturduğu programlama dosyası bağlantı kablosu üzerinden VIRTEX-5 FPGA'e gönderilmiştir. Gerçekleme sonucu, osiloskopta gözlenmiştir. T harfi için çıkış Şekil 6.1'deki gibi [1,0], L harfi için çıkış Şekil 6.2'deki gibi [0,1] olmaktadır. Gerçekleme sonucu, benzetim sonucuyla birebir örtüşmektedir.



Şekil 6.1 : T harfi için çıkışın osiloskoptaki görüntüsü



Şekil 6.2 : L harfi için çıkışın osiloskoptaki görüntüsü

7. SONUÇ

Günümüzde, üzerinde yapılan geliştirme çalışmalarına paralel hızda artan uygulamalarıyla gömülü sistem tasarımının en önemli aşaması haline gelen FPGA ile tasarım, elektronik sistem tasarımı ve araştırma-geliştirme alanlarında çok daha büyük uygulama alanlarına ve çok daha geniş çaplı tasarımlara açıktır. Bu çalışmada, günümüzde bilgisayarlar ve işlemciler ile uygulamaları birçok alanda yapılan yapay sinir ağının özel bir türü olan çok katmanlı algılayıcı algoritması, FPGA ile gerçeklenmiştir. FPGA'ler üzerinde yapılan geliştirmeler hız kazandıkça, çok sayıda girişli ve çok sayıda katmanlı karmaşık sinir ağı yapılarının FPGA üzerinde geliştirilmesi daha kolay ve hızlı biçimde mümkün olacaktır.

KAYNAKLAR

[1] Haykin, S., "Neural Networks – A Comprehensive Foundation", Prentice-Hall, Hindistan, 2005.

[2] Url-1, <wikipedia.org**>** "fpga" alındığı tarih : 01.05.2009.

[3] Url-2 <www.elektrotekno.com**>** "fpga çeviri" alındığı tarih : 15.12.2008.

[4]. **Taright, Y, Hubin, M,** "FPGA Implementation of a Multilayer Perceptron Neural Network using VHDL" , Proceedings of ICSP '98 Seminer Dökümanı, Fransa, 1998.

[5] Ortigosa, E.M., Ortigosa, P.M., Cañas, A., Ros, E., Agís, R. , ve Ortega J. , "FPGA Implementation of Multi-layer Perceptrons for Speech Recognition", İspanya.

[6] Ortigosa, E.M., Ortigosa, P.M., Cañas, A., Ros, E., Mota S., Diaz J., "Hardware Description of Multi-layer Perceptrons with Different Abstraction Levels", İspanya, 2006.

[7] Pedroni, V.A., "Circuit Design with VHDL" MIT Press, İngiltere, 2004.

[8] Xilinx, "Virtex-5 ML506 Evaluation Platform User Guide", v3.1, 2008.

EK A

ÇOK KATMANLI ALGILAYICI YAPISI İÇİN TASARLANMIŞ VHDL KODU

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    10:23:30 03/13/2009
-- Design Name:
-- Module Name:    bitirme_source - bitirme_Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    14:26:35 03/31/2009
-- Design Name:
-- Module Name:    ozgur_ozden - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.lut_package.all;


---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity bitirme_source is
    Port ( clock : in STD_LOGIC;
           reset : in STD_LOGIC;
           giris_1 : in  STD_LOGIC;
           giris_2 : in  STD_LOGIC;
           giris_3 : in  STD_LOGIC;
           giris_4 : in  STD_LOGIC;
```

```
        giris_5 : in  STD_LOGIC;
        giris_6 : in  STD_LOGIC;
        giris_7 : in  STD_LOGIC;
        giris_8 : in  STD_LOGIC;
        giris_9 : in  STD_LOGIC;
        cikis_1 : out  STD_LOGIC;
        cikis_2 : out  STD_LOGIC);
end bitirme_source;

architecture bitirme_Behavioral of bitirme_source is

TYPE state IS (state0, state1, state2, state3, state4, state5, wait_st);
SIGNAL pr_state, nx_state: state;

signal test_giris_1: integer;
signal test_giris_2: integer;
signal test_giris_3: integer;
signal test_giris_4: integer;
signal test_giris_5: integer;
signal test_giris_6: integer;
signal test_giris_7: integer;
signal test_giris_8: integer;
signal test_giris_9: integer;

signal v1_1: integer;
signal v1_2: integer;
signal v1_3: integer;
signal v1_4: integer;

signal y1_1: integer;
signal y1_2: integer;
signal y1_3: integer;
signal y1_4: integer;

signal v2_1: integer;
signal v2_2: integer;
signal v2_3: integer;

signal y2_1: integer;
signal y2_2: integer;
signal y2_3: integer;

signal vo_1: integer;
signal vo_2: integer;

signal yo_1: integer;
signal yo_2: integer;

--agirliklar--

constant agirlik_1_11:  integer := -14;
constant agirlik_1_12:  integer := -9;
constant agirlik_1_13:  integer := 9;
constant agirlik_1_14:  integer := -48;
constant agirlik_1_15:  integer := 76;
constant agirlik_1_16:  integer := 16;
constant agirlik_1_17:  integer := -63;
constant agirlik_1_18:  integer := 16;
constant agirlik_1_19:  integer := -11;
constant agirlik_1_110: integer := 5;
constant agirlik_1_21:  integer := -45;
constant agirlik_1_22:  integer := 46;
constant agirlik_1_23:  integer := 24;
constant agirlik_1_24:  integer := -47;
constant agirlik_1_25:  integer := 60;
constant agirlik_1_26:  integer := 17;
constant agirlik_1_27:  integer := -25;
constant agirlik_1_28:  integer := 8;
constant agirlik_1_29:  integer := -39;
constant agirlik_1_210: integer := 1;    --0.66 1 alindi--
constant agirlik_1_31:  integer := -19;
constant agirlik_1_32:  integer := 19;
constant agirlik_1_33:  integer := 25;
```

```
constant agirlik_1_34:  integer := 19;
constant agirlik_1_35:  integer := -28;
constant agirlik_1_36:  integer := -42;
constant agirlik_1_37:  integer := -21;
constant agirlik_1_38:  integer := -1;
constant agirlik_1_39:  integer := -39;
constant agirlik_1_310: integer := -12;
constant agirlik_1_41:  integer := 1;
constant agirlik_1_42:  integer := 25;
constant agirlik_1_43:  integer := 42;
constant agirlik_1_44:  integer := -33;
constant agirlik_1_45:  integer := 39;
constant agirlik_1_46:  integer := 37;
constant agirlik_1_47:  integer := -18;
constant agirlik_1_48:  integer := -29;
constant agirlik_1_49:  integer := -31;
constant agirlik_1_410: integer := -6;



constant agirlik_2_11:  integer := 23;
constant agirlik_2_12:  integer := 59;
constant agirlik_2_13:  integer := 2;
constant agirlik_2_14:  integer := 26;
constant agirlik_2_15:  integer := -23;
constant agirlik_2_21:  integer := -58;
constant agirlik_2_22:  integer := -101;
constant agirlik_2_23:  integer := 7;
constant agirlik_2_24:  integer := -35;
constant agirlik_2_25:  integer := 73;
constant agirlik_2_31:  integer := 46;
constant agirlik_2_32:  integer := 34;
constant agirlik_2_33:  integer := -10;
constant agirlik_2_34:  integer := 45;
constant agirlik_2_35:  integer := -39;



constant agirlik_o_11: integer := 77;
constant agirlik_o_12: integer := -117;
constant agirlik_o_13: integer := 33;
constant agirlik_o_14: integer := -13;
constant agirlik_o_21: integer := -27;
constant agirlik_o_22: integer := 106;
constant agirlik_o_23: integer := -81;
constant agirlik_o_24: integer := 14;



begin

process(clock,reset)

begin

if(reset='1') then
pr_state <= state0;
--
--test_giris_1 <=0;
--test_giris_2 <=0;
--test_giris_3 <=0;
--test_giris_4 <=0;
--test_giris_5 <=0;
--test_giris_6 <=0;
--test_giris_7 <=0;
--test_giris_8 <=0;
--test_giris_9 <=0;
----
--v1_1 <=0;
--v1_2 <=0;
--v1_3 <=0;
```

```
--v1_4 <=0;
--
--y1_1 <=0;
--y1_2 <=0;
--y1_3 <=0;
--y1_4 <=0;
--
--v2_1 <=0;
--v2_2 <=0;
--v2_3 <=0;
--
--y2_1 <=0;
--y2_2 <=0;
--y2_3 <=0;
--
--vo_1 <=0;
--vo_2 <=0;
--
--yo_1 <=0;
--yo_2 <=0;

ELSIF (clock'EVENT AND clock='1') THEN
pr_state <= nx_state;



END IF;

END PROCESS;

process(giris_1,giris_2,giris_3,giris_4,giris_5,giris_6,giris_7,giris_8,giris_9,reset,pr_state)
BEGIN


--test girisi atamalari--
if(giris_1='1') then

test_giris_1 <= 90;

else

test_giris_1 <= 10;

end if;
-------------------------------
if(giris_2='1') then

test_giris_2 <= 90;

else

test_giris_2 <= 10;

end if;
-------------------------------
if(giris_3='1') then

test_giris_3 <= 90;

else

test_giris_3 <= 10;

end if;
---------------------------------
if(giris_4='1') then

test_giris_4 <= 90;

else

test_giris_4 <= 10;
```

```
end if;
-----------------------------------
if(giris_5='1') then

test_giris_5 <= 90;

else

test_giris_5 <= 10;

end if;
-----------------------------------
if(giris_6='1') then

test_giris_6 <= 90;

else

test_giris_6 <= 10;

end if;
-----------------------------------
if(giris_7='1') then

test_giris_7 <= 90;

else

test_giris_7 <= 10;

end if;
-----------------------------------
if(giris_8='1') then

test_giris_8 <= 90;

else

test_giris_8 <= 10;

end if;
-----------------------------------
if(giris_9='1') then

test_giris_9 <= 90;

else

test_giris_9 <= 10;

end if;

---------------------------------------

nx_state <= pr_state;

CASE pr_state IS


--1. katman v'lerin hesabi--

WHEN state0 =>

v1_1 <= agirlik_1_11 * test_giris_1 + agirlik_1_12 * test_giris_2 + agirlik_1_13 * test_giris_3 + agirlik_1_14 * test_giris_4 +
agirlik_1_15 * test_giris_5 + agirlik_1_16 * test_giris_6 + agirlik_1_17 * test_giris_7 + agirlik_1_18 * test_giris_8 +
agirlik_1_19 * test_giris_9 + agirlik_1_110 * 100;
v1_2 <= agirlik_1_21 * test_giris_1 + agirlik_1_22 * test_giris_2 + agirlik_1_23 * test_giris_3 + agirlik_1_24 * test_giris_4 +
agirlik_1_25 * test_giris_5 + agirlik_1_26 * test_giris_6 + agirlik_1_27 * test_giris_7 + agirlik_1_28 * test_giris_8 +
agirlik_1_29 * test_giris_9 + agirlik_1_210 * 100;
v1_3 <= agirlik_1_31 * test_giris_1 + agirlik_1_32 * test_giris_2 + agirlik_1_33 * test_giris_3 + agirlik_1_34 * test_giris_4 +
agirlik_1_35 * test_giris_5 + agirlik_1_36 * test_giris_6 + agirlik_1_37 * test_giris_7 + agirlik_1_38 * test_giris_8 +
agirlik_1_39 * test_giris_9 + agirlik_1_310 * 100;
```

```
v1_4 <= agirlik_1_41 * test_giris_1 + agirlik_1_42 * test_giris_2 + agirlik_1_43 * test_giris_3 + agirlik_1_44 * test_giris_4 +
agirlik_1_45 * test_giris_5 + agirlik_1_46 * test_giris_6 + agirlik_1_47 * test_giris_7 + agirlik_1_48 * test_giris_8 +
agirlik_1_49 * test_giris_9 + agirlik_1_410 * 100;


--end if;

nx_state <= state1;

----1. katman v'lerin hesabi sonu--
--
----1. katman egitim fonksiyonu--
--
when state1 =>

----look up table------------

y1_1 <= lut(v1_1);
y1_2 <= lut(v1_2);
y1_3 <= lut(v1_3);
y1_4 <= lut(v1_4);


---------look up table sonu------

nx_state <= state2;

----1. katman egitim fonksiyonu sonu----

when state2 =>

--2. katman v'lerin hesabi----

v2_1 <= agirlik_2_11 * y1_1 + agirlik_2_12 * y1_2 + agirlik_2_13 * y1_3 + agirlik_2_14 * y1_4 + agirlik_2_15 * 100;
v2_2 <= agirlik_2_21 * y1_1 + agirlik_2_22 * y1_2 + agirlik_2_23 * y1_3 + agirlik_2_24 * y1_4 + agirlik_2_25 * 100;
v2_3 <= agirlik_2_31 * y1_1 + agirlik_2_32 * y1_2 + agirlik_2_33 * y1_3 + agirlik_2_34 * y1_4 + agirlik_2_35 * 100;
--
----2. katman v'lerin hesabi sonu--
--
----2. katman egitim fonksiyonu--
nx_state <= state5;
-----look up table-------
when state5 =>

y2_1 <= lut(v2_1);
y2_2 <= lut(v2_2);
y2_3 <= lut(v2_3);

--------look up table sonu----

nx_state <= state3;

-----2. katman egitim fonksiyonu sonu----

when state3 =>

---cikis katmani v'lerin hesabi-----

vo_1 <= agirlik_o_11 * y2_1 + agirlik_o_12 * y2_2 + agirlik_o_13 * y2_3 + agirlik_o_14 * 100;
vo_2 <= agirlik_o_21 * y2_1 + agirlik_o_22 * y2_2 + agirlik_o_23 * y2_3 + agirlik_o_24 * 100;
--
----cikis katmani v'lerin hesabi sonu--

nx_state <= state4;

----cikis katmani egitim fonksiyonu--

when state4 =>

-----look up table-------

yo_1 <= lut(vo_1);
yo_2 <= lut(vo_2);
```

```
--------look up table sonu----
nx_state <= wait_st;

--cikis katmani egitim fonksiyonu sonu--

when wait_st =>

nx_state <= wait_st;

when others => NULL;

end case;



end process;

process(yo_1, yo_2, pr_state)
begin


case pr_state is

when state0 =>
 cikis_1 <= '0';
 cikis_2 <= '0';

when state1 =>
 cikis_1 <= '0';
 cikis_2 <= '0';
when state2 =>
 cikis_1 <= '0';
 cikis_2 <= '0';
when state3 =>
 cikis_1 <= '0';
 cikis_2 <= '0';
when state5 =>
 cikis_1 <= '0';
 cikis_2 <= '0';

when state4 =>

 cikis_1 <= '0';
 cikis_2 <= '0';

when wait_st =>
if (yo_1 >= 70) then

cikis_1 <= '1';

elsif (yo_1 <= 30) then

cikis_1 <= '0';

end if;

if (yo_2 >= 70) then

cikis_2 <= '1';

elsif (yo_2 <= 30) then

cikis_2 <= '0';

end if;

when others => NULL;

end case;
end process;
end bitirme_Behavioral;
```

ÖZGEÇMİŞ

Özgür Özden, Muğla'nın Milas ilçesinde, 01.04.1987 tarihinde dünyaya geldi. İlköğretimini Zühtüpaşa İlköğretim Okulu ve Kazım Karabekir ilköğretim Okulu'nda, lise öğrenimini ise İstanbul Atatürk Fen Lisesi'nde tamamladıktan sonra, 2004 yılında İstanbul Teknik Üniversitesi Elektronik Mühendisliği Bölümün'de öğrenim görmeye hak kazandı. Haziran 2009 yılında bölümünden mezun oldu.