DÜZCE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

YÜKSEK LİSANS TEZİ

YAPAY SİNİR AĞLARININ OTOMATİK OLARAK FPGA'YA UYGULANMASI İÇİN VERİ YOLU TASARIM ARACI

NAMIK KEMAL SARITEKİN

ELEKTRİK EĞİTİMİ ANABİLİM DALI

MAYIS 2011
DÜZCE

DÜZCE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

YÜKSEK LİSANS TEZİ

YAPAY SİNİR AĞLARININ OTOMATİK OLARAK FPGA'YA UYGULANMASI İÇİN VERİ YOLU TASARIM ARACI

NAMIK KEMAL SARITEKİN

ELEKTRİK EĞİTİMİ ANABİLİM DALI

MAYIS 2011
DÜZCE

Namık Kemal SARITEKİN tarafından hazırlanan **"**Yapay Sinir Ağlarının Otomatik Olarak FPGA'ya Uygulanması İçin Veri Yolu Tasarım Aracı" adlı bu tezin Yüksek Lisans tezi olarak uygun olduğunu onaylarım.

Yrd. Doç. Dr. İbrahim ŞAHİN : ……...
Tez Danışmanı
Bilgisayar Eğitimi Anabilim Dalı

Bu çalışma, jürimiz tarafından oy birliği ile Elektrik Eğitimi Anabilim Dalında Yüksek Lisans tezi olarak kabul edilmiştir.

Yrd. Doç. Dr. Bilal SARAÇOĞLU : ……...
Elektrik Eğitimi Anabilim Dalı Başkanı
Düzce Üniversitesi

Doç. Dr. Serkan SUBAŞI : ……...
Teknoloji Fakültesi İnşaat Mühendisliği
Düzce Üniversitesi

Yrd. Doç. Dr. İbrahim ŞAHİN : ……..
Tez Danışmanı
Bilgisayar Eğitimi Anabilim Dalı

Tarih: 27/05/2011

Bu tez ile Düzce Üniversitesi Fen Bilimleri Enstitüsü Yönetim Kurulu Yüksek Lisans derecesini onamıştır.

Prof. Dr. Refik KARAGÜL : ……...
Fen Bilimleri Enstitüsü Müdürü

TEZ BİLDİRİMİ

Tez içindeki bütün bilgilerin etik davranış ve akademik kurallar çerçevesinde elde edilerek sunulduğunu, ayrıca tez yazım kurallarına uygun olarak hazırlanan bu çalışmada bana ait olmayan her türlü ifade ve bilginin kaynağına eksiksiz atıf yapıldığını bildiririm.

Namık Kemal SARITEKİN

ÖNSÖZ

Tez danışmanlığımı üstlenerek araştırma konusunun seçimi, yürütülmesi ve sunuma hazırlanması sırasında, değerli bilimsel görüş ve önerilerinden yararlandığım Yrd. Doç. Dr. İbrahim ŞAHİN'e teşekkür eder ve en içten minnet duygularımı sunarım.

Değerli görüşleriyle bana yol gösteren ve yapıcı eleştirileriyle yardımlarını esirgemeyen Yrd. Doç. Dr. Bilal SARAÇOĞLU ve Doç. Dr. Serkan SUBAŞI'na teşekkürü borç bilirim.

Tezin hazırlanmasında ilgileri, sabırları ve maddi manevi destekleri için Sn. Süleyman ÇAKICI, Sn. Metin TOZ'a ve aileme teşekkürlerimi sunarım.

Mayıs 2011 **Namık Kemal SARITEKİN**

İÇİNDEKİLER **Sayfa**

ŞEKİL LİSTESİ

Sayfa

ÇİZELGE LİSTESİ

Sayfa

SEMBOL LİSTESİ

$\mathbf{a_i}$	: Giriş katmanı çıkışı
$\mathbf{B_s}$	: Eşik değeri
h	: Katman türü sayısı
j	: Katman türü içindeki element türü sayısı
k	: Katman türü içindeki element türü içindeki katman sayısı
m	: Katman sayısı
$\mathbf{n_i}$	: Giriş değerleri ile ağırlık değerlerinin çarpımlarının toplamı
p	: FPGA ve hafıza sayısı
r	: Sinir hücresi sayısı
$\mathbf{X_i}$	: Girişler
$\mathbf{W_{i,s}}$	: Ağırlık değerleri
$\mathbf{y_i}$	: Çıkış değeri
z	: Çıkış sayısı
α	: Sabit katsayı

KISALTMA LİSTESİ

ABEL : The Advanced Boolean Expression Language (Gelişmiş Mantıksal Açıklama Dili)
AHDL : Altera Hardware Description Language (Altera Donanım Tanımlama Dili)
AHPL : A Hardware Programming Language (Donanım Tanımlama Dili)
ANN : Artificial Neural Networks (Yapay Sinir Ağları)
ANNGEN : Artificial Neural Network GENerator (Yapay Sinir Ağlarının VHDL Kodunu Oluşturucu)
ASIC : Application Specific Integrated Circuits (Uygulamaya Özel Tümdevre)
B : Bias (Eşik) Değeri
BM : Bulanık Mantık
CDL : Computer Design Language (Bilgisayar Tasarım Dili)
CLB : Configurable Logic Blocks (Yapılandırılabilir Mantıksal Bloklar)
Clk : Clock (Zaman)
COMP : Competitive (Rekabetçi)
CONLAN : CONsensus LANguage (Konsensüs Dili)
CPLD : Complex Programmable Logic Device (Karmaşık Programlanabilir Mantık Aygıtı)
CPU : Central Processing Unit (Merkezi İşlem Birimi)
ÇKA : Çok Katmanlı Algılayıcı
DMA : Direct Memory Access (Doğrudan Hafıza Erişimi)
DSP : Digital Signal Processing (Sayısal Sinyal İşleme)
EDA : Electronic Design Automation (Otomatik Elektronik Tasarım Aracı)
EKG : Elektro Kardiyo Gram
ESA : European Space Agency (Avrupa Uzay Ajansı)
F(x) : Transfer Fonksiyonu
F-CCM : FPGA-based Custom Computing Machine (FPGA tabanlı Özel Amaçlı Bilgisayarlar)
FFT : Fast Fourier Transform (Hızlı Fourier Dönüşümü)
FpAdd8 : Floating Point Adder (Kayan Noktalı Toplayıcı)
FPGA : Field Programmable Gate Array (Alanda Programlanabilir Kapı Dizileri)
FpMult8 : Floating Point Multiplier (Kayan Noktalı Çarpıcı)
GA : Genetic Algorithm (Genetik Algoritma)
GPS : Global Positioning System (Küresel Konumlandırma Sistemi)
GY : Geriye Yayılım
HAL-15 : Hardware Abstraction Layer-15 (Donanım Soyutlama Katmanı-15-Süper Bilgisayar)
HDL : Hardware Description Language (Donanım Tanımlama Dili)
HLIM : Hard Limit (Sabit Limit)
HLMS : Symmetrical Hard Limit (Simetrik Sabit Limit)

IDL : Interactive Design Language (Etkileşimli Tasarım Dili)
IEEE : Institute of Electrical and Electronics Engineering (Elektrik ve Elektronik Mühendisliği Enstitüsü)
IOB : Input/Output Blocks (Giriş Çıkış Blokları)
IP : Intellectual Property (Fikri Mülkiyet)
ISE : Integrated Software Environment (Entegre Yazılım Ortamı)
ISPS : Instruction Set Processor Specification (İşlemci Özelliklerinin Kurulum Talimatları)
LOGS : Log- Sigmoid (Logaritmik Sigmoid)
LUT : Look-Up Table (Bakılan Tablo)
NASA : National Aeronautics and Space Administration (Amerikan Ulusal Havacılık ve Uzay Araştırmaları Merkezi)
PCI : Peripheral Component Interconnect (Çevre Birimleri Bağlantı Yolu)
PLIN : Pure Linear (Doğrusal)
RAM : Random Access Memory (Rastgele Erişimli Hafıza Türü)
RC : Reconfigurable Computing (Yeniden Yapılandırılabilir Hesaplama)
RTL : Register Transfer Level description (Transfer Seviye Kayıt Tanımlaması)
SnO_2 : Kalay Oksit
SoC : System on a Chip (Yonga Üzerinde Sistem)
SOM : Self Organizing Maps (Kendini Ayarlayan Haritalı Ağlar)
SPLD : Simple Programmable Logic Device (Basit Programlanabilir Mantık Devresi)
TANS : Hyperbolic Tangent Sigmoid (Hiperbolik Tanjant Sigmoid)
TBA : Temel Bileşen Analizi
TEGAS : TEst Generation And Simulation (Test Üretimi ve Benzetimi)
TI-HDL : Texas Instruments Hardware Description Language (Texas Aletlerinin Donanım Tanımlama Dili)
US : Uzman Sistemler
VHDL : Very High Speed Integrated Circuit HDL (Çok Hızlı Entegre Devre Donanım Tanımlama Dili)
VLSI : Very Large Scale Integration (Çok Büyük Ölçekli Entegrasyon)
YSA : Yapay Sinir Ağı, Yapay Sinir Ağları
YZ : Yapay Zekâ

EKLER LİSTESİ

YAPAY SİNİR AĞLARININ OTOMATİK OLARAK FPGA'YA UYGULANMASI İÇİN VERİ YOLU TASARIM ARACI
(Yüksek Lisans Tezi)

Namık Kemal Sarıtekin

DÜZCE ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

Mayıs 2011

ÖZ

Yapay Sinir Ağları (YSA) günümüzde değişik alanlarda çok yaygın bir şekilde kullanılmaktadır. YSA'lar yazılım olarak gerçeklenebildiği gibi yazılım yetersiz kaldığı durumlarda donanım olarak da gerçeklenmektedir. Donanım ortamı olarak Alanda Programlanabilir Kapı Dizileri [Field Programmable Gate Arrays (FPGA)] ucuz olmaları, esnek olmaları, ilk üretimlerinin hızlı olması ve ölçeklendirilebilir olmaları gibi özelliklerinden dolayı tercih edilmektedirler. Bu çalışmada, yapay sinir ağlarının FPGA'ya uygulanmasının otomatikleştirilmesi, bu işlem için uzman gereksiniminin azaltılması ve uygulama sürecinin kısaltılması amacıyla, YSA'lar için otomatik veri yolu tasarımı yapabilen bir araç [Yapay Sinir Ağlarının VHDL Kodunu Oluşturucu - Artificial Neural Network GENerator (ANNGEN)] geliştirilmiştir. Bu kapsamda öncelikle ANNGEN tarafından kullanılan ve yapay sinir hücrelerinden oluşan örnek bir yapay sinir hücresi kütüphanesi oluşturulmuştur. Kütüphanede hâlihazırda altı değişik sinir hücresi bulunmaktadır. İstenildiğinde kütüphaneye yeni hücreler eklenebilmekte ve bunlar ANNGEN tarafından otomatik olarak tanınmaktadır. ANNGEN girdi olarak oluşturulmak istenen yapay sinir ağının metin tabanlı tanımlamasını, sinir hücresi kütüphanesini ve şablon dosyasını alır. FPGA yongalarına uygulanabilecek formatta tasarlanmak istenen yapay sinir ağı için gerekli veri yolu tasarımını yapar ve VHDL [Very High Speed Integrated Circuit HDL (Çok Hızlı Entegre Devre Donanım Tanımlama Dili)] kodunu üretir.

Tez kapsamında ANNGEN'i test etmek amacıyla iki test durumu oluşturulmuştur. ANNGEN bu test durumları ile test edilmiş ve başarılı bir şekilde oluşturulmak istenen YSA'lar için VHDL kodunu ürettiği gözlenmiştir. Üretilen VHDL kodlarının doğruluğunu kontrol etmek amacıyla bir EDA [Electronic Design Automation (Otomatik Elektronik Tasarım Aracı)] aracı olan Xilinx'in ISE

[Integrated Software Environment (Entegre Yazılım Ortamı)] kullanılmıştır. Üretilen kodlar ISE aracı ile önce yazım kontrolünden geçirilmiş ardında da sentezlenerek RTL [Register Transfer Level description (Transfer Seviye Kayıt Tanımlaması)] şeması başarılı bir şekilde oluşturulmuştur.

Bu çalışmanın sonuçları göstermiştir ki metin tabanlı YSA tanımlaması verildiğinde ANNGEN sayesinde istenen YSA için veri yolu saniyeler içinde otomatik olarak tasarlanmakta ve VHDL kodu üretilebilmektedir. Dolayısıyla donanım tabanlı YSA tasarım süreci çok kısaltılmış ve bu işlem için uzman personel gereksinimi ortadan kaldırılmıştır.

Bilim Kodu : 401915
Anahtar Kelimeler : ANNGEN, FPGA, VHDL, YSA.
Sayfa Adedi : 76
Tez Yöneticisi : Yrd. Doç. Dr. İbrahim ŞAHİN

A DATA PATHS DESIGN TOOL FOR AUTOMATICALLY MAPPING ARTIFICIAL NEURAL NETWORK ON TO FPGAs
(Master of Science Thesis)

Namık Kemal Sarıtekin

DUZCE UNIVERSITY

INSTITUTE OF SCIENCE AND TECNOLOGY

May 2011

ABSTRACT

Nowadays, Artificial Neural Networks (ANN) is a widely used in different applications. ANN can be produced as software. Although, ANN can be implemented as software, in cases where software implementations are not sufficient in terms of performance, ANN can be implemented as hardware. In hardware implementations, Field Programmable Gate Arrays (FPGAs) are preferred as cheap, flexible, scalable and faster first manufacturing alternative compared to the other hardware implementation techniques.

In this study, a data paths design tool [Artificial Neural Network GENerator (ANNGEN)] was developed to help automate the application of ANNs to FPGAs, to reduce the design and implementation time and to minimize the expert requirements while mapping ANNs to FPGAs. With these goals in mind, first an artificial neuron library was developed. Currently, the library contains designs for six different neurons. As new neurons are designed, they can easily be added to the library and these new neurons are automatically recognized by ANNGEN and used if needed. ANNGEN takes three inputs which are text based definition of the ANN (NetList) to be mapped, a library and a template file. It, first, designs a data path for the given NetList, and then, produces a VHDL (Very High Speed Integrated Circuit Hardware Description Language) code for the design.

Two different test cases were formed to test ANNGEN. It was tested with these test cases and it is observed that ANNGEN was able to successfully made designs and produced VHDL codes for the given test cases. Xilinx ISE (Integrated Software Environment) tool was used to verify correctness of the design and VHDL code produced by ANNGEN. First, Syntax checks of the VHDL codes were done. Then, the codes were synthesized and RTL (Register Transfer Level description) schematics were formed successfully.

The results of this study showed that when a Net-List of an ANN is given to ANNGEN, it can easily design a data path for the given Net-List and produce VHDL code automatically in seconds. As a result, it reduced the time needed for ANN data path design and VHDL coding dramatically, and eliminated the expert requirement.

Science Code : 401915
Key Words : ANNGEN, FPGA, VHDL, Neural Networks.
Page Number : 76
Adviser : Assist. Prof. Dr. İbrahim ŞAHİN

1. GİRİŞ

İnsanlar her zaman geleceği merak etmiş, meydana gelecek olayları önceden haber almak için birçok yöntemlere başvurmuşlardır. Bazen yıldızlardan, bazen doğa olaylarından geleceği tahmin etmeye çalışmışlardır. Zamanın ilerlemesi ile geçmişteki yöntemlerin yerlerini matematiksel işlemler ve bilgisayarlarla hesaplamalar almıştır. İnsanoğlu doğadan ilham alarak değişik modeller ve modellemeler geliştirmişlerdir.

Yapay Zekânın (Artificial Intelligence) bir uygulaması olan Yapay Sinir Ağları [YSA – Artificial Neural Network (ANN)] son yıllarda geleceği öngörmede, örüntü tanımada, verilerin yorumlanmasında, optimizasyon işlemlerinde, doğrusal (lineer) olmayan (non-lineer) fonksiyon yaklaşımlarında ve birçok değişkene bağımlı zaman serilerinin tahmin edilmesinde önemli bir araç olmuştur. Özellikle mühendislik alanında kolaylıkla uygulanabilmeleri nedeniyle YSA'lara ilginin her geçen gün arttığı görülmektedir.

Çağımızda teknolojik gelişmeler, özellikle bilgisayar dünyasında, çok hızlı olduğundan takip etmekte zorlanılmaktadır. Kısa zamanda yapılması gereken hesaplamaların hızlı yapılması, bilgiye erişimin hızlanması, en küçük (mikro) ve en büyük (makro) âlemlerin incelenmesi v.b. konularda bilgisayarlar kolaylıklar sağlamışlardır. Bütün bunlara rağmen beynin bilgisayardan daha verimli çalışmasının nedeni uzun yıllar merak konusu olmuş ve bilim adamlarını beynin çalışma sistemini incelemeye yönlendirmiştir.

Yapılan incelemeler sonucunda insan beyni ile bilgisayarlar arasındaki farklılığın bilgilerin işlenme şeklinden kaynaklandığı ortaya çıkmıştır. Örneğin, insan beynindeki sinirler bilgileri paralel olarak işlemektedir. Bu gerçekten yola çıkılarak yapılan çalışmalarda bilgilerin paralel işlenerek çalıştığı modeller, araçlar ve makineler tasarlanmıştır. Sonunda insan davranışlarını modelleyebilecek akıllı sistemlerin kurulması için YSA, Bulanık Mantık (BM), Uzman Sistemler (US) ve Genetik Algoritma (GA) gibi yöntemler geliştirilmiştir.

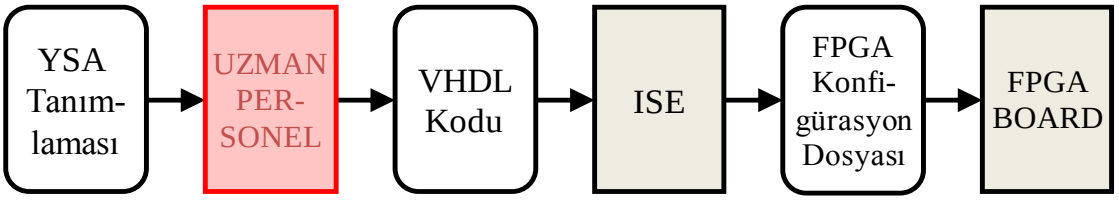
İnsan beyninde doğal olarak var olan sinir hücrelerinin matematiksel olarak taklit edilmesiyle üretilen modele yapay sinir ağları denir. Hâlihazırda geliştirilen YSA

modelleri insan beyninin değil tamamını, küçük bir kısmını dahi, tam olarak taklit edebilmekten uzaktırlar. Buna rağmen YSA'lar, elektrokimyasal olarak çalışan sinir hücrelerinden çok daha hızlı çalışabilme, yorulmama gibi avantajları nedeniyle odaklanmış olduğu problem için sonuç üretmede çok daha başarılıdır. YSA'lar da insan beyni gibi hata yaparak öğrenir, hatasını minimum değere indirmeye çalışır ve bir problem için bir kez eğitildikten sonra çok hızlı şekilde sonuca ulaşırlar. Bu öğrenmelerde bazen bir değişkenin gelecekteki değerini tahmin edebilmek için aynı değişkenin geçmişteki değerlerinin kullanıldığı teknik analiz yöntemi, bazen de, değişkeni etkileyen diğer değişkenlerin dikkate alındığı temel analiz yöntemi uygulanabilir.

YSA'lar yazılım (software) olarak modellendiğinde çok CPU [Central Processing Unit (Merkezi İşlem Birimi)] gücü gerektirir. Özellikle gerçek zamanlı uygulamalarda, anında YSA çıkışının hesaplanması gerekir. Bu amaçla birçok alternatif yöntem geliştirilmiştir. Bunlardan bir tanesi de yeniden yapılandırılabilir hesaplama (RC-Reconfigurable Computing) sistemi kullanımıdır.

RC sistemleri genel amaçlı bir bilgisayara eklenmiş programlanabilir FPGA yonga ya da yonga setlerinden oluşur. Bu sistemlerde programların yoğun kısımları özel olarak tasarlanmış modüller sayesinde FPGA yongaları üzerinde çalıştırılır. Çünkü FPGA'lar paralel işlem yapma ve yeniden yapılandırılabilme özelliklerine sahiptirler. YSA'lar paralel işlem yaptıklarından çok hızlı çalışan paralel işlemcilere ihtiyaç duyarlar. Değişik problemler için esnek, değişebilen katmanlara ve farklı transfer fonksiyonlu modellere sahip olmalıdırlar. YSA'nın içindeki hesaplamalar yüklü miktarda CPU gücü ve paralel hesaplama gerektirmektedir. Bu hesaplamalar FPGA da kolay olarak yapılabilmektedir. Diğer taraftan bir YSA'nın FPGA'ya uygulanması zaman alan ve uzman gerektiren bir iştir.

Şekil 1.1 de, bir YSA tasarımının FPGA'da gerçeklenmesi için izlenecek adımlar görülmektedir. Burada en önemli adım, YSA'nın bir sayısal devre olarak tasarlanması ve uygun bir donanım tanımlama dili [Hardware Description Language (HDL)]'nde kodlanmasıdır. Bu aşama uzman personel ve zaman gerektiren bir süreçtir.



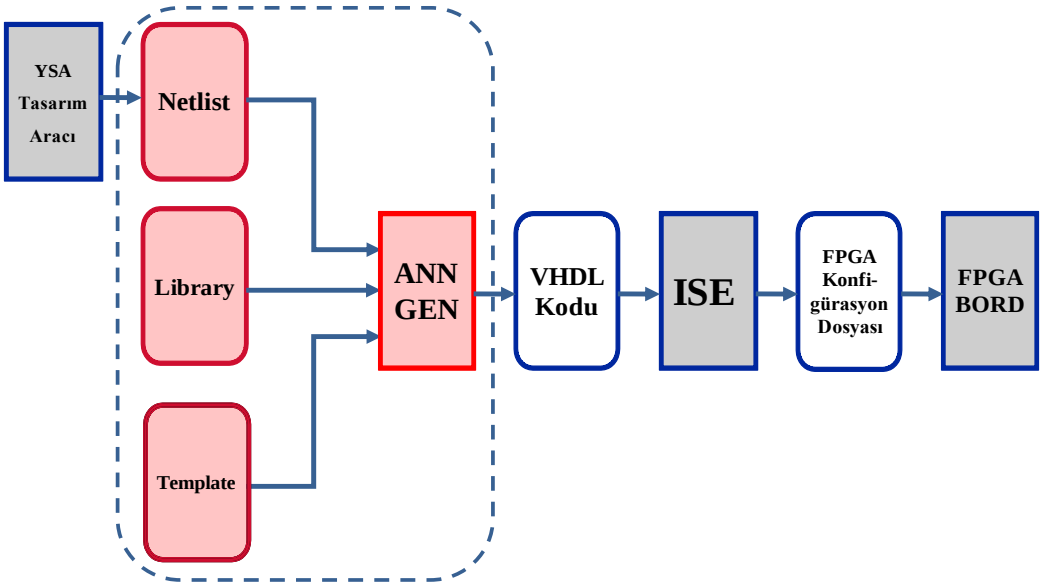
Şekil 1.1: Klasik tasarım akışı

Çalışmanın amacı: Yapay sinir ağlarının FPGA'ya uygulamasında veri yolu tasarımını otomatikleştirmek, böylece uzman gereksinimini azaltmak ve uygulama sürecini kısaltmaktır.

Bu amaçlar doğrultusunda, bu tez çalışmasında Yapay Sinir Ağları'nın otomatik olarak FPGA'lara uygulanmasına yardımcı olacak bir tasarım aracı ANNGEN [Artificial Neural Network GENerator (Yapay Sinir Ağlarının VHDL Kodunu Oluşturucu)] tasarlanmıştır (Şekil 1.2).

ANNGEN girdi olarak; metin tabanlı YSA tanımlamasını (NetList), tanımlaması yapılmış hâlihazırda kullanılabilecek sinir hücrelerinin listelendiği kütüphaneyi (Library), sinir hücrelerinin tanımlamasını ve VHDL [Very High Speed Integrated Circuit HDL (Çok Hızlı Entegre Devre Donanım Tanımlama Dili)] kodu yazımında gerekli diğer bazı şablonların bulunduğu şablon dosyasını (Template) alır. Sonuçta, verilen NetList'e uygun YSA tasarımı için gerekli VHDL kodunu oluşturur.

ANNGEN C++ yazılmıştır. Veri yapısı olarak altmış sekiz satırlık dokuz bölümden ve C++ programı da dört yüz otuz bir satırlık on dokuz fonksiyondan oluşmaktadır. Kütüphane dosyasında FPGA' da çalışan altı tane farklı transfer fonksiyonlu sinir hücresi bulunmaktadır. Burada tanımlanan sinir hücrelerinin VHDL kodları şablon dosyasında kullanılmıştır. Oluşturulmak istenen yapay sinir ağı için örnek NetList dosyası yazılmış ve ANNGEN çalıştırılarak yapay sinir ağının VHDL kodu yazdırılmıştır. Bu VHDL kodu Xilinx ISE Design Suite 12.1_1 Virtex5 yongasında çalıştırılmıştır. Böylece yeni bir YSA tasarım yöntemi geliştirilmiştir.



Şekil 1.2: Hedeflenen tasarım akışı

Tezin birinci bölümü tez ile ilgili genel bilgilerin verildiği giriş bölümüdür. Tezin ikinci bölümünde; YSA'nın yapısı, genel özellikleri, uygulama alanları ve YSA'da öğrenme, FPGA yongaları, avantajları, uygulamaları, FPGA'da tasarım, programlama ve FPGA da yapılan YSA uygulamaları anlatılmıştır. Üçüncü bölümünde; değişik transfer fonksiyonlarına sahip YSA'lar için FPGA'ya uygun VHDL kodları yazılarak kütüphane oluşturulmuştur. Yine bu bölümde YSA'nın FPGA'ya uygulanabilmesi için gerekli veri yolunu tasarlayacak ANNGEN programı yazılmıştır. Dördüncü bölümde ise örnek iki test durumu için ANNGEN'in test edilmesi anlatılmıştır. Son bölümde de ANNGEN'in çok kısa sürede kullanılacak YSA'nın VHDL kodunu yazdığı anlatılmıştır.

2. ÖN BİLGİ VE DİĞER ÇALIŞMALAR

2.1. YAPAY SİNİR AĞLARI

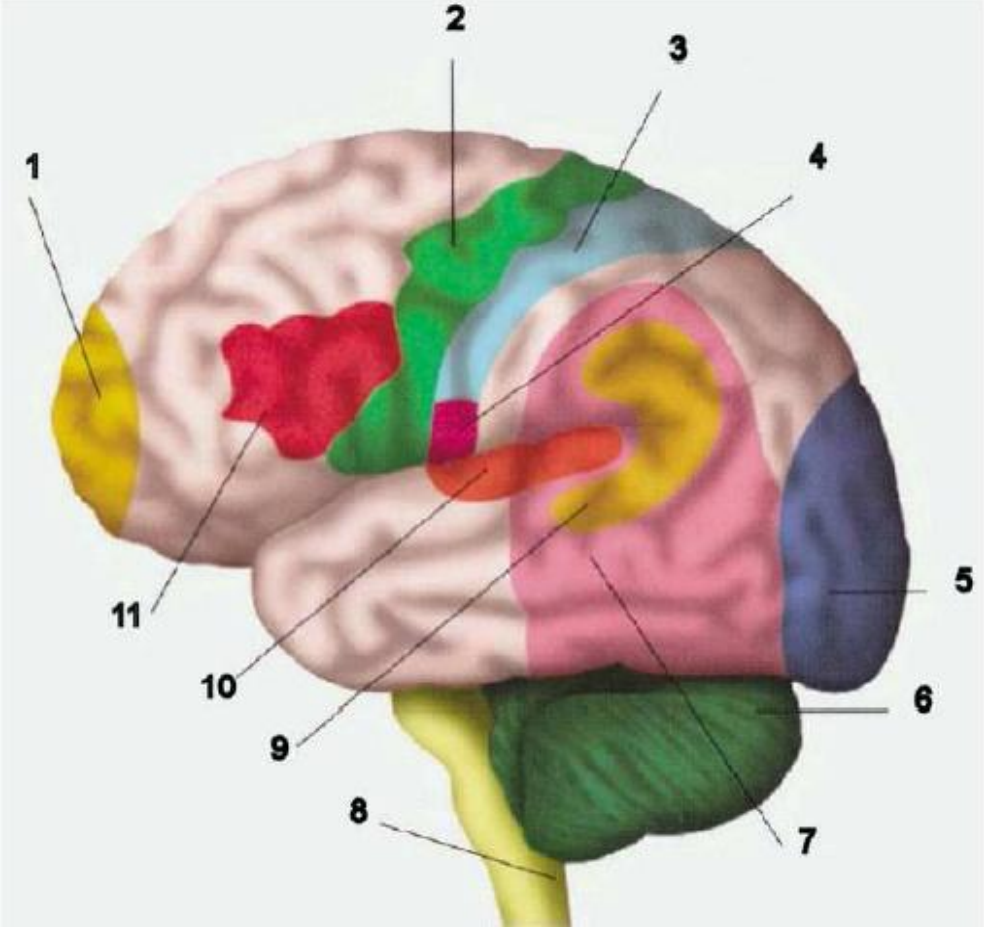
2.1.1. Sinir Hücreleri Ve Sinir Ağları

Yapay Sinir Ağları (YSA) beyni oluşturan sinir hücrelerini (nöron) matematiksel olarak taklit ederek akıllı bir sistem oluşturmaya çalışan bir yapay zekâ yöntemidir. Bir diğer ifadeyle YSA biyolojik sinir ağlarını taklit eden bilgisayar programlarıdır (Elmas, 2003). Yapay sinir ağları henüz biyolojik beyni tam anlamıyla taklit etmekten bir hayli uzaktır. Buna rağmen sinir hücreleri kimyasal olarak çalıştıklarından tepki hızları elektriksel olarak çalışan günümüz bilgisayarlarına göre oldukça yavaştır. Bu da bize bilgisayarların insan beynine göre özellikle belirli bir konuda sonuca ulaşmada ve hesaplamalarda daha üstün olduğunu göstermektedir.

İstatistikte veya zaman serilerinde tüm girdiler ve çıktılar sıralı işlenmesine rağmen YSA'da bu bilgiler paralel olarak işlenmektedirler. YSA mimarisini ışığı renklerine ayıran ışık prizmasına benzetebiliriz. YSA şebekesi de mimari ve matematiksel işlemler ile gelen verileri basit bileşenlerine ayırdıktan sonra bunları istenen çıktılara uygun olacak şekilde yeniden birleştirir. YSA, bir nehrin belli bir noktasından karşı kenardaki belli bir noktayı birleştiren köprü planına benzer. Bu planın içinde köprünün inşası, işletilmesi ve fayda sağlanmasına ait bilgiler bulunmaktadır. Durulan sahildeki bilgiler girilen bilgileri, karşı kenardaki bilgiler çıktıları ve bu ikisi arasındaki köprü de YSA mimarisini temsil eder. Köprüdeki kiriş, kolon, döşeme, kablo, vb kısımlar arasında matematiksel ve fiziksel kurallara göre yapılan bağlantılar gibi, YSA'larda da hücreler ve tabakalar arasında benzer bağlantılar kurulur (Şen, 2004).

2.1.2. İnsan Beyni

İnsan beyninin işlevleri henüz tam olarak keşfedilememiştir. Yapılan çalışmalarla, Şekil 2.1 ve Çizelge 2.1'de görüldüğü gibi, beynin hangi bölgesinin hangi göreve sahip olduğu yaklaşık olarak bulunmuştur. Bu ise başın bir bölgesinden hasar alan insanlardaki değişiklerin gözlenmesi ve ayrıntılı olarak yapılan tomografik taramalar ile belirlenmiştir (Şen, 2004).



Şekil 2.1: İnsan beyni ve fonksiyonları (Anon, 2011)

Çizelge 2.1: Beynin bölgelere göre fonksiyonları

Bölge	***Fonksiyonu***
1	*Prefrontal Korteks*: Zekâ, öğrenme, kişilik bölgesi
2	*Motor Bölge:* Karmaşık hareketlerin eşgüdümü ve istemli hareketler
3	*Duyusal Bölge*: Dokunma duyusu bilgilerinin işlenmesi
4	Tat bölgesi
5	Görme bölgesi
6	*Beyincik*: Denge bölgesi
7	Genel yorumlama bölgesi
8	*Omurilik*
9	*Dil Bölgesi*: Dil anlayışından sorumlu Wernicke bölgesi
10	İşitsel bölge
11	*Konuşma Merkezi*: Konuşmanın üretiminden sorumlu Broca bölgesi

Beynin haberleşme sistemini oluşturan sinirler, sinyal alma, işlem yapma ve elektrokimyasal sinyallerin sinir ağları içinde iletimleriyle görevlidirler.

İnsan beyni ortalama olarak 1,5 kg ağırlığındadır (Şenel, 2003). İnsan beyninin çalışma frekansı 100 hz (hertz)'dir (Elmas, 2007). Eğer bir insanın her saniye 600 bitlik yeni bilgi kayıt ettiğini düşünürsek, bu saatte yaklaşık 265 kbyte (kilobyte) günde de yaklaşık 6,5 Mbyte (Megabyte) eder. Bu bilgilerle insan beyninin hafıza kapasitesi dolmaz. Araştırmalara göre insan beyni ile aynı kapasiteye sahip bir bilgisayar yapılacak olsa 300 trilyon dolardan fazlaya mal olacaktır (Anon, 2011a). Yine bu makinenin çalışabilmesi için 1 trilyon Watt'lık enerjiye ihtiyaç olacaktır ve hacmi de elbette kafatasından binlerce kat büyük olacaktır. Çizelge 2.2'de insan beyni ile bilgisayarın karşılaştırılması yapılmıştır.

2.1.3. Biyolojik Sinir Hücresi

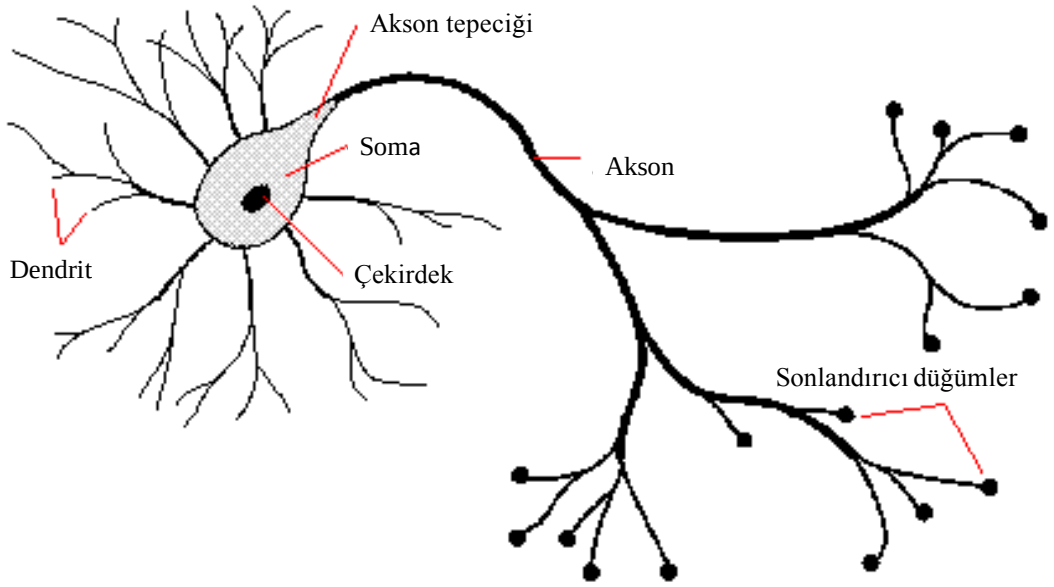
İnsan beyninde, tahminen 100 milyar kadar sinir hücresi ve $60x10^{12}$ (60 trilyon) sinaps bulunmaktadır (Freeman ve Skapura, 1991). Bu rakamın ne kadar büyük olduğunu anlamak için, kafatasımızın içindeki sinir hücrelerinin sayısı yaklaşık olarak Samanyolu'ndaki yıldız sayısı kadardır diyebiliriz. İnsanın bilişsel davranışlarının (öğrenme, hatırlama, düşünme, algılama gibi) temelinde sinir hücreleri bulunmaktadır (Kandel, 1991). Sinir hücreleri, sinir sisteminin temel işlem elemanlarıdır. Şekil 2.2'de, görüldüğü gibi, sinir hücresi; çekirdek (nucleus), dendritler, aksonlar (axon) ve sinapsler (synapse) olmak üzere dört temel parçadan meydana gelir. Bu organellerin görevleri Çizelge 2.3'te görülmektedir. Dendritler, diğer hücrelerden aldığı bilgileri hücrenin çekirdeğine iletir. Aksonlar ise elektriksel darbeler şeklindeki bilgiyi hücreden dışarı taşıyan organellerdir. Aksonların bitimi, ince yollara ayrılabilir ve bu yollar diğer hücreler için dendritleri oluşturur. Buradaki bağlantı elemanına da sinaps denir (Fırat ve Güngör, 2004).

Sinir hücreleri arasındaki iletişim sinapslar yardımıyla elektro-kimyasal olarak gerçekleştirilir. Alıcı durumunda bulunan sinir hücresindeki kimyasal faaliyet, bazı iyonların daha kolay geçmesini sağlayacak şekilde zarın geçirgenliğini değiştirir. Uyarıcı durumda, hücre zarının geçirgenliğinin değişmesi sonucunda hücre içerisine daha kolay akan pozitif iyonlar, sinir hücresinin de-polarize olmasını sağlar ve hücrede sinirsel bir akımın başlamasına yol açar. Engelleyici durumda ise, negatif iyonlar sinir

hücresini daha da polarize ederek sinirsel akımın durmasını sağlarlar (Cooper, 2011). Şekil 2.3'te, sinir sistemine gelen uyarıların alıcılar tarafından alınması, sinir ağı ile beyne bildirilmesi ve çıktı sinyalinin ileticilerle gideceği adrese gönderilmesi gösterilmiştir.

Çizelge 2.2: İnsan beyni ve bilgisayarın karşılaştırılması

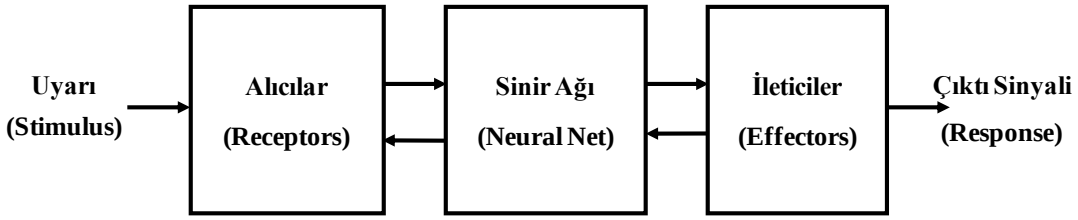
	Beyin	**Bilgisayar**
İşlem Gücü	Yavaş, basit fakat çok fazla işlemci	Hızlı, karışık fakat çok az işlemci
Hafıza	İşlemcilerle bütünleşik, içeriği adreslenebilir	İşlemciden ayrı, Rastgele erişimli hafıza türü (RAM)
Hesaplama	Paralel, dağınık ve her biri kendi başına çalışabilir	Sıralı (seri), merkezi hesaplama
Güvenilirlik	Güvenilir, eksik bilgiler onarılabilir, sağlam	Kolay bozulabilir, bozulan verileri kurtarmak zordur



Şekil 2.2: Basit bir sinir hücresi

Çizelge 2.3: Sinir hücresini oluşturan organeller ve görevleri (Petriu, 2005)

Organel	**Görevi**
Çekirdek (Nucleus)	Sinir hücresinin çekirdeğidir.
Akson (Axon)	Hücrenin diğer hücrelere kimyasal iletimde bulunduğu ucudur. Bir hücre sadece bir adet aksona sahip olmasına rağmen her aksonun birden fazla ucu vardır. Boyları bir milimetreden küçük olabileceği gibi bir metreden de büyük olabilir (Nabiyev, 2005).
Dendrit	Hücrenin alıcısıdır. Kimyasal olarak algılama işlemi yapar. Bir sinir hücresi birden fazla dendrite sahiptir ve her dendritin de birden fazla ucu vardır.
Sinaps (Synapse)	İki sinir hücresinin daha doğrusu birisinin aksonu ile diğerinin dendritini birbirine bağlayan bağlantı elemanıdır.



Şekil 2.3: Sinir sisteminin blok diyagramı (Haykin, 1994)

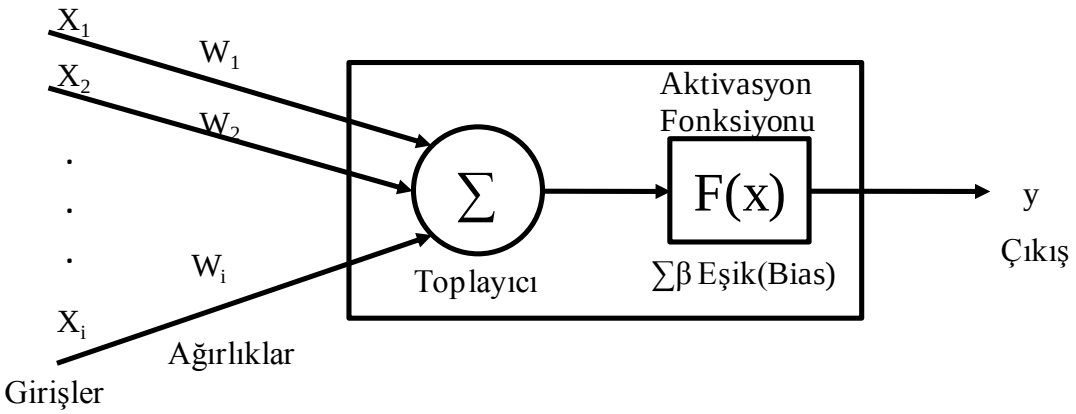
2.1.4. Yapay Sinir Hücresi

Şekil 2.4'te, gerçek bir sinir hücresinin matematiksel olarak modellenmesiyle elde edilen yapay sinir hücresi gösterilmiştir. Gövdenin giriş birimi olan bağlantıların her birinin kendine ait bir ağırlık çarpanı (W_i) vardır. Ağırlık değeri pozitif veya negatif olabilir. Uygulanan sinyallerin ağırlık değerleriyle çarpımları, iki kısımdan oluşan gövdenin ilk kısmında toplanır. Bu toplam, ikinci kısmı tanımlayan transfer fonksiyonunun argümanı olur.

Ağırlıklar *(W_1, W_2,..., Wn)*: Girişlerin sinir hücresi içinde hangi oranda veya hangi ağırlıkta olacağını belirleyen değerlerdir. Ağırlıkların sayısal değerinin negatif ya da pozitif olması ağa olan etkisinin negatif ya da pozitif yönde olduğunu işaret eder (Öztemel, 2003).

Toplama Fonksiyonu (Toplayıcı): Her girdi değeri kendine ait olan ağırlık değeri ile çarpıldıktan sonra çıkan tüm ağırlık-girdi çarpım değerleri toplanarak ağa uygulanan net girdi elde edilmiş olur. Toplayıcı fonksiyonları olarak Toplam, Çarpım, Maksimum, Minimum ve Kümülâtif Toplam kullanılır (Öztemel, 2003). En çok tercih edileni toplam fonksiyonu ($\Sigma\, x_i \cdot w_i$) dur.

Transfer Fonksiyonu: Sıkıştırma, aktivasyon, işlemci veya eşik fonksiyonu olarak da adlandırılan transfer fonksiyonu, matematiksel olarak modellenmiş bir yapay sinir hücresinin çıktısının büyüklüğünü sınırlandıran bir fonksiyondur (Mandic ve Chambers, 2001). Bir ağdaki bütün hücrelerin transfer fonksiyonları birbirinden farklı olabilir.



Şekil 2.4: Matematiksel modelleme ile elde edilen bir yapay sinir hücresi

En çok kullanılan transfer fonksiyonları şunlardır.

a. Doğrusal Fonksiyon (Şekil 2.5) : Hücreye toplayıcıdan gelen veriler bir α katsayısı ile çarpılarak sonuç üretilir. α=1 ise girdiler olduğu gibi çıkar ($\boldsymbol{n = \Sigma x_i \cdot w_i}$).

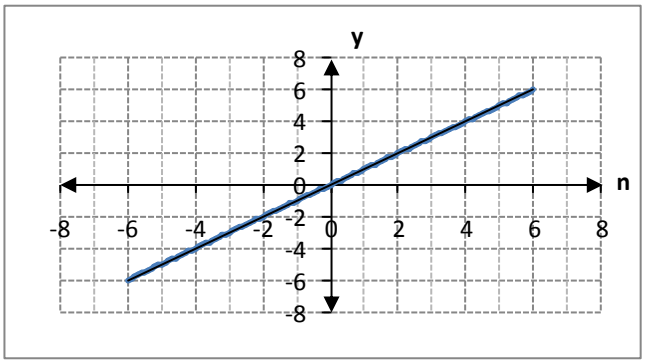
$$y=F(n)= \alpha \cdot n \quad ise \tag{2.1}$$

$$y=n \tag{2.2}$$

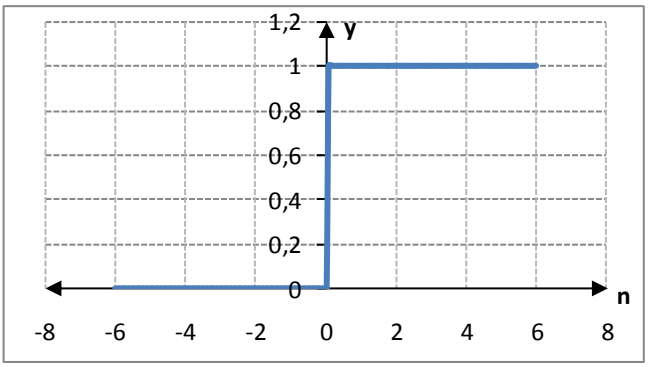
b. Adım Fonksiyonu (Şekil 2.6) : Toplayıcıdan gelen veri belirlenen bir eşik değerinin üstünde ya da altında olmasına göre 1 ya da 0 değerlerini alır.

$$n<0 \quad ise \quad y=0 \tag{2.3}$$

$$n \geq 0 \quad ise \quad y=1 \tag{2.4}$$



Şekil 2.5: Doğrusal transfer fonksiyonu



Şekil 2.6: Adım transfer fonksiyonu

c. Eşik Değer Fonksiyonu (Şekil 2.7) : Toplayıcıdan gelen verilerin belirlenen değerler arasında veya verilen değerden büyük ya da küçük olmasına göre değer alırlar.

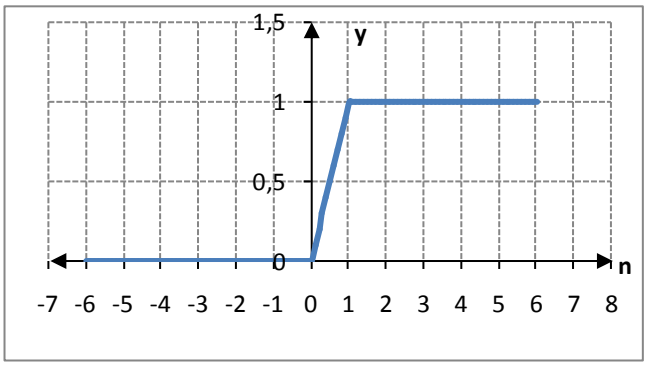
$n<0$ ise $y=0$ (2.5)

$0 \leq n \leq 1$ ise $y=n$ (2.6)

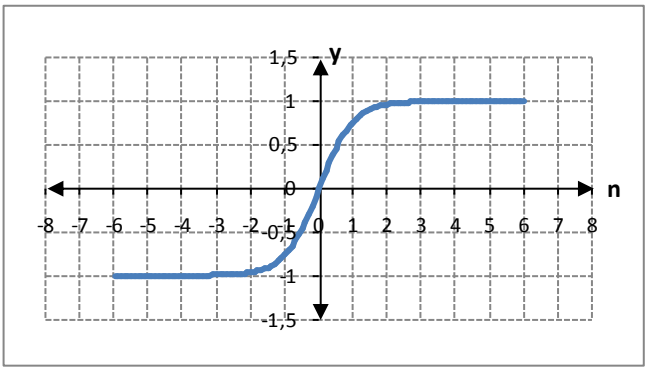
$n>1$ ise $y=1$ (2.7)

d. Hiperbolik Tanjant Fonksiyonu (Şekil 2.8) : Çıktı değeri hücreye toplayıcıdan gelen verilerin tanjant fonksiyonuna tabi tutulmasıyla hesaplanır (Anon, 2011b).

$$y = \frac{e^{n} - e^{-n}}{e^{n} + e^{-n}} \tag{2.8}$$



Şekil 2.7: Eşik değer transfer fonksiyonu



Şekil 2.8: Hiperbolik tanjant transfer fonksiyonu

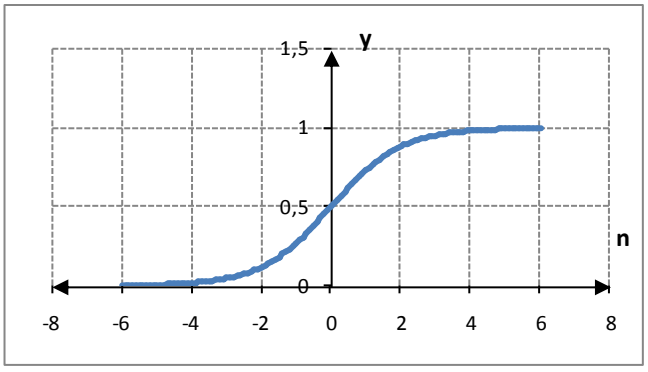
e. Sigmoid Fonksiyonu (Şekil 2.9) : YSA'da en çok kullanılan doğrusal ve doğrusal olmayan davranışlar arasında denge sağlayan sürekli artan bir fonksiyon olarak tanımlanır (Demuth ve diğ., 2011).

$$y = \frac{1}{1+e^{-n}} \tag{2.9}$$

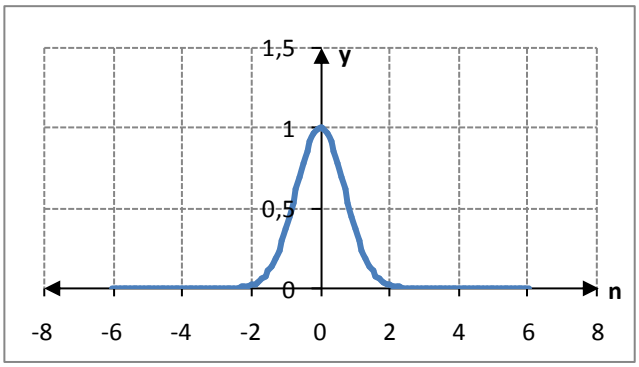
f. Gauss Fonksiyonu (Şekil 2.10) :

$$y = e^{-n^2} \tag{2.10}$$

Diğer en çok kullanılan transfer fonksiyonları Çizelge 2.4'te görülmektedir.

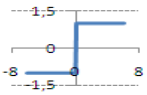
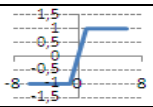
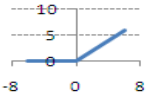
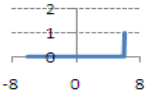
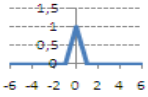


Şekil 2.9: Sigmoid transfer fonksiyonu



Şekil 2.10: Gauss transfer fonksiyonu

Çizelge 2.4: Sık kullanılan ve Matlab'in desteklediği diğer transfer fonksiyonları (Sarle, 1994)

İsim	Formül	Grafikleri
Simetrik Sabit Limit	$n<0$ ise $y=-1$ $n\geq 0$ ise $y=1$	
Simetrik Doymuş Doğrusal	$n<-1$ ise $y=-1$ $-1\leq n\leq 1$ ise $y=n$ $n>1$ ise $y=1$	
Pozitif Doğrusal	$n<0$ ise $y=0$ $n\geq 0$ ise $y=n$	
Rekabetçi	Sinir hücresi maksimumu ise $y=1$ Diğer durumlarda ise $y=0$	
Üçgensel	$-1 \leq n \leq 1$ ise $y=1-mutlak(n)$ diğer durumlarda ise $y=0$	

Şekil 2.11'deki yapay sinir hücresine, Çizelge 2.5'deki örnek giriş ve ağırlık değerleri verilirse; sigmoid olarak belirlemiş olduğumuz transfer fonksiyonuna girecek değer

$$n = \sum X_i \cdot W_i = 0.6 \cdot 0.3 + 5.2 \cdot 0.3 + 0.8 \cdot 0.7 = 2.3 \qquad (2.11)$$

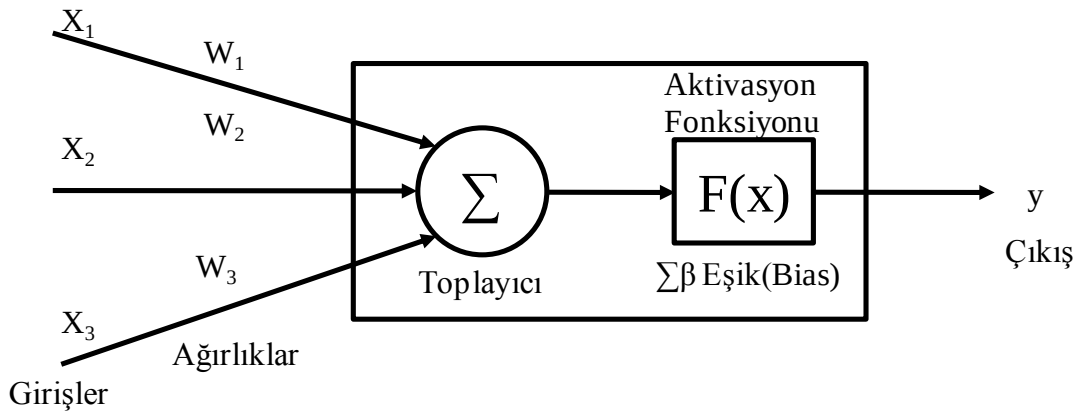
olacaktır. Hücremizin ürettiği y değeri ise;

$$y = \frac{1}{1+e^{-n}} = \frac{1}{1+e^{-2.3}} \cong 0.9 \qquad (2.12)$$

olur.

Çizelge 2.5: Örnek giriş ve ağırlık değerleri

X_1	X_2	X_3	W_1	W_2	W_3
0.6	*5.2*	*0.8*	*0.3*	*0.3*	*0.7*



Şekil 2.11: Yapay sinir hücresi

2.1.5. Yapay Sinir Ağlarının Yapısı

Bir yapay sinir ağı birçok sinir hücresinden ve birden fazla katmandan oluşmaktadır. Bir sinir ağındaki ilk katman giriş katmanı, son katman ise çıkış katmanıdır. Arada kalan katmanlara ise gizli katman ya da ara katman adı verilir. Bir ağdaki gizli katman sayısı birden fazla olabilir.

Yapay sinir ağları bilinen bilgi işleme yöntemlerinden farklı özelliklere sahiptir. Paralellik, hata toleransı, öğrenilebilirlik ve gerçekleme kolaylığı bu özelliklerden bazılarıdır. Bu özellikleri ile YSA'lar alışılagelmiş hesaplama yöntemlerine göre daha başarılı sonuçlar üretebilmektedirler. Yapay sinir ağlarında bilgilerin işlenmesi paralel ve birbirinden bağımsızdır. Aynı tabakadaki bağlantılar arasında zaman bağımlılığı olmadığından tamamı ile eşzamanlı çalışabilmekte dolaysıyla da bilgi akış hızı artmaktadır. Paralel çalışma prensibinden dolayı herhangi bir birimde meydana gelen hata tüm sistemde belirgin bir hataya neden olmamaktadır. Sadece hücrenin ağırlıkları oranında bir etkilenme gerçekleşmektedir. Böylece yerel hatalardan en az şekilde etkilenmektedir.

Öğrenme yapay sinir ağlarında bağlantı ağırlıklarının yenilenmesi şeklinde olmaktadır. Elde edilen bilgiler bağlantı ağırlıklarında uzun süre saklanabilirler. Ayrıca öğrenme yeteneği sayesinde yapay sinir ağlarıyla tam tanımlı olmayan problemlerin

çözülebilmesi mümkündür. Paralel çalışan bir yapay sinir ağı modeli basit işlemler içerdiğinden ve karmaşık bir yapıya sahip olmadığından birçok sorunun çözümlenmesinde tercih edilir. Bir yapay sinir ağı için istenilen hedef ve eldeki verilerin türlerine karar verildikten sonra beklenen çıktıları girdilerden elde etmek için bu ağdaki bilinmeyen bağlantı değerleri ardışık yaklaşımlarda eğitilerek tespit edilir.

2.1.5.1. Tek Katmanlı-İleri Beslemeli (Feed Forward) Yapay Sinir Ağları

Şekil 2.12'de görülen tek katmanlı ileri beslemeli yapay sinir ağı en basit ağ yapısıdır. Bir giriş ve bir çıkış katmanı vardır. Sinir ağı ileri beslemeli olduğundan bilgi girişten çıkışa doğru ilerler. Tek katmanlı olarak adlandırılmasının nedeni, giriş katmanının veri üzerinde hiçbir işlem yapmadan veriyi çıkış katmanına iletmesidir.

2.1.5.2. Çok Katmanlı İleri Beslemeli Yapay Sinir Ağları

Şekil 2.13'deki gibi çok katmanlı ileri beslemeli yapay sinir ağlarında giriş katmanı, gizli katmanlar ve çıkış katmanı vardır. Sinir ağı ileri beslemeli olduğundan bilgi girişten çıkışa doğru ilerler.

2.1.5.3. Geri Dönüşlü (Recurrent) Yapay Sinir Ağları

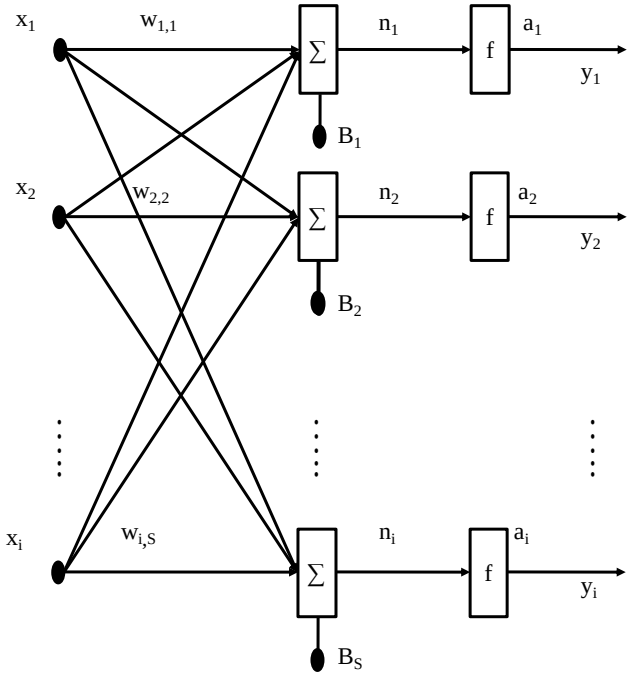
Geri dönüşlü YSA'lar özellikle birinci dereceden doğrusal sistemleri modellemede oldukça başarılıdır. Zamana bağlı olayları işlemede daha önceden elde edilen sonuçları değerlendirmedeki başarılı çıktıları ile özellikle ses ve karakter tanıma problemlerinde etkin olarak kullanılmaktadırlar. Bu yapıdaki sinir ağlarının ileri beslemeli ağlardan farkı en az bir tane geri besleme çeviriminin olmasıdır (Demuth ve diğ., 2011).

Geri dönüşümlü YSA'lardan en basiti ve kullanımı en kolay olan Elman ağıdır. Hopfield, Counterpropagation, Cognitron, Kendini Ayarlayan Haritalı Ağlar (SOM–Self Organizing Maps) ve Boltzman Makinesi gibi birçok geri dönüşümlü YSA modeli vardır. Bu dönüşlü modellerin çoğu birbirlerine benzemektedirler (Öztemel, 2003).

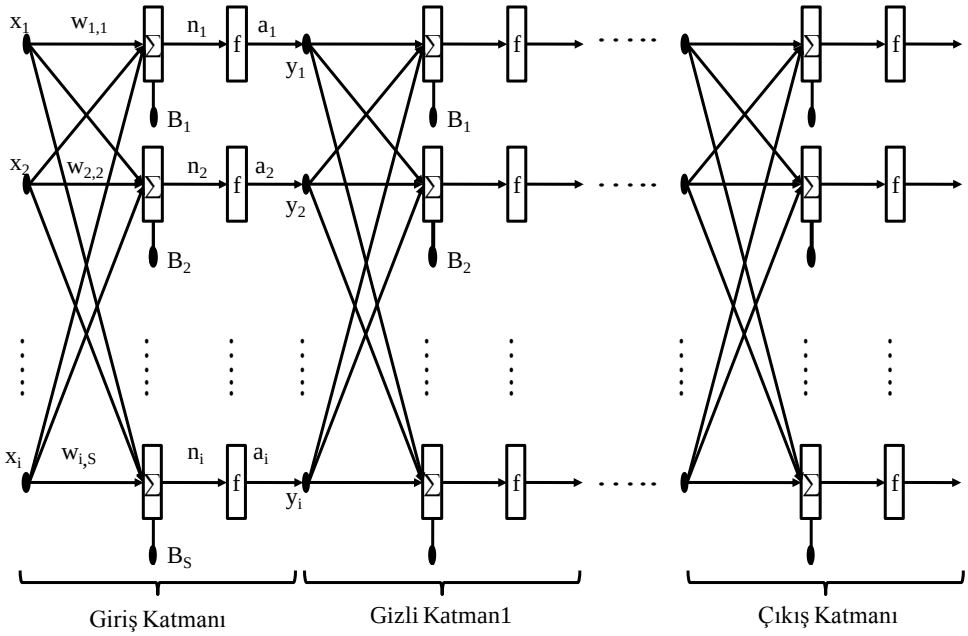
2.1.5.4. Radyal Tabanlı (Radial Basis) Yapay Sinir Ağları

Radyal tabanlı yapay sinir ağları, duyarlı almaç bölgelerinin olduğu giriş tabakası, radyal tabanlı sinir hücrelerini içeren gizli tabaka ve çoğunlukla doğrusal transfer fonksiyonlu sinir hücrelerinden ibaret çıkış tabakasından oluşur. Radyal tabanlı ağlar, geri yayılım algoritmalı ileri beslemeli ağlardan daha fazla nöron kullanımına ihtiyaç

duysa da eğitim süresi çok daha kısadır. Yoğun eğitim verisiyle daha iyi sonuçlar verir (Sarle, 1994).



Şekil 2.12: Tek katmanlı bir YSA (Demuth ve diğ., 2011)



Şekil 2.13: Çok katmanlı bir YSA (Demuth ve diğ., 2011)

2.1.6. Yapay Sinir Ağlarında Öğrenme Kuralları

Bir yapay sinir ağının eğitilmesinde yani ağın öğrenmesinde kullanılacak yöntem çok önemlidir. Çünkü YSA'lar da bir bebek beyninin öğrenmesi gibi deneme yanılma yoluyla hata yaparak öğrenir. Eğitimdeki amaç bulunması gereken doğru sonuçlara en yakın çıktıyı üretebilmektir. Bu nedenle ağ verilen girdilere uygun işlem yaptıktan sonra bir çıktı verir. Çıktı ile hedef değerler arasındaki fark hatadır. YSA bu hatayı kabul edilebilir sınırlara çekmek için yapılandırma değerini (ağırlıklar varsa eşik değerlerini) değiştirerek işlemi tekrarlar. Eğitim setinin ağ içinde bir kez işlemden geçirilmesine devir denir. Devir sayısının çok olması ağın öğrenme sürecinde önemli bir etken olsa da, sayının yüksek olması performansı düşüren bir etkendir. Ağın mimarisi, transfer fonksiyonu, öğrenme yöntemi ve devir sayısı seçilirken optimizasyonun iyi bir şekilde yapılması gerekir. En bilinen ve en yaygın olarak kullanılan öğrenme kuralları şunlardır: Hebb Kuralı, Hopfield Kuralı, Delta Kuralı ve Kohonen Kuralıdır (Yıldız, 2006).

2.1.6.1. Hebb Kuralı (1949)

Hebb kuralı diğer öğrenme kurallarının da temelini oluşturmaktadır. Bu kuralda bir sinir hücresi başka bir sinir hücresinden bilgi alırsa ve her iki hücre de matematiksel olarak aynı işareti taşıyorsa yani aktif ise bu iki hücre arasındaki ağ kuvvetlendirilmelidir, tersi durumda ise zayıflatılmalıdır (Dazsi ve Enbody, 2001).

2.1.6.2. Hopfield Kuralı (1982)

Hebb kuralına benzeyen bu kuralın farkı YSA elemanlarının bağlantılarının ne kadar kuvvetlendirilmesi veya zayıflatılması gerekliliğini belirlemesidir. Öğrenme katsayısı (0–1 arası) oranında ağırlık değerleri artırılır ya da azaltılır (Tebelkis, 1995).

2.1.6.3. Delta Kuralı

Bu kuralda amaç hedef çıktı ile elde edilen çıktı arasındaki hata karelerinin ortalamasını en aza indirmektir. Bu nedenle bu kuralda hedef çıktı ile elde edilen çıktı arasındaki farkı azaltmak için YSA elemanlarının bağlantılarının ağırlık değerleri sürekli yeniden hesaplanır. Bulunan hatalar en son katmandan geriye doğru ardışık iki katman arasındaki bağlantı ağlarına dağıtılır. Bu işleme yani hatanın geriye dağıtılmasına geri besleme işlemi denir (Çorumluoğlu ve diğ., 2005).

2.1.6.4. Kohonen Kuralı (1998)

Kohonen kuralı biyolojik sinir hücrelerinden ilham alınarak oluşturulmuştur (Yıldız, 2006). Bu kuralda sinir hücreleri ağırlıklarını değiştirmek için birbirleri ile yarışırlar. En büyük çıktıyı üreten hücre yarışı kazanır, bağlantı ağırlıklarını değiştirir ve yakınındaki hücrelere göre daha kuvvetli hale gelir. Bir hedef değerler dizisi olmasına gerek duyulmayan bu kuralda hücreler kendi kendine öğretmensiz eğitimini tamamlarlar.

2.1.7. Yapay Sinir Ağlarında Öğrenme Stratejileri

2.1.7.1. Öğretmenli Eğitim

Bu eğitimde, verilen giriş değerlerine karşılık gelen hedef çıktı değerleri vardır. Verilen değerlere göre hedef çıktıyı üretebilmek ağın görevidir. Ağın çıktıları sanki bir öğretmen varmış gibi hedef değerle kıyaslanır ve kabul edilebilir değerler arasında olup olmadığına göre eğitime devam edilir. Hedef değerler ile elde edilen değerler arasındaki farkın yani hatanın karelerinin ortalaması en küçük olacak şekilde ağırlıklar daima güncellenir ve işleme devam edilir (Şen, 2004).

Delta Kuralı, Genelleştirilmiş Delta Kuralı, Rastsal Öğrenme Kuralı, Takviyeli Öğrenme Kuralı öğretmenli eğitim stratejilerinden bazılarıdır (Çorumluoğlu ve diğ., 2005), (Freeman ve Skapura, 1991).

2.1.7.2. Öğretmensiz Eğitim

Sınıflandırma problemlerinde kullanılan bir eğitim yöntemidir. Eğitimde hedef çıktı değerleri olmayıp ağa yalnızca giriş değerleri verilir ve sınıflandırma yapılacak örnekteki parametreler arasındaki ilişkileri ağın kendi kendine öğrenmesi beklenir.

Sınıflandırma işleminde hedef olabildiğince farklı sınıflandırma yapmak olduğundan kullanıcının elde ettiği sınıfların ne anlama geldiğini yorumlaması gerekir. Signal Hebbian Öğrenme Kuralı ve Diferansiyel Hebbian Öğrenme Kuralı öğretmensiz eğitim stratejisine örnek olarak verilebilir (Şen, 2004).

2.1.7.3. Karma Eğitim

Ağın hem öğretmenli ve hem öğretmensiz öğrenme işlemlerinin birkaçının birlikte kullanılarak eğitilmesine denir. Radyal tabanlı YSA'lar ve olasılık tabanlı ağlar bunlara örnek olarak verilebilir.

2.1.8. Bir YSA Modellemesinde Dikkat Edilecek Hususlar

Bir problemi YSA kullanarak çözmek istediğimizde şu işlem basamaklarını uygulamak gerekir.

1. Ağın eğitim ve testinde kullanılacak veriler toplanır. Değişkenin kaç tanesinin girdi, kaç tanesinin çıktı olacağına karar verilir, giriş ve çıkış katmanında kaçar hücre olacağı belirlenir.

2. Problemin türüne göre toplanan veriler eğitim ve test için ayrıştırılır. Bu ayrıştırmada belli bir oran olmasa da genellikle %70, %20 ve %10'luk paketlere ayrılır. Eğer veri dizisinde sıra önemli değilse örnekler veriler arasından rastgele seçilirse eğitim daha başarılı olur.

3. Ağın yapısına problemin çeşidine göre karar verilir.

4. Problemin türüne göre öğrenme yöntemi seçilir.

5. Ağa uygulanacak ilk değerler veri olarak alınır. Bu veriler ağa verilecek şekle getirilir ve ağın giriş katmanına uygulanır.

6. Eğitime başlanır, hatayı azaltmak için ağırlıklar sürekli güncellenir.

Hata kabul edilebilir sınırlara geldiğinde eğitim durdurulur ve ağ test edilir. Paketlere ayrılan verilerin ilk paketi ile eğitim gerçekleştirildikten sonra buradan elde edilen ağırlıklar ikinci veri paketine uygulanır. Eğer ikinci paketteki eğitimden (onaylama-validation) sonra ağırlıklardaki değişimler ihmal edilebilir düzeyde ise ağ kullanılmaya hazır demektir. Eğer testlerde kabul edilebilir sonuçlar elde edilmemiş ise şu işlemler yapılabilir.

1. Eğitim için daha fazla ve daha uygun veriler toplanıp ağa uygulanabilir.

2. Eğitim ve test için ayrılan veriler tekrar düzenlenebilir.

3. Başka bir YSA modeli kullanılabilir veya ağın yapısı değiştirilebilir.

4. Öğrenme algoritması değiştirilebilir.

5. Başlangıç değerleri yenilenip ağ yeniden eğitilebilir (Yıldız, 2006).

2.1.9. Yapay Sinir Ağlarının Genel Özellikleri

Yapılan çalışmalarla değişik amaçlı çok sayıda sinir ağı geliştirilmiştir. Bunların yapısı, çalışması, işlem prensibi farklı olsa da bazı ortak özellikleri bulunmaktadır.

Bu özellikler, öğrenme, genelleme, hata, tolerans, paralel çalışma, uygulanabilirlilik, doğrusal olmama, bilginin saklanması, dereceli bozulma, kendi ilişkisini oluşturma, algılamaya yönelik alanlarda kullanılabilirlilik, sınırsız sayıda değişken ve parametre kullanımıdır (Tebelkis, 1995), (Öztemel, 2003).

2.1.10. Yapay Sinir Ağlarının Dezavantajları

1. YSA'lar üzerinde çalışacağı donanıma çok fazla bağımlıdır. Paralel işlem yaptıklarından gerçek zamanlı işlemlerde çok hızlı çalışan paralel işlemcilere ihtiyaç duyarlar.

2. YSA'lar her çeşit problemin çözümü için kullanılamazlar. Probleme göre ağ yapısında ve verilerin girilmesinde değişiklik yapılması gerekmektedir. Eğer değişiklik yapılmadan kullanılırlarsa ilgisiz ve kabul edilmez sonuçlar üretebilirler.

3. YSA'larda probleme uygun ağ yapısının belirlenmesi deneme yanılma yöntemi ile yapıldığından doğru ağ modeli kullanılmazsa performansı düşük sonuçlar elde edilebilir.

4. YSA'nın ürettiği sonuçlardan ağırlıkların yorumlanması zordur, YSA modeli kapalı bir kutu gibi düşünülebilir.

5. Ağın davranışlarını açıklamak kolay olmaz, bu da ağa olan güveni azaltır. Özellikle hayati öneme sahip konularda ürettiği sonucun nedeninin açıklanamaması kullanım alanını daraltmaktadır.

6. YSA'lar sayısal bilgilerle çalışmaktadır. Bu nedenle ağı eğitmek önemlidir. Tüm verilerin ağa girmeden önce sayısallaştırılıp ölçeklendirilmesi gereklidir. Bu işlemdeki başarı kullanıcının tecrübesi ile doğru orantılıdır.

7. Bir diğer dezavantaj da ağın ne kadar eğitileceğinin bilinmemesidir. Her zaman hatanın kabul edilebilir değer içerisinde kalmasıyla eğitim tamamlanır ama bunun

doğru bir çözüm olacağı kesin değildir. Yine ağın yerel minimum veya maksimuma (stuck at local minima or maksima) takılması ya da ezberlemesi de bir dezavantajdır (Yıldız, 2006).

2.1.11. Yapay Sinir Ağlarının Uygulama Alanları

Örüntü Tanıma (Pattern Recognition): Ses, veri iletimi, hareket tespiti, yüz tanıma, hedef tespiti gibi işlemlerde, robotik sistemlerde, karakter, imza, parmak izi tanımada ve kalite kontrolünde uygulanmaktadır.

Verilerin Yorumlanması: Finans alanında, güvenlik sistemlerinde, jet ve roket motorlarının geliştirilmesinde, tıbbi araştırmalarda, hava durumu tahminlerinde ve personel seçiminde uygulanmaktadır.

Optimizasyon: Birçok ticari ve bilimsel konuda incelenen olayın hedefinin, verilen kısıtlar altında maksimize ya da minimize edilmesi optimizasyon olarak bilinir. Optimizasyon için önceki çalışmalarda klasik birçok yöntem geliştirilmiş olmasına karşılık, bu işlemin YSA modellemesi ile yapılması en azından sınırlayıcı matematik kabullerin bulunmaması açısından yararlıdır.

Fonksiyon Yaklaşımı: Matematiksel fonksiyonu bilinmeyen birçok verinin modellenmesinde uygulanmaktadır.

Diğer Uygulama Alanları: Kontrol, arama çalışmaları, verilerin sınıflandırılması ve kümelendirilmesi, verilerin filtrelenmesi, elektrik sarfiyatı tahmini, gelecek turist sayısının belirlenmesi gibi işlemlerde, verilerin taklit edilmesinde, haritacılıkta, bazı elektriksel ve elektroniksel hesaplamalarda uygulanmaktadır (Yıldız, 2006).

2.2. FPGA

2.2.1. FPGA Yongaları

FPGA (Field Programmable Gate Array - Alanda Programlanabilir Kapı Dizileri), programlanabilir mantık bloklarına, bu blok dizisini çevreleyen giriş-çıkış bloklarına, bu bloklar arasındaki ara bağlantıları olan üretimi tamamlanmış ama yapılandırılması yapılmamış sayısal tümleşik devrelerdir. Mantıksal bloklar basit mantıksal kapıların (AND, XOR…) işlemlerini yerine getirmek için programlanabildikleri gibi, daha karmaşık fonksiyonları da (şifre çözücüler, basit matematiksel fonksiyonlar vb.) yerine getirebilmek için programlanabilmektedirler. Tasarımcının ihtiyacı olan mantıksal fonksiyonları gerçekleştirebilmesi amacıyla, kullanıldığı yerde programlanabilir olarak üretilirler. Dolayısıyla her bir mantık bloğunun işlevi kullanıcı tarafından düzenlenebilmektedir. FPGA ile temel mantık kapılarının ve yapısı daha karmaşık olan devre elemanlarının işlevselliği artırılmaktadır. Alanda programlanabilir ismi verilmesinin nedeni, mantık bloklarının, ara bağlantıların ve giriş/çıkış bloklarının imalat sürecinden sonra programlanabilmesidir.

Birçok FPGA yongası, ayrıca aritmetik işlem blokları ve hafıza yapılarına sahiptirler. Bunlar basit Flip-Floplar olabileceği gibi daha bütünsel hafıza blokları da olabilmektedir (Koyuncu, 2008).

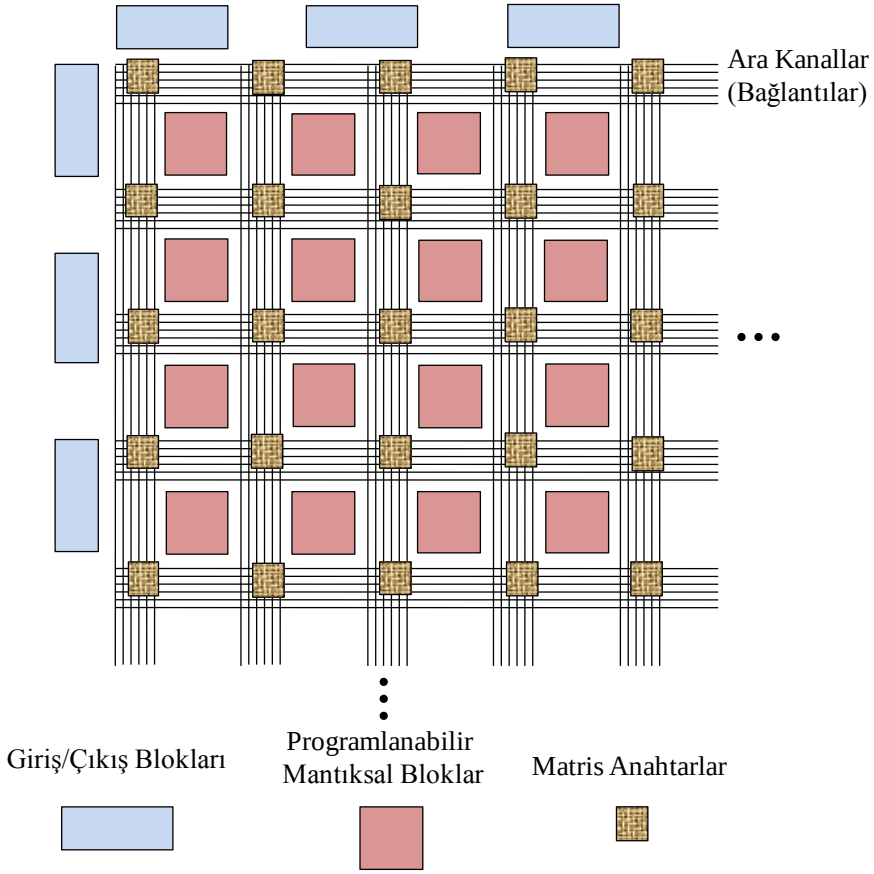
Genel yapısı Şekil 2.14'te gösterilmiş olan FPGA yongasının yapısındaki mantıksal bloklar, aralarındaki bağlantılar ve giriş/çıkış blokları sırayla aşağıda açıklanmıştır.

a. Yapılandırılabilir Mantıksal Bloklar [Configurable Logic Blocks (CLB)]: CLB blokları, mantıksal (boolean) fonksiyonların oluşturulabildiği LUT (Look-Up Table – Bakılan Tablo), Carry Logic ve Flip-Flop'lardan oluşmaktadır. Tipik bir FPGA onbinlerce CLB ve flip-flop içerebilir. Büyüklük ölçüsü olarak, CLB'nin giriş çıkış sayısı, CLB'nin oluşumunda kullanılan transistor sayısı veya CLB'nin gerçekleyebileceği mantıksal fonksiyon sayısı kullanılmaktadır (Bürhan ve Gülenç, 2011). Hafıza kapasitesi (LUT) giriş sayısı ile sınırlıdır. CLB, kullanıcının oluşturmak istediği mantıksal devre için fonksiyonel elemanlar sağlar. CLB mimarisinin esnekliği ve simetrisi, uygulamaların kolaylıkla yerleştirilmesine ve yönlendirilmesine olanak tanır.

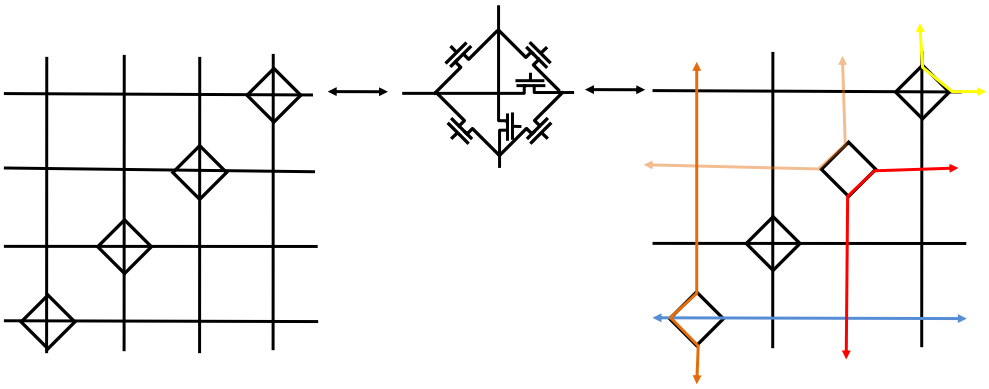
b. Giriş Çıkış Blokları (Input/Output Blocks (IOB)): IOB'ler yonganın iç sinyal hatları ile pinleri arasında programlanabilir arabirim görevini yaparlar. IOB'ler sayesinde FPGA pinleri giriş, çıkış ya da çift yönlü olarak programlanabilir. FPGA yongasının türüne göre bir yongadaki IOB sayısı (dolayısıyla pin sayısı) 1000'li sayılara kadar çıkabilmektedir.

c. Ara Bağlantılar (Interconnections): Bu birimler hem CLB'ler arasında hem de CLB'ler ile IOB'ler arasında bağlantıları yapılandırmada kullanılırlar. Programlanabilir olduklarından çok esnek bir yapıya sahiptirler (Koyuncu, 2008).

Yatay ve dikey kanalların birleştiği yerlerde matris anahtarlar (switch box) vardır. Bu anahtarların içinde altı transistorlu (pass transistor) yönlendirme mekanizması bulunur. Şekil 2.15'de görüldüğü gibi, giriş yapılan taraftan kendisine komşu diğer üç tarafa yönlendirilmeyi sağlayacak programlanabilir anahtarlar vardır.



Şekil 2.14: FPGA genel yapısı (Selim ve Ulucan, 2008)



Şekil 2.15: Matris anahtarların yapısı (Selim ve Ulucan, 2008)

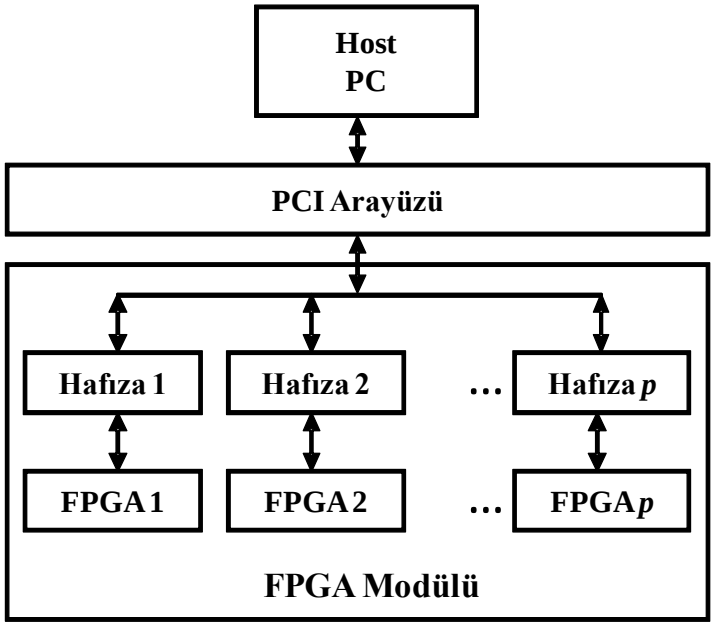
2.2.2. FPGA Tabanlı Özel Amaçlı Bilgisayarlar

F-CCMs (FPGA-based Custom Computing Machines - FPGA tabanlı Özel Amaçlı Bilgisayarlar), donanım ve yazılımı bir araya getiren özel veri işleme platformlarıdır (Şahin ve Gloster, 2005). Genellikle bir F-CCM bir adet genel amaçlı bilgisayar ve bu bilgisayara bağlı üzerinde bir ya da daha fazla FPGA ve hafıza yongaları bulunduran elektronik karttan oluşur (Şahin ve Gloster, 2005a), (Gloster ve Şahin, 2001), (Şahin ve diğ., 2000). Şekil 2.16'da örnek bir FPGA kartı görülmektedir. Bu sistemler genel amaçlı işlemcilerin sunduğu programlanabilme özelliği ile FPGA yongalarının sağladığı hız avantajının bir araya getirildiği sistemlerdir (Rincon, Teres, 1998).

Uygulama programları F-CCM'lere uyarlanırken, programın yüksek işlemci gücü gerektiren bölümleri özel olarak tasarlanmış donanım modülü kullanılarak FPGA yongaları üzerinde çalıştırılır ve bu sayede programın daha hızlı çalışması sağlanır. İyi tasarlanmış bir F-CCM ve donanım modülü sayesinde, uygulama programlarının yoğun Central Processing Unit-İşlemci (CPU) gerektiren bölümlerini işlem türüne bağlı olarak 100 veya daha fazla kez hızlandırmak mümkündür (Şahin, 2002), (Koyuncu, 2008).

Şekil 2.16: RC sistemde kullanılan FPGA kartı (Anon, 2011f)

F-CCM'ler Reconfigurable Computing-Tekrar Yapılandırılabilir Hesaplamalar (RC) sistemi olarak da bilinir. Şekil 2.17'de RC sistemlerin genel yapısı görülmektedir. Bu yapıda *p* adet FPGA yongası ve bunlara ait yerel hafızalar bulunur. FPGA kartı ile bilgisayar arasındaki veri iletişimi, bir Peripheral Compenent Interconnect - Çevre Birimleri Bağlantı Yolu (PCI) ara yüzü üzerinden yapılır. FPGA kartlarının kullanımı ise şu şekildedir: İlk olarak, daha önceden tasarlanmış donanım modülünün yapılandırma bilgileri PCI yolu üzerinden FPGA yongalarına yüklenir. Ardından, yine PCI üzerinden, işlenecek veri kart üzerindeki yerel hafıza ünitelerine aktarılır. Bu aşamadan sonra FPGA yongalarına özel bir başlat sinyali gönderilerek, yongaların hafızadaki veriyi işlemesi sağlanır. Yongalar bu verileri işlerken, ana bilgisayar diğer işlerine devam eder. Yongalar işlemi bitirince, bilgisayara bir kesme sinyali göndererek işlenmiş verinin hazır olduğunu bildirir. Son olarak bilgisayar yine PCI üzerinden FPGA kartı üzerindeki hafızalara erişerek işlenmiş veriyi alır (Şahin ve Gloster, 2005a). FPGA yerel hafızası ile bilgisayar hafızası arasında hızlı veri iletişimini gerçekleştirmek için DMA (Direct Memory Access-Doğrudan Hafıza Erişimi) kullanır.



Şekil 2.17: Tipik bir RC sistem genel yapısı (Şahin, Gloster, 2005a)

2.2.3. FPGA Yongaların Avantajları

Mimarilerine göre sayısal yarı-iletken devreler özel amaçlı, genel amaçlı ve tekrar düzenlenebilir mimariler olarak sınıflandırılabilirler.

1. Özel Amaçlı Mimariler [ASIC (Application-Specific Integrated Circuits-Uygulamaya Özel Tüm devre)]: 1980'lerde sayısal tüm devre teknolojileri arasında bir boşluğun olduğu fark edilmişti. Bir tarafta büyük ve karmaşık fonksiyonları gerçekleyemeyen SPLD ve CPLD gibi programlanabilen ve hızlı tasarım sürelerine sahip mantıksal elemanlar bulunuyordu. Diğer tarafta da tasarım süreleri daha uzun olan, çok büyük ve karmaşık fonksiyonları gerçekleyebilen, uygulamaya yönelik tüm devreler (ASIC) vardı. ASIC'ler belirli bir uygulamaya özel olarak tasarlanırlar, bundan dolayı farklı uygulamalar için düşük performansa sahiptirler. Gereksinimler değişmişse ASIC donanımının yeniden geliştirilmesine ihtiyaç duyulur. ASIC'ler, işletim hızında artışa, düşük enerji tüketimine ve yüksek başarı performansına sahiptirler. Diğer taraftan esnek (flexible) değildirler ve ilk üretim maliyetleri çok yüksektir. Fazla üretimde maliyetleri düşer (Özdamar, 2007).

2. Genel Amaçlı Mimariler (İşlemci): Özel komut setine sahip ve problemlerin çözümü için bu komut setinde yazılmış programları işletebilen sayısal yarı-

iletken devrelerdir. İşlemciler herhangi bir probleme göre değil genel amaçlı tasarlanmışlardır. Programlanabilir olduklarından esnektirler ama bazen basit bir problemi çözmek için yüzlerce ya da binlerce komut işlemek zorunda olduklarından her zaman istenen performansı gösteremezler (Özdamar, 2007).

3. Tekrar Düzenlenebilir Mimariler (SPLD (Simple Programmable Logic Device), CPLD (Complex Programmable Logic Devices), FPGA): Donanım temelli mimari olarak yüksek performansa, genel amaçlı mimarilerdeki gibi esnekliğe ve programlanabilen sayısal elemanlara sahiptirler. Donanım statik olmayıp dinamik olduğundan, farklı uygulamalara adapte olabilirler (Özdamar, 2007).

ASIC mimarisine göre tekrar düzenlenebilir mimarilerde devreler yavaş ve daha büyüktür. Tekrar düzenlenebilir mimarilerde yapılandırma bilgisi yonga üzerinde depolanmak zorunda olmasına rağmen tekrar tekrar programlanabilir, üzerinde çalıştırdığı algoritma güncellenebilir, hatalar düzeltilebilir. Ayrıca aşamalı hesaplamalarda daha yüksek başarıya sahiptir. ASIC'lerde tasarımlar üretime geçmeden önce fonksiyonel olarak FPGA'lar üzerinde gerçeklenerek test edilirler.

Tekrar düzenlenebilir mimariler, mikroişlemciler gibi işlemleri adım adım değil paralel (aynı zamanlı) yapabilmektedirler. Tekrar düzenlenebilir mimarilerden FPGA yongaları paralel bilgisayarlara, hatta özel tasarlanmış grafik kartlarına göre daha düşük maliyete sahip olduğundan daha ucuzdurlar ve daha yüksek performansa sahiptirler.

FPGA'ların diğer bir alternatifi olan SPLD'ler ve CPLD'ler de mantık kapı sayısı daha azdır. Modern FPGA'lar da bütün ya da kısmi sistemlerin yeniden yapılandırılması anında (on the fly) tasarım değişimi ve sistemlerin yükseltilmeleri mümkündür. Bazı FPGA'ların bir parçası yeniden yapılandırılırken diğer kısımları çalışmaya devam edebilmektedirler (Özden, 2010).

Bazı hız gerektiren özel uygulamalarda özel bütünleşmiş (entegre) tasarımlarına ihtiyaç duyulmaktadır. Bunlara VLSI (Very Large Scale Integration) tasarım denilmektedir. VLSI, sadece tek bir uygulamaya özel, zaman alan ve pahalı bir üretim olduğundan önce FPGA'lar üzerinde test edilir ve istenen sonuçların alınmasından sonra üretilirler (Selim ve Ulucan, 2008).

FPGA'nın Üstün Kabiliyetleri: Çok hızlı tasarım yapılabilmesi, sonuçları çok hızlı analiz edilebilmesi, kullanıma hazır olması, yeniden düzenlenebilir olması, düşük geliştirme maliyeti ve paralel çalışabilmesidir. Örneğin, XC4000 serisinde tekrar düzenleme zamanı FPGA aygıtının yoğunluğuna göre 1-50 mS (milisaniye) aralığında değişmektedir.

FPGA yongaları, hızlı oldukları için çok büyük avantaj sağlarlar. Örneğin, ASIC yongaları, uzun süre gerektiren çeşitli tasarım ve üretim aşamalarından geçtikten sonra ancak kullanıma hazır hale gelebilmektedirler ve tasarım esnasında yapılan bir hatadan dolayı kullanılmaz hale gelmektedirler. Fakat FPGA yongaları kullanılarak, tasarlanan ürünün FPGA yongalarına yüklenmesi için geçen kısa sürede işlem gerçekleştirilebilmektedir. Ayrıca, FPGA yongalarında, yeniden programlanabilme özelliğinden dolayı, oluşan bir hata çok kısa bir sürede ve ekonomik bir zarara uğranılmadan telafi edilebilmektedir. Yongaları programlama işlemi, boyutlarına ve tasarıma bağlı olarak milisaniyeler içinde gerçekleştirilebilmektedir. FPGA yongaları sınırsız olarak tekrar programlanabilirler. FPGA'ların esnekliği paralel birim artışına izin vermektedir. Bu da çalışma performansını artırmakta ve maliyeti düşürmektedir. Daha kısa zamanda yapılması gereken yoğun işlemler önceden yazılımlarla yapılmasına rağmen şimdi yazılımlar yerine FPGA'lar tercih edilmektedirler. Yongaları yapılandırmak için tasarlanan donanım modülleri, sadece bir özel problemin çözümü için tasarlandıklarından yazılıma göre çok hızlı çalışmaktadırlar. Ayrıca birden fazla FPGA yongası birbiri ile paralel olarak çalışabildiği hatta tek bir yonga içerisinde birden fazla modül yerleştirilebildiği için yazılıma göre çok yüksek hız kazançları elde edilebilmektedir (Şahin ve Gloster, 2005a). FPGA yongaları üretici firmaları arasında, Altera, Xilinx, Flex, Lucent, Actel sayılabilir. Bu firmalar kendi üretmiş oldukları yongalara özel isimler vermektedirler. Örneğin Xilinx firması, üretmiş olduğu yongalara Spartan, Virtex (Anon 2011c) gibi isimler verirken, Altera firması ürettiği yongalara Cyclone ve Stratix (Anon 2011d) gibi isimler vermektedir.

2.2.4. FPGA Uygulamaları

FPGA'lar ilk başta ASIC tasarımların ilk örneklerini oluşturmak veya yeni bir algoritmanın fiziksel gerçeklenebilirliğini doğrulamak için donanım ortamı sağlamak amacıyla geliştirildiler. Bununla birlikte, geliştirme maliyetlerinin düşük olması ve kısa sürede pazara sunulabilme özelliklerinden dolayı son ürün yelpazesindeki yerleri zanla

arttı. 2000'lerin ilk yıllarında milyonlarca kapı içeren yüksek performanslı FPGA'lar piyasada yerlerini aldılar. Bazıları gömülü mikroişlemci çekirdekleri, yüksek hızlı Giriş/Çıkış (I/O) ara yüzleri, gömülü RAM ve DSP öbekleri sunmaktadır. Günümüz FPGA'ları beş ana pazar kolunda yer bulmaktadırlar:

1. ASIC ve Custom Silicon (Genel Silikon): ASIC ve Custom Silicon, büyük hacimli yongaların kullanımını olanak veren fakat uzun tasarım süreci gerektiren yöntemlerdir. Bundan dolayı bu alanda test ve prototipleşme gibi çeşitli tasarımlarda FPGA kullanımı giderek artmaktadır (Aydın, 2008).

2. DSP (Digital Signal Processing - Sayısal Sinyal İşleme): DSP uygulamalarına özel entegrelerin bulunmasına rağmen, bugünün FPGA'ları, üzerlerindeki gömülü mikroişlemciler, DSP işlemlerini kolaylaştıracak çarpıcılar, atanmış aritmetik birimler ve yonga üzerindeki büyük miktardaki RAM'lar ile DSP işlemlerini yapabilmektedirler (Anon, 2011e). Bu özellikler, FPGA'larca sağlanan yoğun paralel işlem yeteneğiyle birleştiğinde sonuç en hızlı DSP yongasından 500 kez ve daha fazla hızlı olabilmektedir. Örneğin, Vestel Pixellence (Aydın, 2008), (Selim ve Ulucan, 2008).

3. Embedded Microcontroller (Gömülü Mikro Denetleyiciler): Küçük kontrol fonksiyonları genellikle özel amaçlı mikro denetleyiciler ile çözümlenir. FPGA'ların fiyatlarının düşük olması, en küçük modellerinin bile özel I/O fonksiyonlarını içeren işlem çekirdeklerini sağlayabilecek kapasitede olmaları ve yüksek işlem performanslarından dolayı tercih edilirler. Örneğin, FFT (Hızlı Fourier Dönüşümü) ve konvolüsyon işlemleri gibi yüksek performans gerektiren hesaplamalarda bir mikroişlemci yerine FPGA'lar kullanılmaktadır (Aydın, 2008).

4. Fiziksel Katman Haberleşmeleri (Radyo Haberleşme Sistemi): FPGA'lar fiziksel katman ile yüksek seviyeli haberleşme protokol katmanları arasında ara yüz bağlantısını gerçeklemektedirler (Glue Logic). Son teknoloji FPGA'lar da haberleşme ve ağ oluşturma fonksiyonlarını tek cihazda birleştiren yüksek hızlı alıcı-verici birimleri bulunabilmektedir (Aydın, 2008).

5. RC (Reconfigurable Computing - Yeniden Düzenlenebilir Hesaplama): FPGA'ların hesaplama işlemlerinde kullanımı reconfigurable computing olarak bilinir. RC'ler FPGA'larca sağlanan içsel paralellik ve tekrar yapılandırılabilirliği sağlayan "hardware accelerate - donanım hızlandırma" yazılım algoritmalarını kullanmaktadırlar. Çeşitli firmalar halen yeni çözümler keşfetmek amacıyla donanım benzetimlerinden şifreleme tahlillerine (şifreleme algoritmalarında özellikle kod kırma işlemlerinde bütün olası senaryoları deneme durumlarında kullanılmaktadır) ve ilaç keşiflerine kadar büyük FPGA temelli tekrar yapılandırılabilir hesaplama tertibatları hazırlamaktadırlar. Bu alanda şirketler algoritmalarını yazılımdan FPGA'ya geçirerek algoritmaların işleyişlerini hızlandırmaktadırlar (Aydın, 2005).

Ayrıca savunma sistemlerinde sınırlı sayıda üretilen tasarımlar için ASIC oldukça pahalıdır. Bu alanlarda FPGA tercih edilir. Halen ASELSAN da dâhil olmak üzere, savunma sektörüne çalışan birçok firma FPGA kullanmaktadır (Selim ve Ulucan, 2008).

FPGA Üzerinde CPU Tasarımı: FPGA'lar CPU tasarımlarıyla farklı sistemlerin içinde gömülü olarak kullanılmaktadır. Bu farklı sistemlerde FPGA'ların hızlı, güvenilir olma özelliklerinden ve paralel işlem yapabilme yeteneklerinden faydalanılmaktadır. ESA tarafından tasarlanan açık kaynak kodlu LEON işlemcisi uzay araçlarında kullanılmış bir FPGA tabanlı CPU tasarımıdır.

FPGA'lar aynı zamanda süper bilgisayar yapımında da kullanılmaktadır. Starbridge firmasının ürettiği ve HAL-15 isimli hiper bilgisayar NASA tarafından kullanılmaktadır. Bu ürün birçok yönüyle bilinen diğer süper bilgisayarlardan ayrılmaktadır (Selim ve Ulucan, 2008).

FPGA uygulamaları; medikal resimleme, robotik, ses tanıma, şifreleme, biyoinformatik, bilgisayar donanım emülasyon, havacılık ve uzay teknolojileri, taşıt teknolojileri gibi alanlarda kullanılmaktadır. FPGA başlangıçta CPLD'lerin rakibi olarak başladı ve benzeri bir alanda mücadele etti. Büyüklükleri, kapasiteleri ve hızları attıkça pazarda SoC (full Systems on a Chips) yaklaşımı ile daha da fazla yer almaya başladı. Günümüzde SoC teknolojisi özellikle yer ve enerji sorunlarını çözmede kullanılır.

Tasarımlarının zorluğundan dolayı FPGA'ların yüksek performans hesaplamalarında kabul görmeleri halen belli bir limittedir (Özden, 2010).

2.2.5. FPGA Tasarım Ve Programlama

Şekil 2.18'de görüldüğü gibi FPGA uygulama süreci istenen tasarımın bir HDL (Hardware Description Language–Donanım Tanımlama Dili) dilinde kodlanması ile başlar.

Birçok HDL dili bulunmasına rağmen yaygın olarak VHDL (Very High Speed Integrated Circuit HDL-Çok Yüksek Hızlı Tümleşik Devreli Donanım Tanımlama Dili) veya Verilog kullanılır.

Diğer donanım tanımlama dillerinden bazıları şunlardır; ABEL (Advanced Boolean Expression Language), AHDL (Altera Hardware Description Language), AHPL (A Hardware Programming Language), CDL (Computer Design Language), CONLAN (CONsensus LANguage), IDL (Interactive Design Language), ISPS (Instruction Set Processor Specification), TEGAS (TEst Generation And Simulation) ve TI-HDL (Texas Instruments Hardware Description Language)'dir (Selim ve Ulucan, 2008).

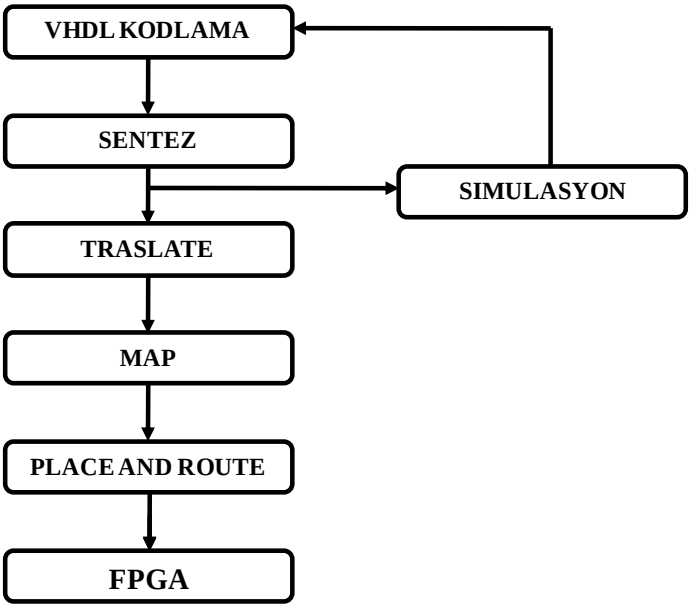
Ardından yazılan HDL kodu uygun tasarım aracıyla sentezlenir (mantık devre karşılığı oluşturulur) ve benzetimi gerçekleştirilir.

Benzetimde istenen sonuçlar alınırsa sentez işlemi ile oluşturulan netlist, çeviri (translate) işlemi ile hedef FPGA yongasına uygun formata dönüştürülür.

Bağla (Map) işleminde netlistte bulunan devre elemanları ile FGA yongasındaki devreler eşlenir.

Son olarak eşleşen bu elemanlar place-route (yerleştir-bağla) işlemi FPGA'da en uygun (optimal) şekilde yerleşimi yapılır ve bunlar arasındaki bağlantılar tamamlanır.

Bu işlemler için genellikle FPGA yongasını üreten firma tarafından geliştirilmiş EDA (Electronic Design Automation- Otomatik Elektronik Tasarım Aracı) yazılım araçları kullanılır. Bizim çalışmamızda Xilinx firması tarafından geliştirilen ISE (Integrated Software Environment-Entegre Yazılım Ortamı) sürümü (versiyonu) 12.1 kullanılmıştır.



Şekil 2.18: VHDL ile tasarım adımları

2.3. FPGA'DA YSA UYGULAMALARI

FPGA tabanlı özel tasarlanmış modüller bilimsel alanda yüksek performans, esneklik ve düşük maliyet gibi önemli parametreleri elinde bulundurmakla büyük ilgi çekmiştir.

Çalışmamızdaki temel amaç YSA'nın FPGA'ya uygulanmasında uzman gereksinimini ve uygulama süresini azaltmak için veri yolu tasarımını otomatik yapabilecek araç geliştirmek olduğu için yapılan literatür taraması da bu yönde olmuştur. Bu bölümde bu konuda yapılmış belli başlı çalışmalar özet olarak sunulmuştur.

Yapay Sinir Ağı Eğitiminin IEEE 754 Kayan Noktalı Sayı Formatı İle FPGA Tabanlı Gerçeklenmesi: Çavuşoğlu ve diğ. (2008) yayınladıkları bu bildiride, ileri beslemeli iki katmanlı bir YSA'nın geriye yayılım ile eğitiminin FPGA üzerinde gerçeklenmesi ÖZEL-VEYA problemi temel alınarak anlatılmıştır. Çalışmada, YSA eğitim işlemlerinin [Çok Katmanlı Algılayıcı (ÇKA) ve Geriye Yayılım (GY)] FPGA üzerinde paralel gerçeklenmesi özellikle sağlanmıştır. Eğitimin gerçeklenmesinde Xilinx FPGA ailesine ait 2vp30fg676-7 yongası kullanılmıştır. Çalışma sonucunda az yer kaplayan ve çıkışta elde edilen hata değerinin önemsenmeyecek kadar küçük olduğu bir eğitim gerçeklenmiştir.

Plaka Yeri Tespiti İçin Kenar Bulma, Bit Tabanlı Öznitelik Çıkartma ve YSA Sınıflandırıcısının FPGA Üzerine Uyarlanması: Çavuşlu ve diğ. (2008a), görüntü işlem ve örüntü tanıma problemleri gibi yoğun işlem yükü gerektiren işlevlerin gerçek zamanlı olarak gömülü sistemler üzerinde uyarlanmasının yarı-iletken teknolojilerinde meydana gelen son yıllardaki gelişmeler ile mümkün olabildiğini görmüşlerdir. Bu çalışmada plaka yeri tespit problemi için gömülü sistem alt birimlerinin paralel işlem yeteneğine sahip FPGA üzerine uyarlanması gerçekleştirilmiştir. Geliştirilen alt birimler görüntü işlem ve örüntü tanıma uygulamalarında yoğun olarak kullanılmakta olan görüntü işlem ve sınıflandırıcı alt birimleridir. Bu alt birimler kenar bulma, istatistiksel bit-tabanlı öznitelik çıkarma ve YSA kullanılarak bir resim üzerinde plaka yerinin belirlenmesi işlevlerini başarı ile gerçekleştirmektedir. YSA'nın FPGA ile gerçeklenmesinde sabit noktalı sayı ve kayan noktalı sayı formatlarında ve farklı bit uzunluklarında işlem yapabilen on iki çarpma, on dört toplama ve iki bölme modülü kullanılmıştır. Resim üzerinde ilk olarak kenar bulma işlemi için Sobel süzgeci kullanılmıştır. İşlenmiş imge bit tabanlı öznitelik çıkartma aracı ile işlenerek elde edilen

öznitelikler YSA girişi olarak sunulmuştur. YSA eğitiminde toplam dört yüz doksan sekiz örnek kullanılmıştır. Bu örneklerin yarısı plaka içeren pozitif örnek diğer yarısı da plaka içermeyen negatif örnek olacak şekilde ayarlanmıştır. YSA donanım üzerinde gerçeklenmesinde paralel bir yapı kullanılmıştır. Ayrıca sabit nokta sayı formatında işlem yapılması durumunda FPGA kaynaklarının kullanımında çok ciddi bir kazanım elde edilir iken, YSA sınıflandırıcısının eğitim başarısında ihmal edilebilecek oranda bir bozulma oluştuğu görülmüştür. Yapılan çalışmada bilgisayar üzerinden FPGA'ya gönderilen imgelerle donanım üzerinde plaka yeri bulma uygulaması gerçekleştirilmiştir (Çavuşlu ve diğ., 2008a).

FPGA Tabanlı Aritmi Sınıflandırıcı: Bu çalışmayı Özdemir (2009) yapmıştır. EKG (Elektro Kardiyo Gram) kayıtlarının yazılım tabanlı uzman sistemler tarafından yorumlanması 1960'lı yıllara dayanmaktadır. Son yirmi yıldır bu konuda birçok teknik üzerine çalışmalar yapılmıştır. Fakat güçlü tahminsel yeteneğine olan inanç sebebiyle, medikal tanı destek sistemi uygulamalarında kullanılan en popüler yöntem YSA olmuştur. Literatürde önerilen YSA modelleri, çok karmaşık yazılım tabanlı çözümlerdir ve bunlar gerçek zamanlı çalışmazlar. Bu tür yapıların donanım gerçeklemeleri ise ancak pahalı işlemciler üzerinde yapılabilmektedir. YSA donanım gerçeklemelerini taşınabilir ucuz cihazlar üzerinde oluşturmayı mümkün kılmak için, YSA girişine uygulanan EKG işaretinin özellik sayısı azaltılmalıdır. Bu sayede daha az sayıda bilgi ile daha basit bir mimari oluşturmak mümkün olacaktır. Bu çalışmada YSA'nın öğrenme hatası %5 gibi kabul edilebilir bir seviyede tutularak, TBA (Temel Bileşen Analizi) yöntemi kullanılarak EKG giriş işareti özellik sayısı önemli ölçüde azaltılmıştır. Bu sayede 8x2x1 boyutlu basit bir Matlab YSA modeli FPGA donanımı üzerinde IEE-754 32-bit kayan noktalı nümerik sayı formatı kullanılarak gerçekleştirilmiştir.

Çok Katmanlı Algılayıcı Yapısının FPGA İle Gerçeklenmesi: Özden'in (2010) bu çalışmasında, 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 birçok 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.

Dokuz 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.

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 yapılan tasarım FPGA yongasına gömülerek çok katmanlı algılayıcı örneği gerçeklenmiştir (Özden, 2010).

Sıradan Gaz Tanımlama Sistemleri İle Endüstriyel Gazlardan Bir Kısmını Sınıflandırabilmek: Bu çalışma sıradan gaz tanımlama sistemleri ile endüstriyel gazlardan bir kısmını sınıflandırabilmek için tasarımlanmış (dizayn edilmiş) ve benzetimleri yapılabilmiştir. Elektronik burun, programlanabilir mantık araç yongalarında farklı seçicilik ailelerine, sinyal toplama, aile tanımlama ve karar verme bölümlerine sahip SnO_2 den yapılmış ince film şeklinde gaz sensorlarından yapılmış, sekiz küçük portatif sobacık dizisinden oluşur. Çoklu perceptron yapılı geri yayılımlı sinir ağları dizayn edilmiş ve FPGA'ya uygulanmıştır. Sekiz türdeki gaz sensorlarından gelen giriş sinyallerini işlemek için yapılan tasarım VHDL dilinde kodlanmıştır. Elektronik burun endüstri gazlarının beş türünü bilgisayar benzetiminde başarılı olarak sınıflandırmıştır (Benrekia ve diğ., 2009).

Sinir Sistemi Modellerinin Yüksek Performanslı FPGA'ya Gömülme Mimarileri: Bu çalışmada, değişik bilgisayarlarda farklı yazılım programlarıyla modellenmiş olan YSA'ların eş zamanlı çalışma hızı FPGA'ya gömülerek oluşturulan YSA'lardan daha yavaş olduğu tespit edilmiştir. Ancak FPGA'ların paralel işlem yapabilmeleri sinir sistemi modellerinde performansı artırmaktadır. Yine YSA modellemesi yapılabilen kişisel bilgisayar fiyatlarıyla rekabet edebilmesi için FPGA'ların kullanımı tercih edilebilmektedir (Weinstein, Lee, 2006).

Türkçe Fonemlerin Sınıflandırılmasında Kullanılan Sinir Ağının FPGA Yongalarına Uygulanması: A. Uçar'ın çalışmasında Türkçe fonemlerin sınıflandırılmasında kullanılan sinir ağı, FPGA yongası kullanılarak gerçekleştirilmiştir. Gerçek zamanlı hızlara ulaşmanın, sinir ağlarının donanımsal olarak gerçekleştirilmesiyle mümkün olabileceği belirtilmiş ve bu amaçla radyal tabanlı fonksiyon ağı tasarlanarak oluşturulan donanım mimarisi FPGA yongası için sentezlenmiştir. Sentez sonuçları FPGA ile gerçeklenen sistemin, gerçek zamanlı fonem sınıflandırması yapabilen bir sistem olduğunu göstermiştir (Uçar, 2007).

3D Grafik Dönüşümleri İçin 32-Bit Kayan Noktalı Modül Tasarımı: Günümüzde, bilgisayar animasyonlarında, yüzlerce animasyon nesnesi tipik bir animasyon sahnesi tanımlamak için sahnede yerleştirilir ve her nesnenin sahnedeki yerini matematiksel olarak ifade etmek için binlerce köşe kullanılır. Bu tür sahneler için üç boyutlu (3D) dönüşümleri kullanmak büyük miktarda işlemci zamanı gerektirir. Bundan dolayı, bir animasyon sahnesinin hesaplanması uzun zaman alır. Ayrıca, gerçek zamanlı animasyonlarda, dönüşüm hesaplamalarını zamanında yapabilmek neredeyse imkansız hale gelir. Bu çalışmada, 3D grafik dönüşümlerini daha hızlı yapabilmek için FPGA üzerinde çalışabilen 32-bit kayan nokta tabanlı bir donanım modülü tasarlanmıştır. Modülün veri işleme hızı genel amaçlı bilgisayarlar (PC) ile karşılaştırıldığında 3D grafik dönüşümlerinin tasarlanan modül sayesinde FPGA üzerinde daha hızlı yapabildiği görülmüştür (ŞAHİN, 2010).

Güç Elektroniği Uygulamaları İçin FPGA'ya Gömülmüş YSA Tabanlı Denetleyici: Güç elektroniği uygulamalarında yapay sinir sisteminin yazılımlara gömülebilmesi için değişik yaklaşımlar vardır. Yapay sinir ağlarında işlemlerin paralel yapılabilmesi çok önemlidir. Mikroişlemcilerin FPGA'lara gömülmesiyle, mikroişlemciler daha güçlü yazılım seçeneklerine (opsiyonlarına), düşük maliyetlere, işlemleri yüksek hızda yerine getirme, tekrar programlanabilme ve paralel işlem yapabilme özelliklerine sahip olmuşlardır ve bunlar güç elektroniği uygulamalarında YSA tabanlı denetleyici olarak kullanılmışlardır (Bastos ve diğ., 2006).

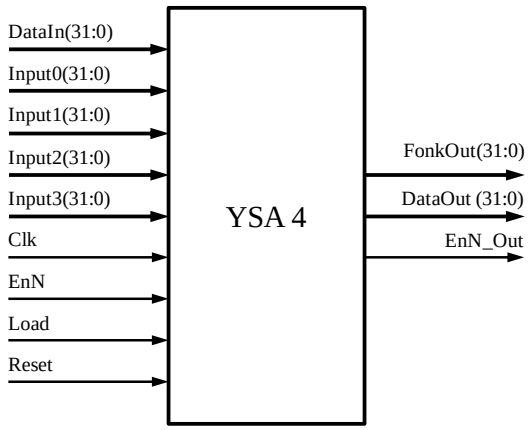
3. MATERYAL VE YÖNTEM

3.1. YSA İÇİN FPGA VERİYOLU TASARIMI VE SİNİR HÜCRESİ (NÖRON) KÜTÜPHANESİ (LIBRARY)

Bu çalışmada bir YSA'nın her katmanında kullanılabilecek değişik sayıda girişlere ve farklı transfer fonksiyonlarına sahip lego şeklinde birbirine bağlanabilen modülerden oluşan nöron kütüphanesi oluşturulmuştur. Kütüphanedeki nöronlar birbirine bağlanarak katmanları, katmanlarda birbirine bağlanarak bir yapay sinir ağını oluşturacak şekilde tasarlanmışlardır.

Tasarlanan modüller bir donanım tanımlama dili olan VHDL'de kodlanmış ve sentez aracı olarak Xilinx ISE 12.1 kullanılmıştır. Modüller, IEEE 745 standartlarına uygun 32-bit kayan noktalı veri işleyecek şekilde tasarlanmıştır.

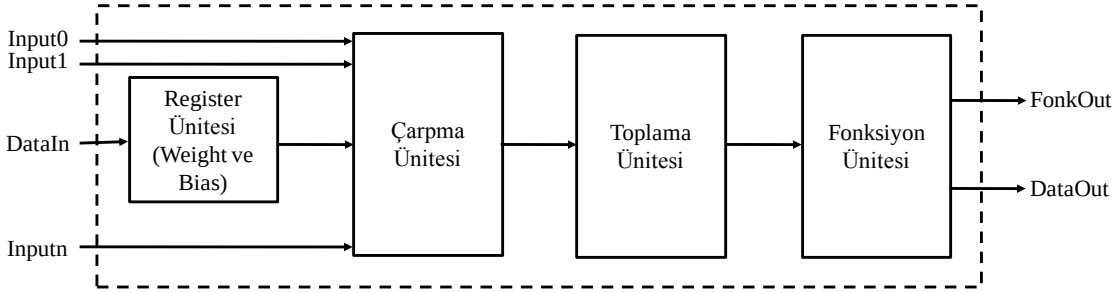
Şekil 3.1'de sayısal olarak tasarlanan dört girişli sinir hücrelerinin en üst seviye standart blok diyagramı görülmektedir. Hücreler üzerinde 32-bitlik DataIn ve DataOut giriş ve çıkışları hücrelerin zincirleme birbirlerine bağlanması ve hücre içindeki ağırlık (weight) ve eşik (bias) değerlerinin tutulduğu registerlerin ilk değerlerinin yüklenmesi için tasarlanmıştır. Input0, Input1, Input2 ve Input3 hücreye giden girdi sinyalleridir. EnN EnN_Out sinyalleri hücreler arasında veri akışını senkronize etmek için kullanılmıştır. Hücreler EnN sinyali aktif olduğunda girişlerdeki Input sinyallerini almakta ve işlemeye başlamakta, çıkışta sonuç hazır olduğunda ise kendinden sonra gelen hücreyi uyarmak için EnN_Out sinyalini aktif etmektedirler. Hücre hesapladığı en son değeri FonkOut çıkışı ile kendinden sonraki hücreye iletmektedir.



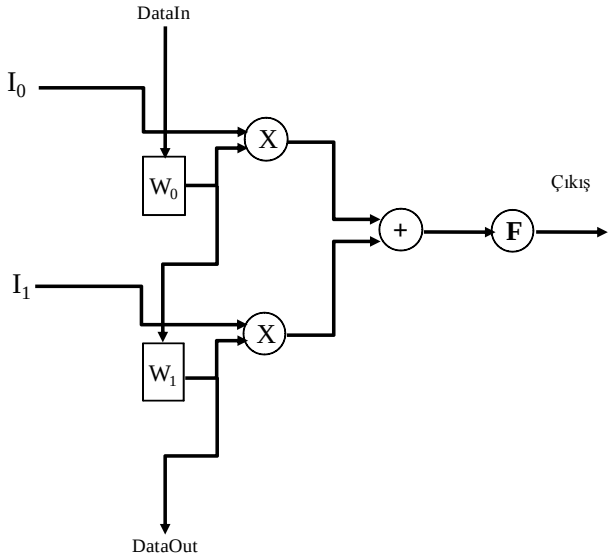
Şekil 3.1: Nöron için genel en üst seviye blok diyagramı

Şekil 3.2'de hücrelerin ikinci seviye blok diyagramı görülmektedir. Hücreler, *Register Ünitesi, Çarpma Ünitesi, Toplama Ünitesi* ve *Fonksiyon Ünitesi* olmak üzere, dört bölümde tasarlanmıştır. *Register Ünitesi* nöronun ağırlık değerlerinin ve eğer hücre eşik değerli olarak tanımlanmış ise eşik değerlerinin tutulması içindir. *Çarpma Ünitesi,* Register ünitesinden gelen ağırlık değerleri ile Girişlerden gelen giriş değerlerinin çarpımını, *Toplama Ünitesi* ise *Çarpma Ünitesinden* gelen değerlerin toplanarak tek bir değer elde edilmesini sağlar. Elde edilen bu değer *Fonksiyon Ünitesine* girdi olarak verilir. *Fonksiyon Ünitesi* kendi transfer fonksiyonlarına göre gelen değeri değerlendirir ve nöronun çıkış değerini oluşturur.

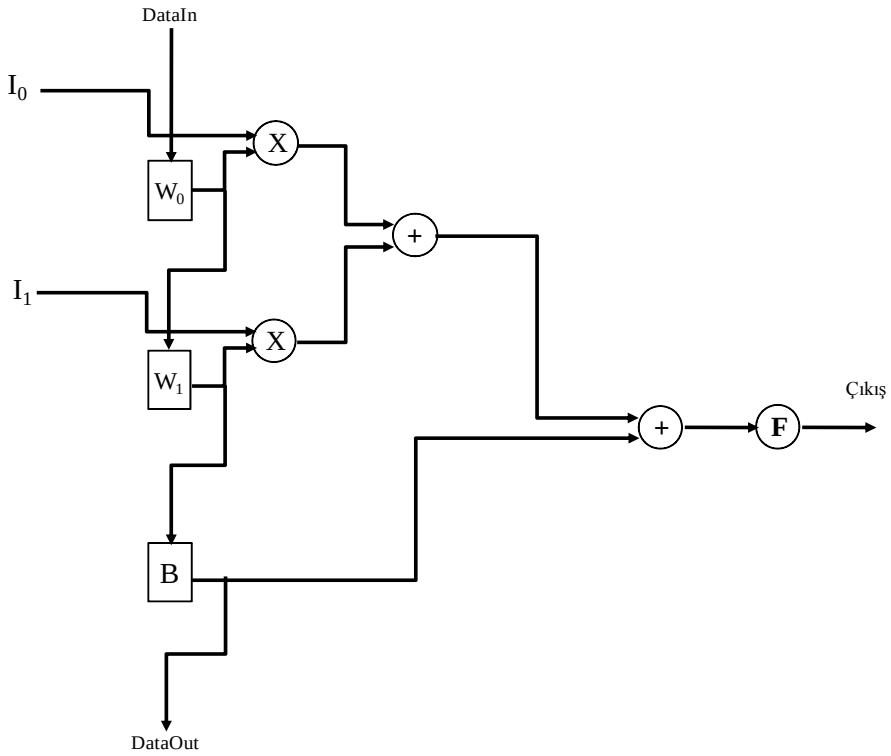
İki girişli sinir hücreleride dört girişli hücrelere benzer şekilde tasarlanmıştır. Bu hücrelerin register ünitesinde eşikli olması ya da olmaması durumuna göre üç ya da iki register bulunmaktadır. Aynı şekilde çarpma ve toplama üniteleri de iki girişten gelen veriyi işleyecek şekilde tasarlanmıştır. Şekil 3.3 ve Şekil 3.4'te normal ve eşikli iki girişli hücrelerin yapısı görülmektedir.



Şekil 3.2: Nöronların ikinci seviye blok diyagramı



Şekil 3.3: FPGA'da normal iki girişli yapay sinir modeli

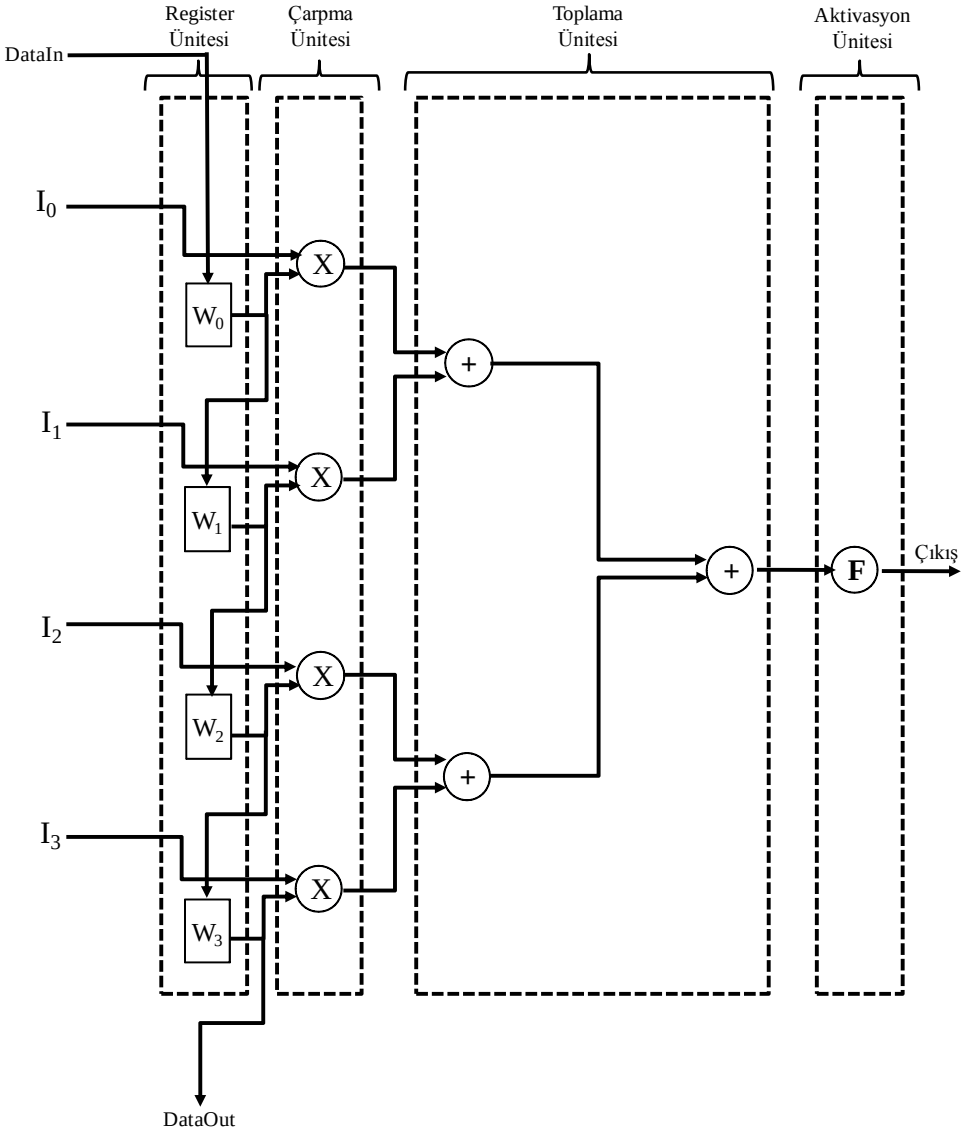


Şekil 3.4: FPGA'da eşikli iki girişli yapay sinir modeli

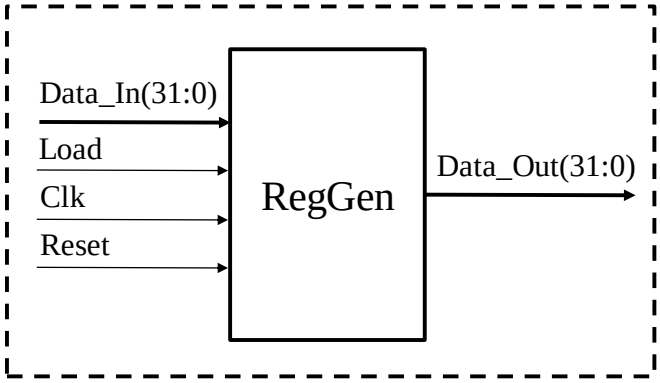
Şekil 3.5'de, dört girişli normal bir nöronun içyapısı görülmektedir. Register Ünitesindeki bütün registerler bir zincir şeklinde birbirine bağlanmıştır. Bu şekildeki bağlantı sayesinde registerlere ilk değerler seri olarak tek bir girişten yüklenebilmektedir. Bu tasarım hem hücrenin giriş sayısını azaltmakta hem de hücre için tasarlanabilecek bir kontrol ünitesini daha basitleştirmektedir.

Eşik'li hücrelerde eşik değerinin tutulduğu registerde ağırlık registerleri ile aynı zincire eklenir. Eşik değerini işleme katmak için toplama ünitesinde ayrıca toplayıcılar yerleştirilmiştir.

Register ünitesinde weight ve eşik değerlerinin tutulması için kullanılan registerler generic register olarak tasarlanmışlardır. Bu registerler generic parametresi sayesinde istenilen uzunluktaki veriyi tutabilecek uzunlukta sentezlenebilmektedir. Şekil 3.6'da tez çalışmasında kullanılan 32 bitlik register görülmektedir. Registerin çalışması ise şu şekildedir: Clk sinyalinin yükselen kenarında eğer Load sinyali 1 ise DataIn daki bilgi okunur. Diğer durumlarda var olan bilgi register içinde saklanır.



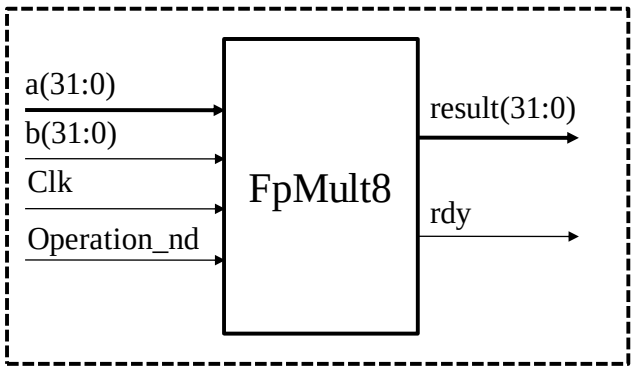
Şekil 3.5: FPGA'da dört girişli normal yapay sinir modeli



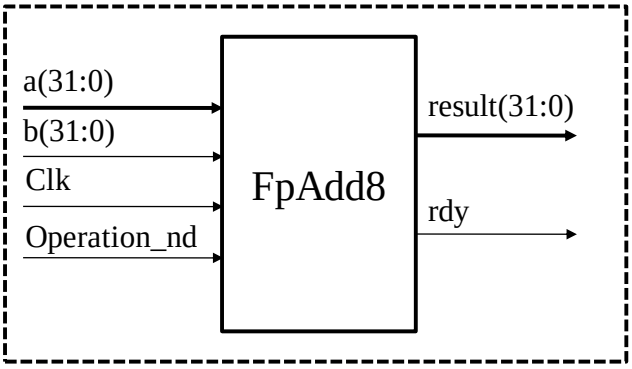
Şekil 3.6: Register

Hücre tasarımlarında kullanılan 32-bit kayan noktalı çarpma ve toplama üniteleri Xilinx'in IP Core Generator (Intellectual Property Core Generator - Fikri mülkiyet çekirdek üretici) tasarım aracı kullanılarak oluşturulmuştur. Şekil 3.7 ve Şekil 3.8'de çarpma ve toplama ünitelerinin blok diyagramları görülmektedir. Her iki ünitede Operation_nd sinyali girişleri hazır olduğunu belirtmek için kullanılır. İşlem sonucu hazır olduğunda ise rdy çıkışı aktif yapılmaktadır. Bu üniteler ağırlık değerleri ile giriş değerlerinin çarpılmasında ve elde edilen çarpımların toplanmasında kullanılmıştır.

Bu çalışma kapsamında giriş adedine, eşik durumuna ve transfer fonksiyonuna bağlı olarak altı değişik nöron modülü tasarlanmış VHDL'de kodlanmış ve test edilmiştir. Tasarlanan ve tasarlanabileceklere örnek modüller Çizelge 3.1'de listelenmiştir. Böylece ANNGEN'i test etmek amaçlı bir örnek nöron kütüphanesi oluşturulmuştur.



Şekil 3.7: Fpmult8 kayan noktalı çarpım ünitesi



Şekil 3.8: Fpadd8 kayan noktalı toplama ünitesi

Çizelge 3.1: Sinir hücresi kütüphanesi

Transfer Fonksiyonu (NeuronType)	Bias (Eşik)	InputCount (Giriş Sayısı)	Transfer Fonksiyonu (NeuronType)	Bias (Eşik)	InputCount (Giriş Sayısı)
*HLIM **	*0*	*2*	TANS	0	2
HLIM	1	2	TANS	1	2
*HLIM **	*0*	*4*	TANS	0	4
HLIM	1	4	TANS	1	4
*HLMS**	*0*	*2*	LOGS	0	2
HLMS	1	2	LOGS	1	2
*HLMS**	*0*	*4*	LOGS	0	4
HLMS	1	4	LOGS	1	4
*PLIN**	*0*	*2*	COMP	0	2
PLIN	1	2	COMP	1	2
*PLIN**	*0*	*4*	COMP	0	4
PLIN	1	4	COMP	1	4

*Hâlihazırda tanımlaması bitmiş modüller

Sinir hücresi kütüphanesi için tasarlanan modüller Xilinx'in Virtex-5 FPGA yongası için sentezlenmiş ve modüllerle ilgili istatistikî bilgiler Çizelge 3.2, Çizelge 3.3, Çizelge 3.4, Çizelge 3.5, Çizelge 3.6 ve Çizelge 3.7'de verilmiştir. Bu istatistiklere göre en yavaş hücre 395 MHz (Megahertz) de çalışabilmektedir. Bunun anlamı bu hücrelerle oluşturulacak herhangi bir YSA yapılandırması saniyede 395 000 000 tane zaman (clk) sinyali üretilebilmektedir. Hücrelerin yonga içinde kapladıkları alana bakılırsa dört girişli hücreler en fazla %23, iki girişli hücreler ise en fazla %10 yer kaplamaktadır. Bu sayılar Virtex-5 ailesinin en küçük yongası olan xc5vlx30 versiyonu için geçerlidir. Bu ailenin en büyük yongası 10 kat daha fazla donanım içerdiğinden bu yüzde değerleri 10 da 1'e kadar inebilmektedir. Dolayısıyla dört girişli hücrelerden bir yonga içine yaklaşık 43 adet iki girişli hücrelerden ise 100 adet sığmaktadır. Bu adet değerleri teorik üst sınırlardır.

Çizelge 3.2: Hardlimnormal2 (Normal sabit limit 2)

Kullanılan Mantık Katmanları	Kullanılan	Kullanılabilir	Kullanılanın Yüzdesi
Register Katman Sayısı	1729	19200	9%
Bakma Tablosu Katman Sayısı	1514	19200	7%
Minimum periyot: 2.528ns (Maksimum frekans: 395.570MHz)			

Çizelge 3.3: Hardlimsnormal2 (Normal simetrik sabit limit 2)

Kullanılan Mantık Katmanları	Kullanılan	Kullanılabilir	Kullanılanın Yüzdesi
Register Katman Sayısı	1,729	19,2	9%
Bakma Tablosu Katman Sayısı	1,514	19,2	7%
Minimum periyot: 2.432ns (Maksimum frekans: 411.184MHz)			

Çizelge 3.4: Purelinnormal2 (Normal doğrusal 2)

Kullanılan Mantık Katmanları	Kullanılan	Kullanılabilir	Kullanılanın Yüzdesi
Register Katman Sayısı	1,975	19,2	10%
Bakma Tablosu Katman Sayısı	1,666	19,2	8%
Minimum periyot: 2.435ns (Maksimum frekans: 410.678MHz)			

Çizelge 3.5: Hardlimnormal4 (Normal sabit limit 4)

Kullanılan Mantık Katmanları	Kullanılan	Kullanılabilir	Kullanılanın Yüzdesi
Register Katman Sayısı	4,248	19,2	22%
Bakma Tablosu Katman Sayısı	3,598	19,2	18%
Minimum periyot: 2.477ns (Maksimum frekans: 403.714MHz)			

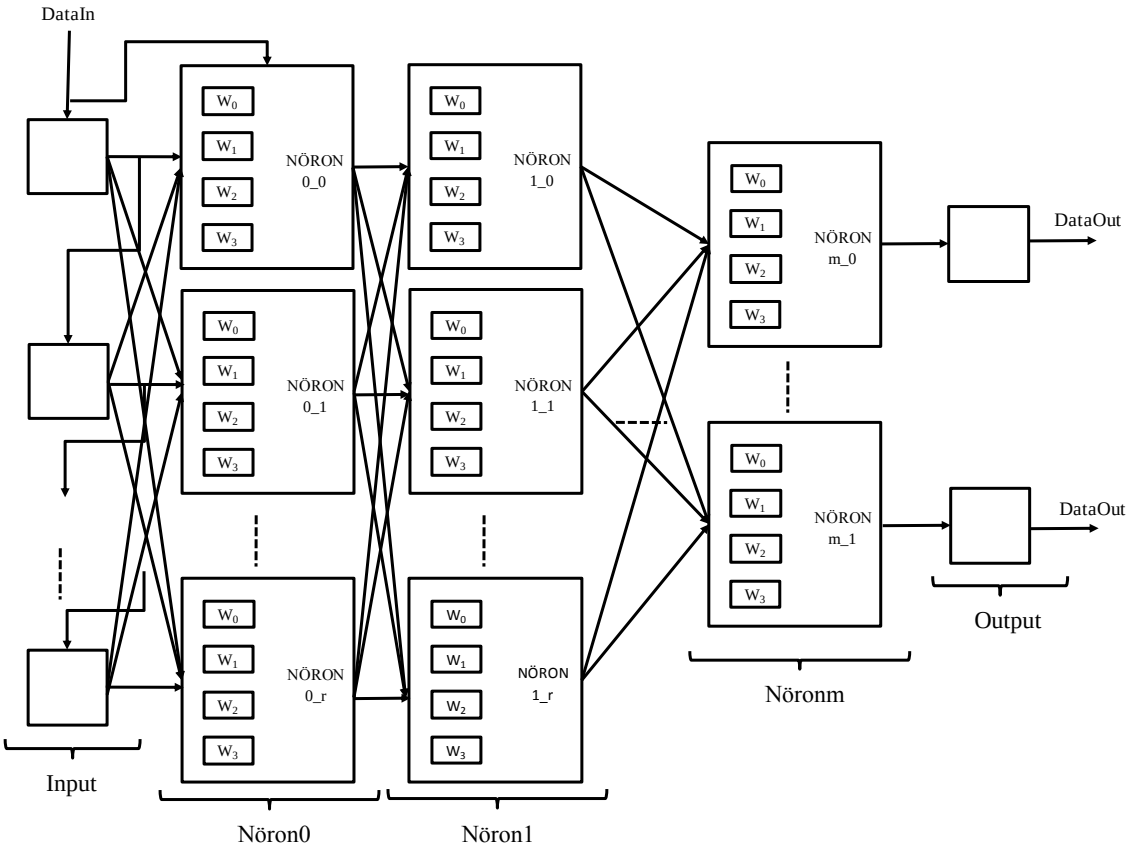
Çizelge 3.6: Hardlimsnormal4 (Normal simetrik sabit limit 4)

Kullanılan Mantık Katmanları	Kullanılan	Kullanılabilir	Kullanılanın Yüzdesi
Register Katman Sayısı	4,248	19,2	22%
Bakma Tablosu Katman Sayısı	3,596	19,2	18%
Minimum periyot: 2.479ns (Maksimum frekans: 403.388MHz)			

Çizelge 3.7: Purelinnormal4 (Normal doğrusal 4)

Kullanılan Mantık Katmanları	Kullanılan	Kullanılabilir	Kullanılanın Yüzdesi
Register Katman Sayısı	4,497	19,2	23%
Bakma Tablosu Katman Sayısı	3,756	19,2	19%
Minimum periyot: 2.516ns (Maksimum frekans: 397.456MHz)			

Bu modülleri lego şeklinde kullanarak *i* girişli, *z* çıkışlı ve *m* katmanlı yapay sinir ağları oluşturulabilmektedir ve Şekil 3.9'da hücrelerin lego parçalı gibi birbirine bağlanarak yapay sinir ağlarını nasıl oluşturulduğu görülmektedir. Burada giriş katmanından gelen verileri geçici olarak saklamak üzere generic registerler kullanılmıştır. Yine bu registerler bir zincir şeklinde birbirine bağlanarak oluşacak veri yolunun giriş sayısı azaltılmıştır. Hücreler içindeki weight (ağırlıklar) ve eğer varsa eşik değerlerini tutan registerlere ilgili değerleri yükleyebilmek için tek bir kanaldan gelen DataIn sinyali ilk katmanın ilk hücresinin data girişine bağlanmış, bu hücreden çıkan DataOut sinyali bir sonraki hücrenin DataIn girişine bağlanarak bir zincir oluşturulmuştur. Bir katmandaki son hücrenin yine DataOut çıkışı bir sonraki katmanın ilk hücresinin DataIn girişine bağlanarak bütün katmanlardaki bütün hücrelerin bu zincire dahil olması sağlanmıştır. Bu sayede yapay sinir modelinde kaç katman, her bir katmanda kaç hücre ve her bir hücrede kaç ağırlık registeri olursa olsun bunların hepsi tek bir veri girişinden seri olarak yüklenebilmektedir.



Şekil 3.9: FPGA'da m katmanlı r sinir hücreli yapay sinir modeli

3.2. ANNGEN (YAPAY SİNİR AĞLARININ VHDL KODUNU OLUŞTURUCU)

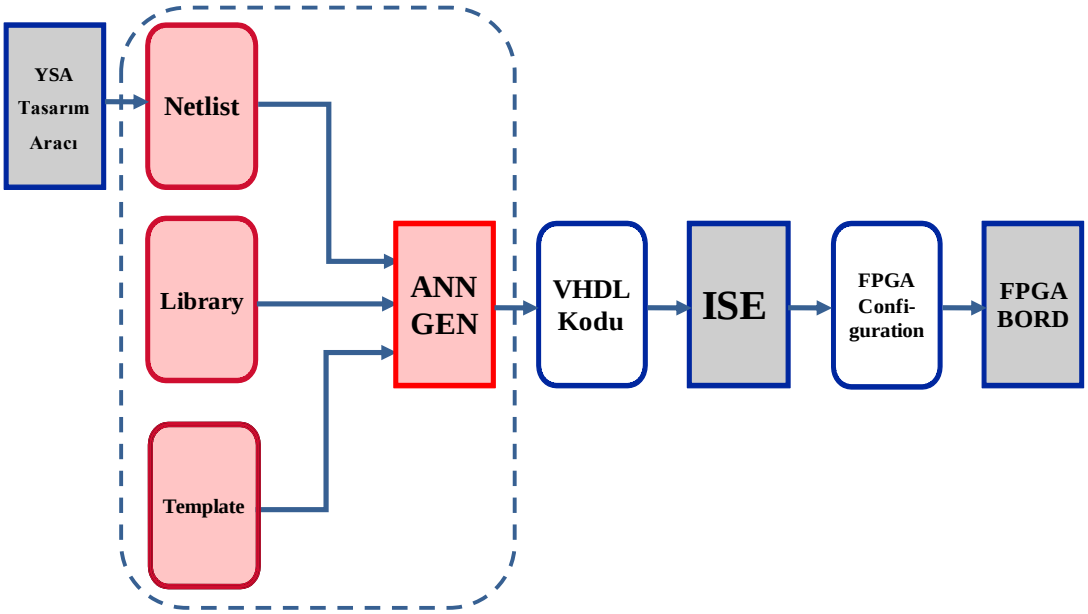
3.2.1. ANNGEN'in Genel Yapısı

Bu bölümde, verilen metin tabanlı Neural Network tanımlamasına göre VHDL kodu üretmek üzere oluşturulmuş ANNGEN yazımı tanıtılacaktır. Şekil 3.10'da ANNGEN'nin FPGA'da YSA tasarımı akış diyagramındaki yeri görülmektedir. ANNGEN girdi olarak üç farklı dosyaya ihtiyaç duyar. Bunlar metin tabanlı YSA tanımlaması (NetList), modül kütüphanesi (Library) ve Şablon (Template) dosyalarıdır.

3.2.2. ANNGEN'in Girdileri

3.2.2.1. NetList

NetList oluşturulacak YSA'nın metin tabanlı tanımlamasını içeren bir metin dosyasıdır. Bu dosya içinde hiyerarşik bir yapıda oluşturulmak istenen YSA ile ilgili bilgiler yer alır. Şekil 3.11'de, NetList dosyasının genel formatı görülmektedir. Bu yapıda öncelikle YSA'nın kaç katmandan oluştuğu belirtilir. Ardından bloklar halinde katmanlar tanımlanır. Her bir katman tanımlamasında o katmanda yer alan hücrelerin türleri, girişleri, eşik durumu, ağırlık bilgileri ve diğer hücrelerle olan bağlantıları tanımlanır. Input, Neuron ve Output olmak üzere üç değişik katman türü tanımlanabilmektedir.

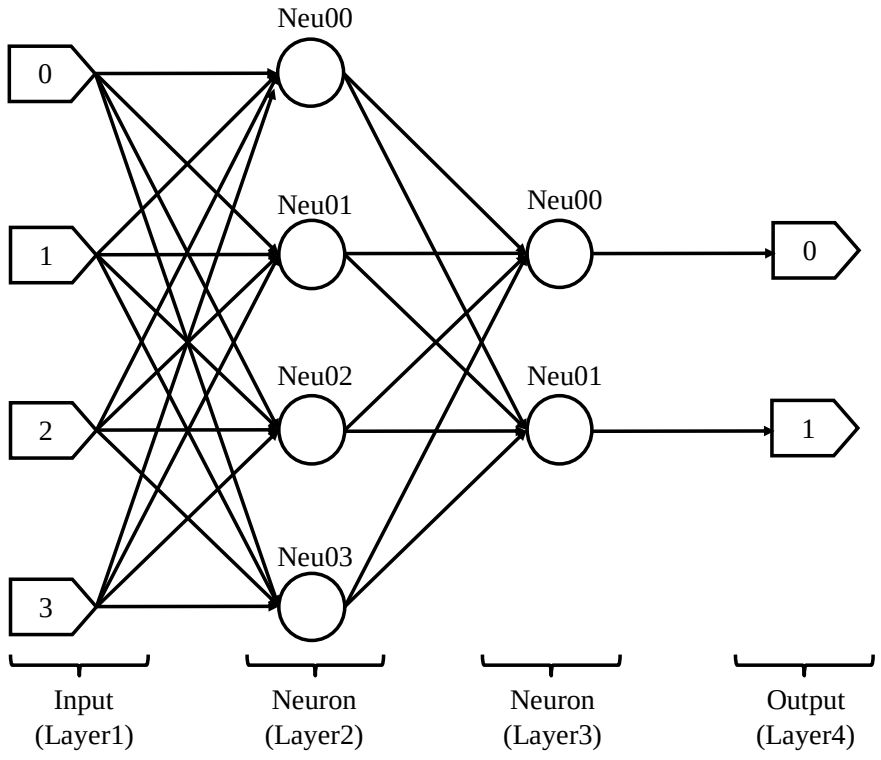


Şekil 3.10: FPGA'ya gömülecek YSA'nın hedeflenen tasarım akışı

```
NETLİST 4<Katman Sayısı>
[
LAYER <Katmanlar> 0 <KatmanNo> INPUT <KatmanTürü> 4 <Katmandaki Element Sayısı >
[
INP00 INP01 INP02 INP03<GirişSayısı>
]
LAYER 1 NEURON 4
[
NEU00 <İsim> PLIN <TransferTürü> 0 <Eşik> 0.0 < EşikAğırlığı>  4 < GirişSayısı>
0<KatmanNo> INP00 1.1 <Ağırlık0>
0 INP01 1.1<Ağırlık 1>
0 INP02 1.1<Ağırlık 2>
0 INP03 1.1<Ağırlık 3>
.
.
.
```

Şekil 3.11: NetList Şablonu

Şekil 3.12 ve Şekil 3.13'de yapay sinir ağının NetList olarak tanımlanmış hali görülmektedir.



Şekil 3.12: Netlist sinir ağı şeması

```
NETLIST 4
[
     LAYER 0 INPUT 4
     [
          INP00 INP01 INP02 INP03
     ]
     LAYER 1 NEURON 4
     [
          NEU00 PLIN 0 0.0 4
               0 INP00 1.1
               0 INP01 1.1
               0 INP02 1.1
               0 INP03 1.1
          NEU01 PLIN 0 0.0 4
               0 INP00 1.1
               0 INP01 1.1
               0 INP02 1.1
               0 INP03 1.1
          NEU02 PLIN 0 0.0 4
               0 INP00 1.1
               0 INP01 1.1
               0 INP02 1.1
               0 INP03 1.1
          NEU03 PLIN 0 0.0 4
               0 INP00 1.1
               0 INP01 1.1
               0 INP02 1.1
               0 INP03 1.1
     ]
     LAYER 2 NEURON 2
     [
          NEU00 PLIN 0 0.0 4
               1 NEU00 1.1
               1 NEU01 1.1
               1 NEU02 1.1
               1 NEU03 1.1
          NEU01 PLIN 0 0.0 4
               1 NEU00 1.1
               1 NEU01 1.1
               1 NEU02 1.1
               1 NEU03 1.1
     ]
     LAYER 3 OUTPUT 2
     [
          OUT00 2 NEU00
          OUT01 2 NEU01
     ]
]
PARAMETERS 4
[
     DataType float
     DataWidth 32
     AddressWidth 32
     VHDLName Deneme
]
```

Şekil 3.13: Örnek NetList

3.2.2.2. Kütüphane (Library)

ANNGEN'in bu girdi dosyası da yine metin tabanlı hâlihazırda var olan hücrelerin ve özelliklerinin listelendiği bir dosyadır. Şekil 3.14'de görüldüğü gibi bu dosyada hücrelerin isimleri, eşik durumları ve giriş adetleri listelenmiştir. Eşik durumu bilgisinin 0 olması hücrenin eşik değersiz yani normal olduğunu, 1 olması ise eşik değerli olduğunu belirtmektedir. ANNGEN bu dosyadan hâlihazırda tanımlaması yapılmış kullanabileceği hücrelerin bilgilerini alır.

3.2.2.3. Şablon (Template)

Bu dosyada ANNGEN'in yeni VHDL kodu yazımında kullandığı çeşitli şablonlar (VHDL kod parçaları) ve ayrıca kütüphanede tanımlanmış hücrelerin VHDL kodları yer almaktadır. Bu bilgiler şablon dosyasına sistematik bir şekilde yerleştirilmiştir. ANNGEN'e yeni hücreler eklenmek istendiğinde bu hücre ile ilgili bilgilerin Kütüphane dosyasına eklenmesi ve hücrenin VHDL kodunun Template dosyasına eklenmesi yeterlidir. ANNGEN otomatik olarak yeni hücreyi tanır.

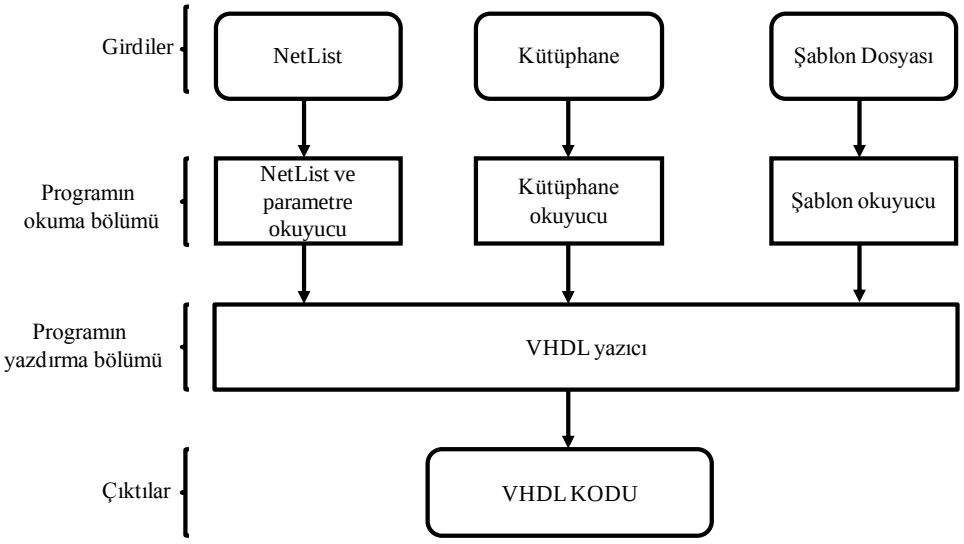
3.2.3. ANNGEN'in Bileşenleri

Şekil 3.15'de görüldüğü gibi ANNGEN dört temel bileşenden oluşmaktadır. Bunlar sırası ile NetList ve Parametre okuyucu, Kütüphane okuyucu, Şablon okuyucu ve VHDL yazıcıdır.

Şekil 3.16'da ANNGEN'in temel çalışma algoritması görülmektedir. Algoritmaya göre önce isimleri verilen NetList, Kütüphane ve Şablon dosyaları okunur ve okunan bu bilgiler ilgili bölüm için tanımlanmış veri yapısında saklanır. Ardından oluşturulmak istenen YSA'nın tanımladığı NetList'teki hücrelerinin tamamının kütüphanede var olup olmadığı kontrol edilir. Bu aşamada eğer NetList'teki herhangi bir sinir hücresinin tanımlaması kütüphanede bulunamazsa, ANNGEN bir uyarı mesajı vererek işlemi sonlandırır. Eğer bütün hücrelerin tanımlaması varsa ANNGEN NetList için VHDL kodunu üreterek sonlanır.

Sinir Hücresinin (Nöron) İsmi	Eşik (Bias) Durumu	Giriş Adedi
HLIM	0	2
HLIM	1	2
HLIM	0	4
HLIM	1	4
HLMS	0	2
HLMS	1	2
HLMS	0	4
HLMS	1	4
PLIN	0	2
PLIN	1	2
PLIN	0	4
PLIN	1	4

Şekil 3.14: Kütüphane (Library) dosyasının formatı



Şekil 3.15: ANNGEN'in genel yapısı

01 Başla.
02 NetList'i oku.
03 Kütüphaneyi oku.
04 Şablon Dosyasını oku.
05 **Eğer** (NetList'teki bütün hücreler kütüphanede varsa)
    VHDL kodunu yaz.
    Değilse
    Uyarı mesajı yaz.
06 Son.

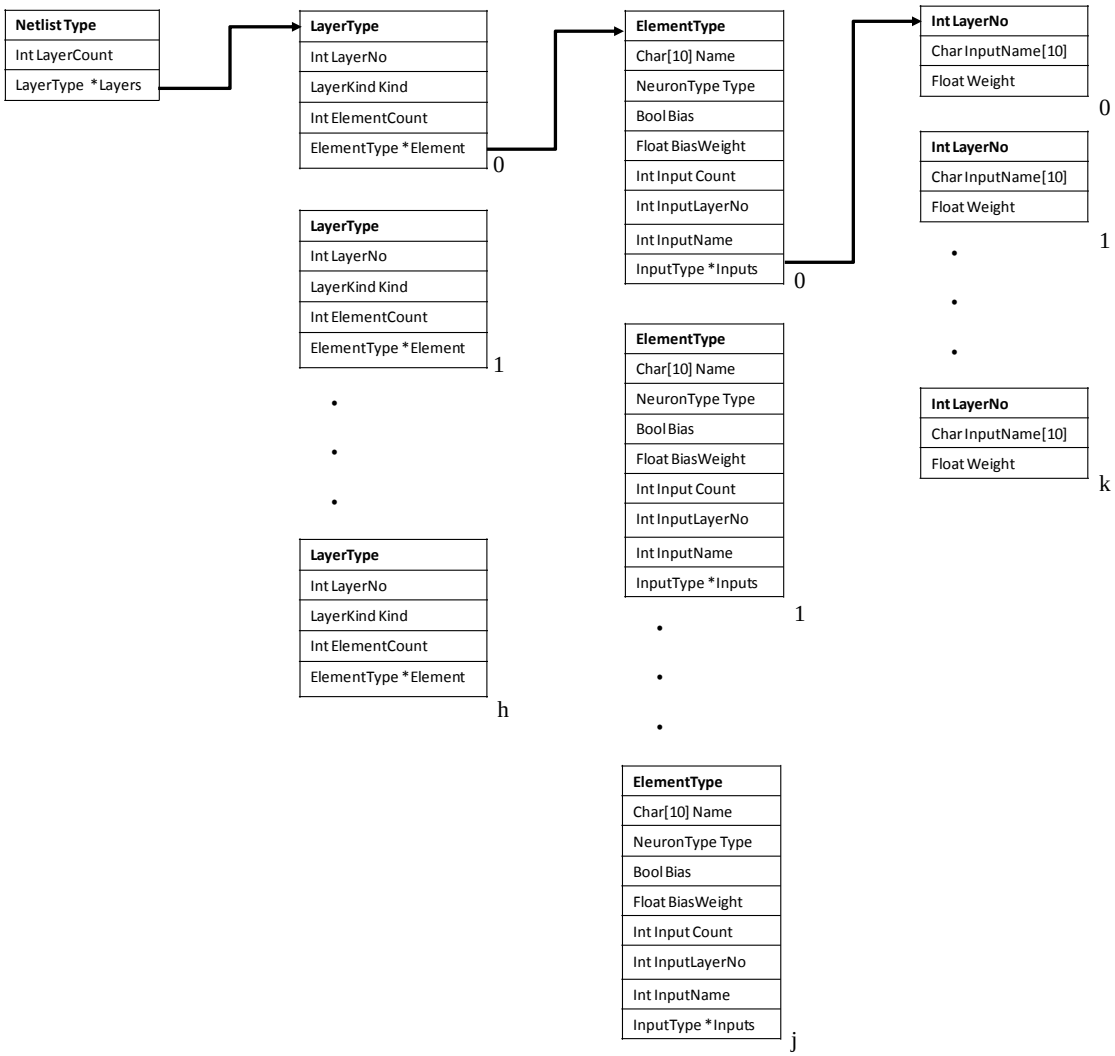
Şekil 3.16: ANNGEN algoritması

3.2.3.1. NetList Okuyucu

NetList okuyucunun görevi verilen NetList dosyasındaki ağ yapısını okumak, parse etmek (uygun şekilde parçalara bölmek) ve NetList için oluşturulmuş veri yapısında saklamaktır. Şekil 3.17'de NetList okuyucunun kullandığı parse edilen ağın bilgilerinin saklandığı veri yapısı görülmektedir. Bu veri yapısı işretçiler (pointerler) kullanılarak çok esnek bir şekilde tanımlanmıştır. NetList okuyucu bir yandan verilen NetList'i okurken bir yandan da ilgili NetList'in ilgili bölümünü depolamak için gerekli hafıza alanını oluşturarak bu yapıya ekler. Örneğin NetList'ten katman adedi okunur okunmaz belirtilen katman kadar hafıza alanı C++ new operatörü ile oluşturularak veri yapısına eklenir. Aynı şekilde her bir katmandaki nöron sayısı belli olur olmaz ilgili katmanda istenen adette nöronu saklayacak hafıza alanı yine new operatörü ile oluşturularak veri yapısına eklenir.

Şekil 3.18'de, NetList okuma algoritması görülmektedir. Öncelikle NetList dosyasından katman adedi okunarak okunan adet kadar katmanı saklayabilecek veri alanı oluşturulur. Ardından bu kısıma yardımcı olarak yazılan Katman okuma fonksiyonu çağırılarak katmanlarla ilgili bütün bilgilerin okunması ve ilgili veri alanında depolanması sağlanır. Katmaların okunmasının ardından oluşturulacak veri yolu ile ilgili parametreler yine aynı NetList dosyasından okunarak NetList veri yapısında ilgili alanlara aktarılır. Okunan parametreler sırasıyla oluşturulacak veri yolunun işleyeceği veri türünü belirten DataType, veri genişliğini belirten DataWidth, adres genişliğini belirten AddressWidth ve oluşturulacak VHDL kodunu yazılacağı VHDL dosyasının ismini belirten VHDLName parametreleridir.

Netlist okuyucuya yardımcı olarak Nöron karşılaştırma ve Katman okuyucu fonksiyonları yazılmıştır. Nöron karşılaştırma fonksiyonu NetList'te metin olarak okunan nöron ismini dâhili olarak tanımlanmış nöron türlerine dönüştürür. Katman okuma fonksiyonu ise NetList dosyasında yer alan katmanları NetList veri alanına okur.



Şekil 3.17: NetList'in tutulduğu veri yapısı

Şekil 3.19'da, katman okuma fonksiyonunun algoritması verilmiştir. Fonksiyon bir döngü içerisinde döngünün her tekrarlanışında bir katmana ait bütün bilgileri okur ve NetList veri yapısına aktarır. Döngünün içinde öncelikle katmanın türü okunur. Üç değişik katman türü bulunmaktadır. Bu katman türleri INPUT, NEURON ve OUTPUT'dur. Her bir katman türünün yapısı ve içeriği farklılık gösterdiğinden katman okuma fonksiyonu katman türü belli olduktan sonra ilgili katmana göre sadece ilgili bilgileri okuyarak Netlist'e aktarır.

```
NetListType * ReadNetList(char * FileName)
Begin
    NetListType *NL;
    NL ← New NetListType;
    Fpt ← OpenFile (FileName);
    NL.LayerCount ←Fpt;
    NL.Layers ← New LayerType (NL.LayerCount)
    For  (each Layers (i) in the NetList) Do
        NL.Layers(i) = ReadLayer(Fpt)
    End For
    ParamCount ← Fpt
    For (each Parameter (i) in the NetList) Do
        Word←Fpt;
        If (Word = "DataType") Then
            Word ←Fpt
            If (Word= "integer") Then
                NL.DataType = 0  #   0 ==> integer
            Else
                NL.DataType = 1        #   1 ==> float
            End If
        End If
        If (Word = "DataWidth") Then
            NL.DataWidth←Fpt
        End If
        If (Word = "AddressWidth") Then
            NL.AddressWidth←Fpt
        End If
        If (Word = "VHDLName") Then
            NL.VHDLName←Fpt
        End If
    End For
    Fpt←CloseFile()
    Return NL
End
```

Şekil 3.18: NetList'i okuma fonksiyonu algoritması

```
LayerType ReadLayer(Ifstream & Fpt)
Begin
    LayerType LT
    LT.LayerNo←Fpt
    LT.ElementCount ← Fpt
    LT.Elements ←New ElementType(LT.ElementCount)
    If (Word="INPUT")Then
        LT.Kind = Input
        for (each Element in the LT) Do
            LT.Elements(i).Name← Fpt
        End For
    Else If (Word="NEURON")
        LT.Kind = Neuron
        for (each Element in the LT) Do
            LT.Elements(i).Name← Fpt
            Word← Fpt
            LT.Elements(i).Type = StrToNeuron(Word)
            Bias← Fpt
            LT.Elements(i).Bias = Bias
            LT.Elements(i).BiasWeight← Fpt
            LT.Elements(i).InputCount← Fpt
            LT.Elements(i).Inputs ← New InputType[LT.Elements(i).InputCount]
            for (each Input in the LT.Elements(i)) Do
                LT.Elements(i).Inputs(k).LayerNo → LT.Elements(i).Inputs(k).InputName →
                    LT.Elements(i).Inputs(k).Weight← Fpt
            End For
        End For
    Else
        LT.Kind = Output;
        For (each Element in the LT) Do
            LT.Elements(i).Name →  LT.Elements(i).OutputLayerNo →
                LT.Elements(i).OutputNeuronName← Fpt
        End For
    End If
    Return LT;
End
```

Şekil 3.19: Katmanları okuma fonksiyonu algoritması

3.2.3.2. Kütüphane Okuma Bölümü

Şekil 3.20'de, Kütüphane okuma fonksiyonunun algoritması görülmektedir. Bu fonksiyon verilen kütüphane dosyasında hâlihazırda var olan ve veri yolu oluşturmada kullanılabilecek nöron hücreleri hakkındaki bilgileri okuyarak Kütüphane türünde tanımlanmış veri yapısına aktarır.

```
LibraryType *ReadLibrary(char * FileName, int &EC)
Begin
    LibraryType *LB
    LB ← New LibraryType (100)
    Fpt← OpenFile(FileName)
    While (!Fpt.eof()) Do
        Word← Fpt
        LB[EC].Type = StrToNeuron(Word)
        Tmp← Fpt
        LB(EC).Bias  = Tmp
        LB(EC).InputCount← Fpt
        If (EC=100)Then
            break
        End If
    End While
    Fpt← CloseFile()
    Return LB
End
```

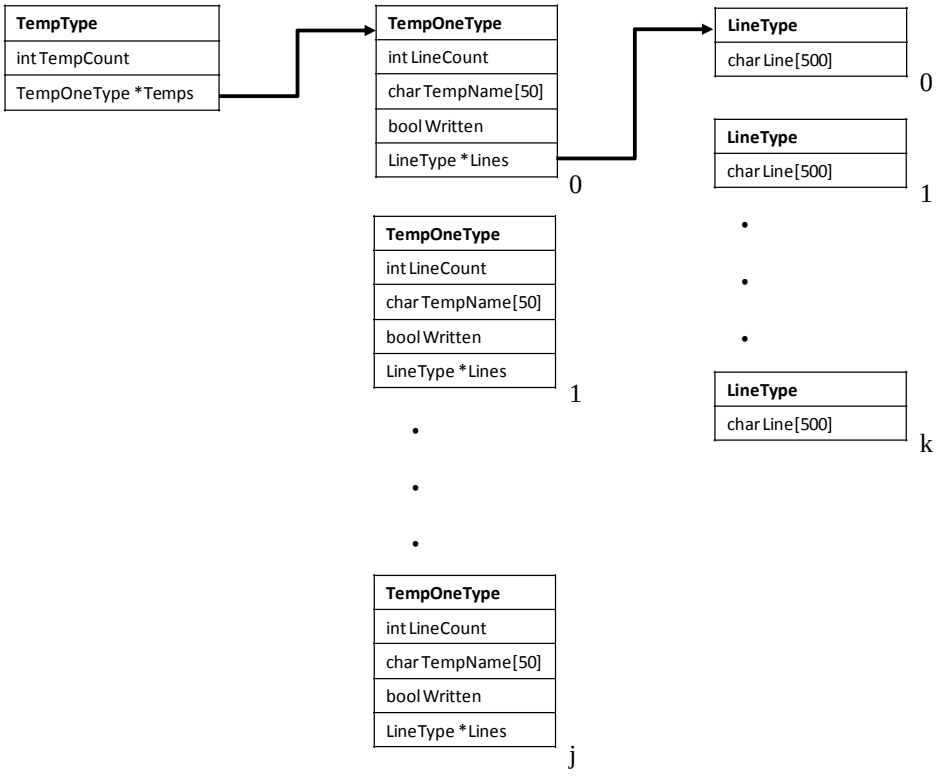
Şekil 3.20: Kütüphane okuma fonksiyonu algoritması

3.2.3.3. Şablon Dosyasını Okuma Bölümü

Şekil 3.21'de Şablon okuma fonksiyonunun algoritması görülmektedir. Şablon dosyası yalın metin formatında tanımlanmış VHDL kalıplarını ve nöronların VHDL kod tanımlamalarını içerir. Fonksiyon, dosya adı verilen Şablon dosyasını açarak dosya içinde bulunan ve VHDL kod yazımında kullanılacak şablon bilgileri ile nöronların VHDL kod tanımlamalarını okur ve Template türünün tanımlanmış veri alanına düzenli bir şekilde aktarır. Şablonların depolandığı Template veri türü Şekil 3.22'de görülmektedir. Yine burada NetList türünde olduğu gibi içine değişik sayıda satırlar içeren yüzlerce şablonun saklanabileceği ama hafızadan sadece ihtiyaç duyulan miktarda alan kullanan esnek bir veri yapısı oluşturulmuştur.

```
TempType *ReadTemplate(char * FileName)
Begin
    TempType *T
    T ← New TempType
    Fpt← OpenFile(FileName)
    T.TempCount← Fpt
    T.Temps ← New TempOneType(T.TempCount)
    For (each element of Temp)
        T.Temps(i).TempName←Fpt
        T.Temps(i).LineCount←Fpt
        T.Temps(i).Written←false
        T.Temps(i).Lines←New LineType (T.Temps(i).LineCount)
        ch ← Fpt
        For (each element of T.Temps(i).Line)
            ch ← Fpt
            While (ch != endl)Do
                T.Temps(i).Lines(k).Line(j)=ch
                ch ← Fpt
            End While
            T.Temps(i).Lines(k).Line(j) = 0
        End For
    End For
    Fpt ←CloseFile()
    Return T
End
```

Şekil 3.21: Şablon okuma fonksiyonu algoritması



Şekil 3.22: Şablonun tutulduğu veri yapısı

3.2.3.4. Nöron Kontrolü

ANNGEN bütün girdi dosyalarını okuduktan sonra istenen YSA veri yolunu oluşturmadan önce verilen NetList'deki bütün nöronların hâlihazırda kütüphanede olup olmadığını kontrol eder. Bu amaçla bir modül kontrol fonksiyonu (Module Check) yazılmıştır. Şekil 3.23'te modül kontrol fonksiyonunun algoritması görülmektedir. Modül kontrol fonksiyonu parametre olarak NetList ve Kütüphane verilerini alır ve NetList deki bütün katmanları dolaşarak her bir katmandaki her bir nöronun kütüphanede olup olmadığını kontrol eder. Eğer bütün nöronların tanımlaması kütüphanede varsa Doğru (True), diğer durumlarda Yanlış (False) değerini gönderir. Herhangi bir nöronun kütüphanede olup olmadığını kontrol için bir nöronu kontrol fonksiyonu (CheckOneNeuron) geliştirilmiştir. Fonksiyon parametre olarak verilen her bir nöron için kütüphaneyi baştan sona kadar tarar ve o nöronun kütüphanede olup olmadığına karar verir.

ANNGEN Modül kontrol fonksiyonundan Doğru sonucu gelmiş ise NetList de belirtilen YSA için gerekli VHDL kodunu üretmek üzere VHDL yazıcı fonksiyonunu çalıştırır. Eğer False gelmiş ise bir uyarı mesajı vererek sonlanır.

3.2.3.5. VHDL Kod Yazdırma Bölümü (VHDL Yazıcı)

ANNGEN Modül kontrol fonksiyonu ile verilen NetList'deki bütün nöronların kütüphanede olduğundan emin olduktan sonra VHDL kod yazımına başlar. Bu amaçla geliştirilen VHDL Yazıcı (WriteVHDL) fonksiyonunun algoritması Şekil 3.24'te görülmektedir. Fonksiyon öncelikle VHDL kod yapısı gereği üretilen VHDL kodu içinde kullanılacak RegGen, FPMult8 ve FpAdd8 modüllerinin VHDL kodlarını şablon veri yapısından bularak yeni VHDL kodunun başına yazar. Ardından tüm NeList'i dolaşarak istenen nöronların VHDL tanımlamalarını şablon veri yapısı içinden bularak oluşturulan yeni VHDL kodunun devamına yazar. Burada her bir nöronun VHDL kodu yalnızca bir defa yazılır. Daha sonra ANNGEN tarafından tasarlanan veri yolunun yazımına geçilir. Burada ilk olarak şablonlar kullanılarak bir açıklama kısmı yazdırılır. Bu açıklama kısmında klasik bir VHDL kodunda görülebilecek açıklamaların başlıkları yazılır ve geri kalanı kullanıcı tarafından doldurulmak üzere boş bırakılır. Sadece tarih bilgisi sistemden alınarak buraya yazılır.

```
Bool CheckModules (NetListType *N, LibraryType *L, int EC)
Begin
    int i,j;
    Bool Result = True;
    For (each Layer in N) Do
        If (N->Layers[i].Kind == Neuron)
            For (j=0; j<N->Layers[i].ElementCount; j++) Do
                Result = Result & CheckOneNeuron(L, &N->Layers[i].Elements[j],EC);
            End For
        End If
    End For
    Return Result;
End
```

Şekil 3.23: Modül kontrol fonksiyon algoritması

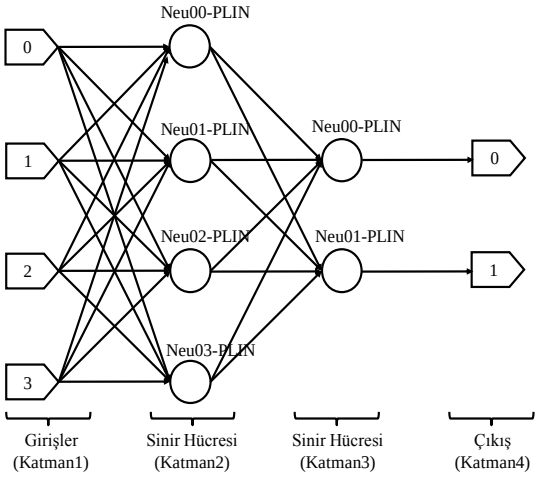
```
Void WriteVHDL(char *FileName, NetListType *NT, LibraryType *Lib, TempType *T)
Begin
   Fptn ←OpenFile(FileName)
   For (k=0 To TempCount; k++)
      If ((strcmp(T->Temps[k].TempName, "REGGEN"))==0  ||
          (strcmp(T->Temps[k].TempName, "FPMULT8"))==0 ||
          (strcmp(T->Temps[k].TempName, "FPADD8"))==0 )
              WriteOneTemp(ofn,T,k);
      End If
   End For
   For (each element of NT.Layer) Do
      If (NT.Layers(i).Kind = Neuron) Then
          For (each element of NT.Layers(i).Element) Do
              For (each element of T.Temp) Do
                  If ((strcmp(T.Temps(k).TempName, Str)) = 0 && !(T.Temps(k).Written)) Then
                      WriteOneTemp(Fptn,T,k)
                      T.Temps(k).Written = true
                  End If
              End For
          End For
      End If
   End For
   WriteOneTemp(Fptn,T,0)
   time (&rawtime)
   timeinfo = localtime (&rawtime )
   "-- " →  asctime (timeinfo)←Fptn
   WriteOneTemp(Fptn,T,1)
   WriteOneTemp(Fptn,T,2)
   WriteOneTemp(Fptn,T,3)
   WriteENTITY(Fptn,NT)
   WriteARCHITECTURE(Fptn, NT, Lib, T)
   Fptn←CloseFile()
End
```

Şekil 3.24:VHDL yazdırma algoritması

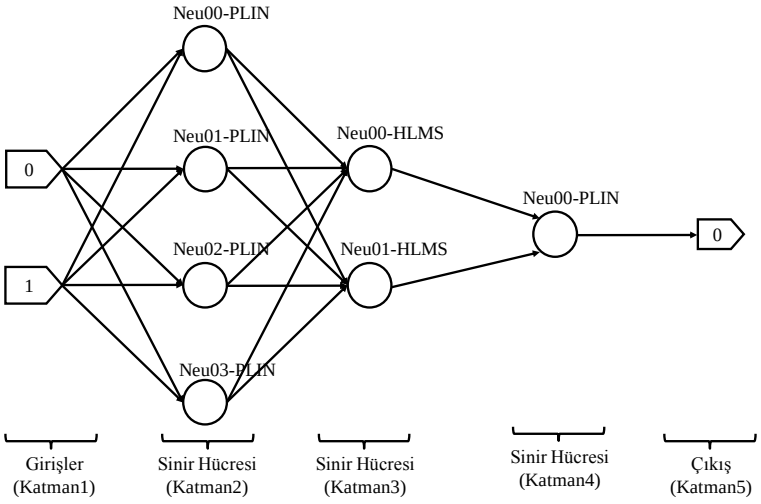
4. BULGULAR

4.1. ANNGEN'İN TEST EDİLMESİ

Bu çalışmada geliştirilen ANNGEN' test etmek amacıyla Şekil 4.1 ve Şekil 4.2'de görülen YSA'lar için Şekil 4.3.a ve b'de görülen NetList tanımlamaları yapılmıştır. Birinci YSA dört giriş, iki çıkış ve iki gizli katmandan oluşmaktadır. Gizli katmanlarda Doğrusal (Pure Linear) aktivasyon fonksiyonuna sahip nöron hücreleri kullanılmıştır. İkinci YSA ise iki giriş, tek çıkış ve üç gizli katmandan oluşmaktadır. Birinci katmandaki nöronlar Doğrusal, ikinci katmandaki nöronlar Simetrik sabit limit (Symmetric Hard Limit) ve üçüncü katmandaki nöron ise Doğrusal aktivasyon fonksiyonuna sahiptir.



Şekil 4.1: Örnek Netlist1'in sinir ağı şeması



Şekil 4.2: Örnek Netlist2'nin sinir ağı şeması

```
NETLIST 4
[
     LAYER 0 INPUT 4
     [
          INP00 INP01 INP02
INP03
     ]
     LAYER 1 NEURON 4
     [
          NEU00 PLIN 0 0.0 4
               0 INP00 1.1
               0 INP01 1.1
               0 INP02 1.1
               0 INP03 1.1
          NEU01 PLIN 0 0.0 4
               0 INP00 1.1
               0 INP01 1.1
               0 INP02 1.1
               0 INP03 1.1
          NEU02 PLIN 0 0.0 4
               0 INP00 1.1
               0 INP01 1.1
               0 INP02 1.1
               0 INP03 1.1
          NEU03 PLIN 0 0.0 4
               0 INP00 1.1
               0 INP01 1.1
               0 INP02 1.1
               0 INP03 1.1
     ]
     LAYER 2 NEURON 2
     [
          NEU00 PLIN 0 0.0 4
               1 NEU00 1.1
               1 NEU01 1.1
               1 NEU02 1.1
               1 NEU03 1.1
          NEU01 PLIN 0 0.0 4
               1 NEU00 1.1
               1 NEU01 1.1
               1 NEU02 1.1
               1 NEU03 1.1
     ]
     LAYER 3 OUTPUT 2
     [
          OUT00 2 NEU00
          OUT01 2 NEU01
     ]
]
PARAMETERS 4
[
     DataType float
     DataWidth 32
     AddressWidth 32
     VHDLName Deneme
]
```

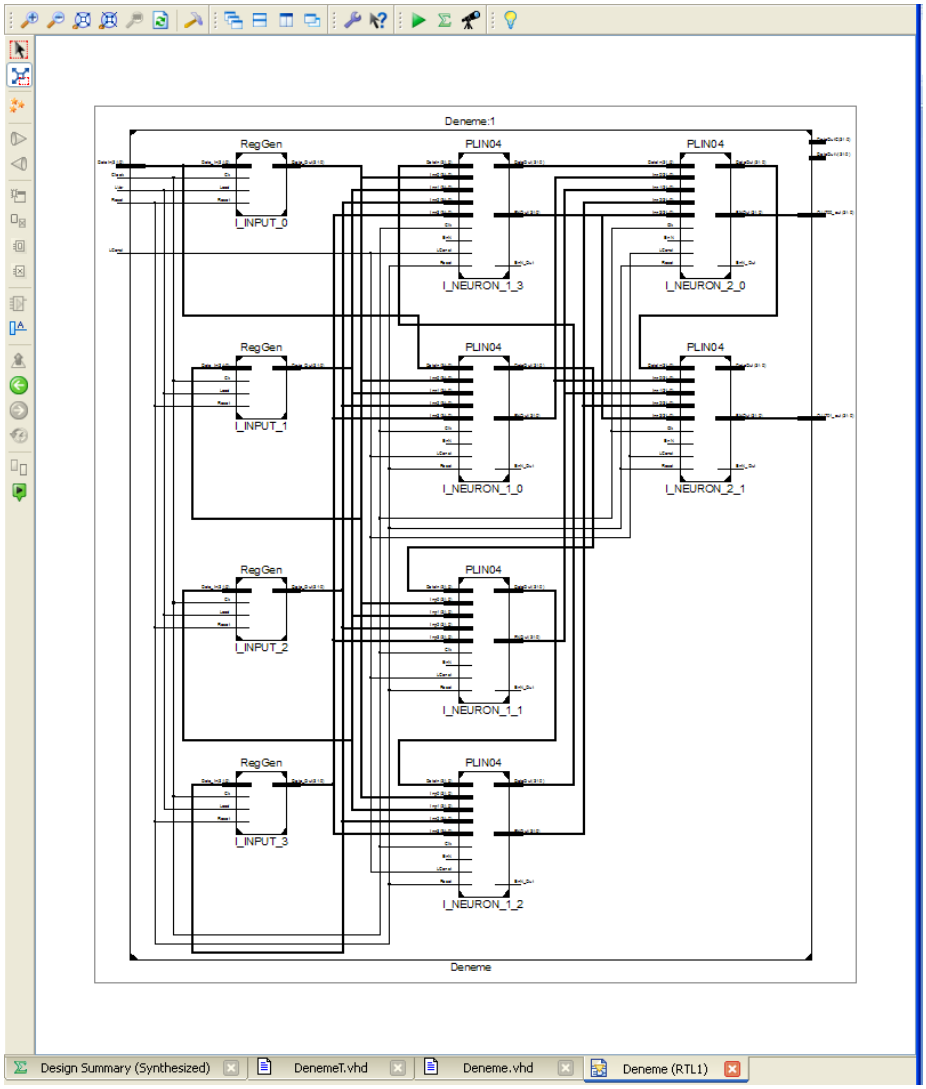
(a)

```
NETLIST 5
[
     LAYER 0 INPUT 2
     [
          INP00 INP01
     ]
     LAYER 1 NEURON 4
     [
          NEU00 PLIN 0 0.0 2
               0 INP00 1.1
               0 INP01 1.1
          NEU01 PLIN 0 0.0 2
               0 INP00 1.1
               0 INP01 1.1
          NEU02 PLIN 0 0.0 2
               0 INP00 1.1
               0 INP01 1.1
          NEU03 PLIN 0 0.0 2
               0 INP00 1.1
               0 INP01 1.1
     ]
     LAYER 2 NEURON 2
     [
          NEU00 HLMS 0 0.0 4
               1 NEU00 1.1
               1 NEU01 1.1
               1 NEU02 1.1
               1 NEU03 1.1
          NEU01 HLMS 0 0.0 4
               1 NEU00 1.1
               1 NEU01 1.1
               1 NEU02 1.1
               1 NEU03 1.1
     ]
     LAYER 3 NEURON 1
     [
          NEU00 PLIN 0 0.0 2
               2 NEU00 1.1
               2 NEU01 1.1
     ]
     LAYER 4 OUTPUT 1
     [
          OUT00 3 NEU00
     ]
]
PARAMETERS 4
[
     DataType float
     DataWidth 32
     AddressWidth 32
     VHDLName Deneme
]
```

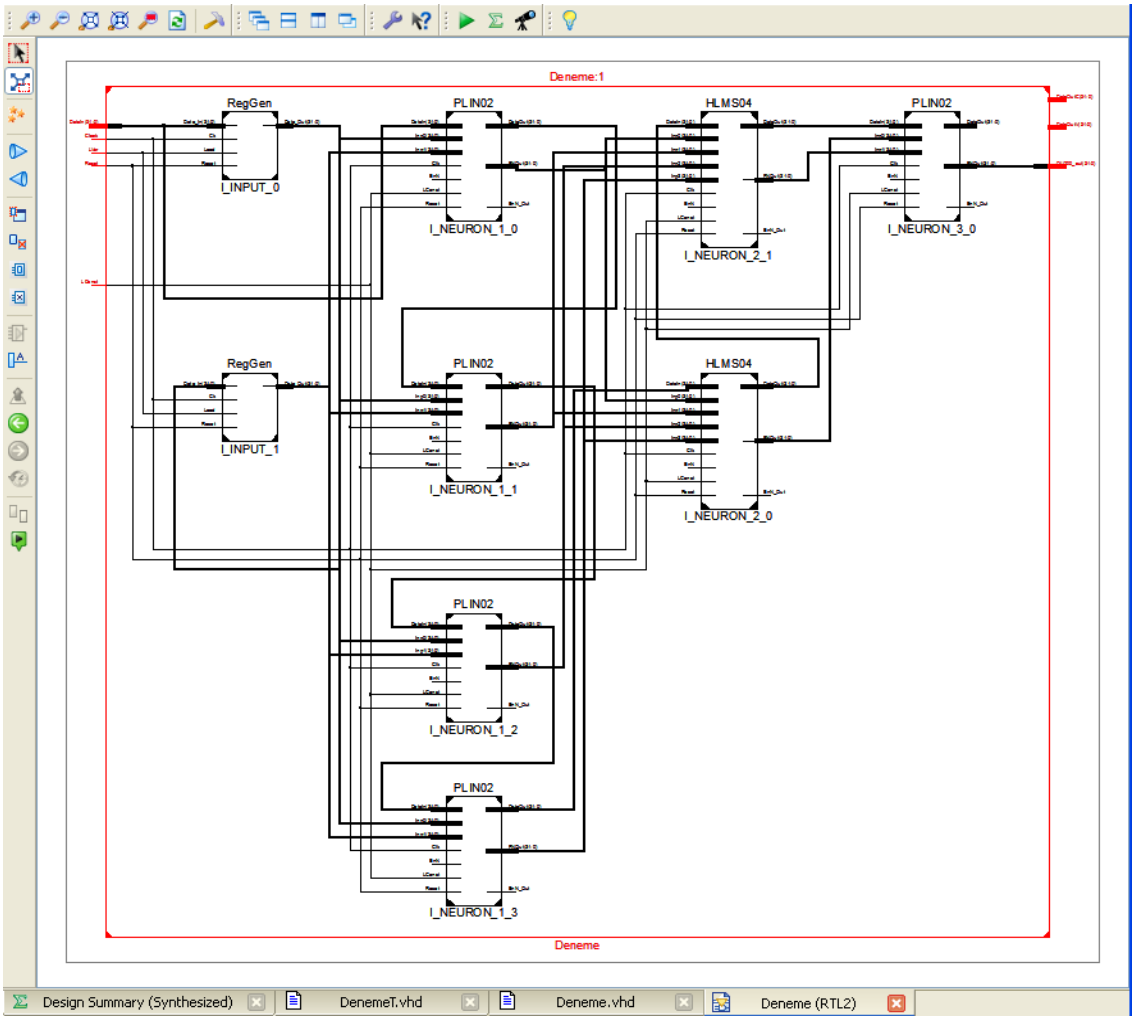
(b)

Şekil 4.3: Test durumları için yapılan NetList tanımlamaları: (a) Dört girişli iki gizli birinci NetList (b) İki girişli üç gizli katmanlı ikinci NetList

ANNGEN, Şekil 4.3.a ve b'de görülen NetList'ler ile birlikte çalıştırılarak istenen YSA'lar için VHDL kodu üretmesi sağlanmıştır. ANNGEN tarafından üretilen VHDL kodları Ek-A ve Ek-B'de verilmiştir. Kodların doğruluğunu test etmek amacıyla Xilinx'in ISE tasarım aracında her bir kod için ayrı ayrı projeler tanımlanmıştır. Tanımlanan bu projelere kodlar eklenerek ISE tarafından önce yazım hatası kontrolü yapılmış ardından devre sentezlenerek her bir YSA için RTL (Register Transfer Level) görüntüsü oluşturulması sağlanmıştır. Oluşturulan RTL'ler tasarlanmak istenen YSA'lar ile karşılaştırılarak ANNGEN'in doğruluğu teyit edilmiştir. Bu işlemlerin tamamı dakikalar içinde tamamlanmıştır. Böylece ANNGEN'in FPGA de YSA tasarımını nasıl hızlandırdığı da görülmüştür. Şekil 4.4 ve Şekil 4.5'de, her bir örnek test YSA'sı için oluşturulmuş RTL görüntüleri sunulmuştur.



Şekil 4.4: Örnek NetList1'in RTL yapısı



Şekil 4.5: Örnek NetList2'nin RTL yapısı

5. TARTIŞMA VE SONUÇ

İnsanoğlu doğadan ilham alarak değişik model ve modellemeler geliştirmiştir. Yapay sinir ağları, insan beyninde doğal olarak var olan sinir hücrelerinin matematiksel olarak taklit edilmesiyle üretilen modele denir. Yapay zekânın bir uygulaması olan yapay sinir ağları son yıllarda geleceği öngörmede, örüntü tanımada, verilerin yorumlanmasında, optimizasyon işlemlerinde, doğrusal olmayan fonksiyon yaklaşımlarında ve birçok değişkene bağımlı zaman serilerinin tahmin edilmesinde önemli bir araç olarak kullanılmaktadır.

YSA'lar, hem yazılım hem de donanım olarak gerçeklenebilmektedir. Yazılım olarak modellendiğinde özellikle ağ yapısı karmaşıklaştıkça CPU gücü gereksinimi artmaktadır. Özellikle gerçek zamanlı uygulamalarda, anında YSA çıkışının hesaplanması gerektiği durumlarda yazılım olarak modellenen YSA'lar istenen performansı gösterememektedir. Yazılım olarak gerçeklenen YSA'lara birçok alternatif yöntem geliştirilmiştir. Bunlardan bir tanesi de yeniden yapılandırılabilir hesaplama (RC) sistemi kullanımıdır.

RC sistemleri genel amaçlı bir bilgisayara eklenmiş programlanabilir FPGA yonga ya da yonga setlerinden oluşur. Bu sistemlerde programların yoğun kısımları özel olarak tasarlanmış modüller sayesinde FPGA yongaları üzerinde çalıştırılır. Çünkü FPGA'lar paralel işlem yapma ve yeniden yapılandırılabilme özelliklerine sahiptirler. YSA'lar paralel işlem yaptıklarından çok hızlı çalışan paralel işlemcilere ihtiyaç duyarlar. Değişik problemler için esnek, değişebilen katmanlara ve farklı transfer fonksiyonlu modellere sahip olmalıdırlar. YSA'lar için gerekli olan bu özellikler FPGA de kolay olarak yapılabilmektedir. Ancak YSA'nın FPGA'ya uygulanması zaman alan ve uzman gerektiren bir iştir.

Bu tez çalışmasında Yapay Sinir Ağları'nın otomatik olarak FPGA'lara uygulanmasına yardımcı olacak bir tasarım aracı (ANNGEN) geliştirilmiştir. ANNGEN girdi olarak üç dosyaya ihtiyaç duyar. Bunlar kullanıcının oluşturulmak istenen YSA'yı metin tabanlı olarak tanımladığı NetList dosyası, ANNGEN'in kullanabileceği sinir hücrelerinin listelendiği Kütüphane dosyası ve hem sinir hücrelerinin VHDL kodlarının bulunduğu hem de VHDL kodu yazımında kullanılan şablonların bulunduğu şablon dosyasıdır.

ANNGEN bu üç dosyayı alarak FPGA'da oluşturulmak istenen YSA için gerekli VHDL kodunu saniyeler içinde üretir. Normal şartlarda bu kodun üretimi zaman alan ve uzman gerektiren bir işlemdir. ANNGEN ise bunu çok kısa bir sürede ve uzman gereksinimi olmadan gerçekleştirir. Burada tek yapılması gereken ANNGEN'e oluşturulmak istenen YSA'nın tanımlandığı NetList dosyasını ve daha önceden bir defa tanımlanmış kütüphane ve şablon dosyaları ile vermektir.

Hâlihazırda ANNGEN'in kütüphanesinde üç farklı transfer fonksiyonu için iki girişli ve dört girişli olmak üzere altı adet sinir hücresi tanımlanmıştır. Burada tanımlanan sinir hücrelerinin VHDL kodları şablon dosyasına da eklenmiştir. Yeni sinir hücreleri çok kolay bir şekilde ANNGEN'e tanıtılabilmektedir. Yeni hücre tanımlandığında yapılması gereken iki şey bu hücrenin özelliklerinin bir satırlık metin olarak kütüphaneye eklenmesi ve ardında hücreye ait VHDL kodunun şablon dosyasına uygun formatta yerleştirilmesidir. Bu aşamadan sonra ANNGEN üzerinde herhangi bir değişiklik yapılmadan yeni hücreler YSA oluşturulmasında kullanılabilmektedir.

ANNGEN'i ve bu tezde geliştirilen bu yeni YSA tasarım yöntemini test etmek için iki adet test durumu oluşturulmuştur. Her bir test durumu için ayrı ayrı NetList tanımlamaları yapılmış ve mevcut kütüphane ve şablon dosyaları ile ANNGEN'e verilmiştir. ANNGEN her iki test durumunda da verilen NetList'e uygun VHDL kodunu saniyeler içinde üretmiştir. Üretilen VHDL kodlarının doğrulunu test etmek için kodlar Xilinx'in ISE tasarım aracı ile önce yazım kontrolünden geçirilmiş ardından da sentezlenerek başarılı bir şekilde RTL şeması oluşturulmuştur. NetList verildiğinde ANNGEN'in saniyeler içinde otomatik olarak VHDL kodunu üretmesi sayesinde, YSA için gerekli VHDL kodu tanımlama sürecini çok kısalttığı ve bu işlem için uzman personel gereksinimini ortadan kaldırdığı gözlenmiştir.

ANNGEN FPGA'da YSA oluşturmak için gerekli veri yolu tasarımını yapmaktadır. Tasarlanan veri yolunu kontrol edecek bir kontrol ünitesine ihtiyaç vardır. Gelecekte ANNGEN kontrol ünitesini de oluşturacak şekilde geliştirilebilir. Yapılabilecek bir diğer çalışma ise ANNGEN'in kütüphanesinin genişletilmesidir. Hâlihazırda kütüphanede altı adet nöronun tanımlaması vardır. Daha değişik aktivasyon fonksiyonlarına sahip daha çok giriş sayılı yeni sinir hücreleri geliştirilerek ANNGEN'in kütüphanesine dâhil edilebilir.

KAYNAKLAR

ANON, 2011, *İnsan Beyni* [Online], Ankara-Türkiye, http://www.biltek.tubitak.gov.tr/bdergi/poster/icerik/bellek.pdf, [Ziyaret Tarihi: 8 Nisan 2011].

ANON, 2011a, *The Hows, Whats and Whos of Neuroscience* [Online], USA, http://faculty.washington.edu/chudler/what.html, [Ziyaret Tarihi: 10 Nisan 2011].

ANON, 2011b, *Transfer Functions* [Online], USA, http://www.mathworks.com/help/toolbox/nnet/ref/f7-23438.html, [Ziyaret Tarihi: 10 Nisan 2011].

ANON, 2011c, *FPGA Device Families* [Online], USA, http://www.xilinx.com/support/documentation/data_sheets, [Ziyaret Tarihi: 8 Nisan 2011].

ANON, 2011d, *Altera Devices* [Online], USA, http://www.altera.com/products/devices/dev-index.jsp, [Ziyaret Tarihi: 8 Nisan 2011].

ANON, 2011e, *FPGA* [Online], Türkiye, http://www.cpu-turkey.com/upload/ group/4/fpga_ceviri.pdf , [Ziyaret Tarihi: 8 Nisan 2011].

ANON, 2011f, *Xilinx Virtex™ 5 FX70T, FX100T 4-Lane PCI Express® Gen. 2 / USB 3.0 Development Board*, [Online], USA, http://www.hitechglobal.com/boards/v5ddr3_pcie.htm, [Ziyaret Tarihi: 8 Nisan 2011].

AYDIN, A., 2008, *Field Programmable Gate Array Kullanım Alanları*, *Field Programmable Gate Array (FPGA) VHDL*, [Online], Türkiye, http://fpga.blogcu.com/field-programmable-gate-array-kullanim-alanlari/3864589, [Ziyaret Tarihi:10 Nisan 2011].

BASTOS, J. L., FİGUEROA, H. P., MONTİ, A., 2006, FPGA Implementation of Neural Network-Based Controllers for Power Electronics Applications, *Applied Power Electronics Conference and Exposition, 2006. APEC '06. Twenty-First Annual IEEE* , 19-23 March 2006, USA, 6.

BENREKİA,F., ATTARİ, M., BERMAK, A,. BELHOUT, K., 2009, FPGA Implementation of a Neural Network Classifier for Gas Sensor Array Applications, *6th International Multi-Conference on Systems, Signals and Devices*, 23-26 March 2009, Djerba,1-6.

BÜRHAN, Y., GÜLENÇ , H., 2011, *FPGA*, [Online], Türkiye, web.firat.edu.tr/bilmuh/gaydin/dersler/0809/bmu401/ppt/fpga.doc, [Ziyaret Tarihi:10 Nisan 2011].

COOPER, E., 2011, *Understanding Of How Nerve Cells Acquire Their Unique Structures And Functions* [Online], Kanada, http://www.medicine.mcgill.ca/physio/cooperlab/cooper.htm, [Ziyaret Tarihi: 10 Nisan 2011].

ÇAVUŞLU, M.A., KARAKUZU,C., ŞAHİN, S., KARAKAYA, F., 2008, Yapay Sinir Ağı Eğitiminin IEEE 754 Kayan Noktalı Sayı Formatı İle FPGA Tabanlı Gerçeklenmesi, *Gomsis 2008*,3-4-5 Kasım 2008, İstanbul Teknik Üniversitesi Süleyman Demirel Kültür Merkezi, Maslak Yerleşkesi, İstanbul.

ÇAVUŞLU,M.A., ALTUN,H., KARAKAYA, F., 2008a, Plaka Yeri Tespiti İçin Kenar Bulma, Bit Tabanlı Öznitelik Çıkartma ve YSA Sınıflandırıcısının FPGA Üzerine Uyarlanması, *Gomsis 2008*,3-4-5 Kasım 2008, İstanbul Teknik Üniversitesi Süleyman Demirel Kültür Merkezi, Maslak Yerleşkesi, İstanbul.

ÇORUMLUOĞLU, Ö., ÖZBAY, Y., KALAYCI, İ., ŞANLIOĞLU, İ., 2005, *GPS (Global Positioning System - Küresel Konumlandırma Sistemi) Yüksekliklerinden Ortometrik Yüksekliklerin Elde Edilmesinde Yapay Sinir Ağı Tekniğinin Kullanımı*, Har. ve Kad. Müh.Odası, Mühendislik Ölçmeleri STB Komisyonu 2. Mühendislik Ölçmeleri Sempozyumu, Kasım 2005, İTÜ.

DAZSI B. A., ENBODY, R., 2001, *Artificial Neural Networks For Branch Prediction*, Master Science Thesis, Department of Electrical and Computer Engineering, Michigan State University.

DEMUTH, H., HAGAN, M. T., BEALE M., 2011[Online], *Neural Network Toolbox™ 7 For User's Guide with MATLAB*, The MathWorks, USA, Revised for Version 7.0.1 (Release 2011a), April 2011**.**

ELMAS, Ç., 2003, *Yapay Sinir Ağları (Kuram, Mimari, Eğitim, Uygulama)*, Seçkin Yayıncılık, Ankara, 9.789.753.476.126.

ELMAS, Ç., 2007*, Yapay Zekâ Uygulamaları*, Seçkin Yayıncılık, Ankara, 9.789.750.206.146.

FIRAT, M., GÜNGÖR M., 2004, Askı Madde Konsantrasyonu ve Miktarının Yapay Sinir Ağları ile Belirlenmesi, *İMO Teknik Dergi*, Cilt 15, Sayı 3, 3267-3282.

FREEMAN, J., SKAPURA D., 1991, *Neural Networks Algorithms, Applications, and Programming Techniques*, Addison-Wesley Publishing Company, U.S.A, 0.201.513.765.

GLOSTER, C., and ŞAHİN, İ., 2001, *Floating-Point Modules Targeted for Use with RC Compilation Tools*, Earth Science Technology Conference (ESTC), College Park, MD.

HAYKIN, S., 1994, *Neural Networks : A Comprehensive Foundation*, Macmilan College Publishing, U.S.A., 0 02 352761 7.

KANDEL, E.R., 1991, *Principles of Neural Science*, Elsevier Science Publishing Co. U.S.A. *9.780.444.006.516.*

KOYUNCU, İ., 2008, *A Matrix Multiplication Engine for Graphic Systems Designed to run on FPGA Devices*, Düzce Üniversitesi, Doktora Tezi, Düzce-Türkiye.

MANDIC, D.P., CHAMBERS J.A., 2001, *Recurrent Neural Networks For Prediction - Learning Algorithms, Architectures And Stability*, John Wiley & Sons Ltd,USA, 0.471.495.174

NABİYEV, V., 2005, *Yapay Zekâ – Problemler, Yöntemler, Algoritmalar*, Seçkin Yayınevi, İkinci Basım, Ankara, 9789753479851.

ÖZDAMAR, S. G., 2007, *Reconfigurable Architectures (Tekrar Düzenlenebilir Mimariler)*, [Online], Türkiye, http://web.itu.edu.tr/~orencik/BilgMimYenYakl2007/Selcuk

G_Ozdamar/Tekrar_Duzenlenebilir_Mimariler_Rapor_504061529.pdf, [Ziyaret Tarihi:10 Nisan 2011].

ÖZDEMİR, A.T., 2009, FPGA Tabanlı Aritmi Sınıflandırıcı-FPGA Based Arrhythmia Classifier, *Biyomut 2009*, 14, Biyomedikal Mühendisliği Ulusal Toplantısı, 20-24 Mayıs, 2009, Dokuz Eylül Üniversitesi, Balçova, İzmir, 1-4.

ÖZDEN,Ö., 2010, *Çok Katmanlı Algılayıcı Yapısının FPGA İle Gerçeklenmesi*, Yüksek Lisans Tezi, Mayıs 2010, İstanbul

ÖZTEMEL, E., 2003, *Yapay Sinir Ağları*, Papatya Yayıncılık, İstanbul, 9756797398

PETRIU, E.M., 2011, *Neural Networks: Basics*, [Online], School Of Information Technology And Engineering University Of Ottowa, Canada, http://www.site.uottawa.ca/~petriu/nn_basics-tutorial-2004.pdf, [Ziyaret Tarihi: 10 Nisan 2011].

RINCON, F., and TERES, L., 1998, *Reconfigurable Hardware Systems*, International Semiconductor Conference, vol. 1, pp. 45-54.

SARLE W. S., 1994, *Neural Networks and Statistical Models*, Proceedings of the Nineteenth Annual SAS Users Group International Conference, Nisan 1994, N.C. USA.

SELİM, E., ULUCAN, A., 2008, *FPGA Donanımı ve FPGA Üzerinde İşlemci Tasarımı*, E.Ü. Bilgisayar Mühendisliği Bölümü Emin Hitay Konferans Salonu, 14 Kasım 2008, İvme – Momentum Fpga Topluluğu, Türkiye

ŞAHİN, İ., 2002, *A Compilation Tool for Automated Mapping of Algorithms onto FPGA Based Custom Computing Machines*, Doktora Tezi, NC State University, Raleigh-USA.

ŞAHİN, İ., 2010, A 32-Bit Floating-Point Module Design for 3D Graphic Transformations, *Scientific Research and Essays* Vol. 5 (20), pp. 3070-3081, 18 October, 2010, Available Online at http://www.academicjournals.org/SRE, ISSN 1992-2248 ©2010 Academic Journals, Full Length Research Paper.

ŞAHİN, İ., GLOSTER, C.S., and DOSS, C., 2000, *Feasibility of Floating-Point Arithmetic in Reconfigurable Computing Systems*, Military and Aerospace Applications of Programmable Devices and Technology Conference, Washington, DC.

ŞAHİN, İ., GLOSTER, C.S., 2005, *Evaluation of Ic Physical Design Optimization Algorithms for Acceleration Using FPGA-Based Custom Computing Machines*, İleri Teknolojiler Sempozyumu, Konya.

ŞAHİN, İ., ve GLOSTER, C.S., 2005, *FPGA Tabanlı CCM'ler İçin Genel Amaçlı Bir Ara yüz,* Bilimde Modern Yöntemler Sempozyumu, Kocaeli.

ŞEN, Z., 2004, *Yapay Sinir Ağları İlkeleri*, Su Vakfı, İstanbul, 9756455136.

ŞENEL, F., 2003, *Beynin Gizemi* [Online], Ankara-Türkiye, *Bilim ve Teknik, Yeni Ufuklara,* http://www.biltek.tubitak.gov.tr/bdergi/ , [Ziyaret Tarihi: 8 Nisan 2011].

TEBELKIS, J., 1995, *Speech Recognition Using Neural Networks*, Doctor of Philosophy Thesis, School of Computer Science Carnegie Mellon University, Pennsylvania.

UÇAR, A., 2007, *Türkçe Fonemlerin Sınıflandırılmasında Kullanılan Sinir Ağının FPGA Uygulaması*, Yüksek Lisans Tezi, Hacettepe Üniversitesi, Elektrik ve Elektronik Mühendisliği Anabilim Dalı, İstanbul.

WEINSTEIN, R.K., LEE H., 2006, Architectures for High-Performance FPGA Implementations of Neural Models, *Insıtute of Physics Publishing Journal Of Neural Engineering*, 3 (2006) 21–34.

YILDIZ, Ö., 2006, *Döviz Kuru Tahmininde Yapay Sinir Ağlarının Kullanımı*, Yüksek Lisans Tezi, Eskişehir, Temmuz 2006.

EKLER

EK-A: ANNGEN İLE HAZIRLANAN ÖRNEK VHDL KODU 1

Aşağıda, ANNGEN ile üretilmiş birinci örneğin VHDL kodu görülmektedir. Üretilen kod çok uzun olduğundan, IP Core Generator ile üretilmiş ve bu kod'a ANNGEN tarafından dahil edilmiş FpMult8 çarpıcı ve FpAdd8 toplayıcı üniteleri ve bu tez çalışması kapsamında bir defa geliştirilmiş, Generic Register ile YSA 4 Pureline Normal VHDL kod parçaları kısaltılarak verilmiştir. Kısaltma yapılırken bu ünitelerin ENTITY ve ARCHITECTURE bölümleri içinden kısımlar çıkartılarak yerlerine ":" konulmuştur.

```
------------------------------------------------------------
-- Copyright 2011 by Duzce University All rights reserved.
--
-- Design:    FpMult8
-- Name:      İbrahim Şahin-N.Kemal Sarıtekin
--
-- Date  :    January 2011
-- Description:
-- Pipelined.
------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY FpMult8 IS
     :
END FpMult8;
ARCHITECTURE Behavioral OF FpMult8 IS
:
END Behavioral;
------------------------------------------------------------
-- Copyright 2011 by Duzce University All rights reserved.
--
-- Design:    FpAdd8
-- Name:      İbrahim Şahin-N.Kemal Sarıtekin
--
-- Date  :    January 2011
-- Description:
-- Pipelined.
------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY FpAdd8 IS
      :
END FpAdd8;
ARCHITECTURE Behavioral OF FpAdd8 IS
:
END Behavioral;
----------------------------------------------------------------
-- Copyright 2011 by Duzce University All rights reserved.
--
-- Design:    Generic Register
-- Name:      İbrahim Şahin-N.Kemal Sarıtekin
--
-- Date  :    January 2011
-- Description:
-- Pipelined.
----------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY RegGen IS
      :
END RegGen;
ARCHITECTURE Behavioral OF RegGen IS
      :
END Behavioral;
----------------------------------------------------------------
-- Copyright 2011 by Duzce University All rights reserved.
--
-- Design:    YSA 4 PURELIN NORMAL
-- Name:      İbrahim Şahin-N.Kemal Sarıtekin
--
-- Date  :    January 2011
-- Description:
-- Pipelined.
----------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity PLIN04 is
:
end PLIN04;
architecture RTL of PLIN04 is
      :
END RTL;
-----------------------------------------
-- Company:
-- Engineer:
```

```
--
-- Create Date:-- Fri May 27 14:52:38 2011
-- Design Name:
-- Module Name:
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Additional Comments:
--
----------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY Deneme IS
	PORT (
		Clock : in STD_LOGIC;
		Reset : in STD_LOGIC;
		LConst : in STD_LOGIC;
		LVar : in STD_LOGIC;
		DataIn : in STD_LOGIC_VECTOR (31 downto 0);
		DataOutC : out STD_LOGIC_VECTOR (31 downto 0);
		DataOutV : out STD_LOGIC_VECTOR (31 downto 0);
		OUT00_out : out STD_LOGIC_VECTOR (31 downto 0);
		OUT01_out : out STD_LOGIC_VECTOR (31 downto 0)
		);
END ENTITY Deneme;
ARCHITECTURE RTL OF Deneme IS
		--Internal data flow signal
		SIGNAL S_INP00_0 : STD_LOGIC_VECTOR (31 downto 0);
		SIGNAL S_INP01_0 : STD_LOGIC_VECTOR (31 downto 0);
		SIGNAL S_INP02_0 : STD_LOGIC_VECTOR (31 downto 0);
		SIGNAL S_INP03_0 : STD_LOGIC_VECTOR (31 downto 0);
		SIGNAL S_NEU00_1 : STD_LOGIC_VECTOR (31 downto 0);
		SIGNAL S_NEU01_1 : STD_LOGIC_VECTOR (31 downto 0);
		SIGNAL S_NEU02_1 : STD_LOGIC_VECTOR (31 downto 0);
		SIGNAL S_NEU03_1 : STD_LOGIC_VECTOR (31 downto 0);
		SIGNAL S_NEU00_2 : STD_LOGIC_VECTOR (31 downto 0);
		SIGNAL S_NEU01_2 : STD_LOGIC_VECTOR (31 downto 0);
		--Init Chain "IC" signals
		SIGNAL S_NEU00_IC_1 : STD_LOGIC_VECTOR (31 downto 0);
		SIGNAL S_NEU01_IC_1 : STD_LOGIC_VECTOR (31 downto 0);
		SIGNAL S_NEU02_IC_1 : STD_LOGIC_VECTOR (31 downto 0);
		SIGNAL S_NEU03_IC_1 : STD_LOGIC_VECTOR (31 downto 0);
		SIGNAL S_NEU00_IC_2 : STD_LOGIC_VECTOR (31 downto 0);
		SIGNAL S_NEU01_IC_2 : STD_LOGIC_VECTOR (31 downto 0);
		--EnN Singal Definitions
		SIGNAL EnN_Out_1_0 : STD_LOGIC;
		SIGNAL EnN_Out_1_1 : STD_LOGIC;
```

```
        SIGNAL EnN_Out_1_2 : STD_LOGIC;
        SIGNAL EnN_Out_1_3 : STD_LOGIC;
        SIGNAL EnN_Out_2_0 : STD_LOGIC;
        SIGNAL EnN_Out_2_1 : STD_LOGIC;
        SIGNAL EnN : STD_LOGIC;
BEGIN
    EnN <= '1';
        --Register Instantiations
        I_INPUT_0: entity work.RegGen Generic Map (32) Port Map (Clock, Reset, LVar,
    DataIn, S_INP00_0);
        I_INPUT_1: entity work.RegGen Generic Map (32) Port Map (Clock, Reset, LVar,
    S_INP00_0, S_INP01_0);
        I_INPUT_2: entity work.RegGen Generic Map (32) Port Map (Clock, Reset, LVar,
    S_INP01_0, S_INP02_0);
        I_INPUT_3: entity work.RegGen Generic Map (32) Port Map (Clock, Reset, LVar,
    S_INP02_0, S_INP03_0);
        --Neurons
        --Layer No : 1
        I_NEURON_1_0: entity work.PLIN04 Port Map (Clock, Reset, LConst, S_INP00_0,
    S_INP01_0, S_INP02_0, S_INP03_0, DataIn, S_NEU00_IC_1, S_NEU00_1, EnN,
    EnN_Out_1_0);
        I_NEURON_1_1: entity work.PLIN04 Port Map (Clock, Reset, LConst, S_INP00_0,
    S_INP01_0, S_INP02_0, S_INP03_0, S_NEU00_IC_1, S_NEU01_IC_1, S_NEU01_1, EnN,
    EnN_Out_1_1);
        I_NEURON_1_2: entity work.PLIN04 Port Map (Clock, Reset, LConst, S_INP00_0,
    S_INP01_0, S_INP02_0, S_INP03_0, S_NEU01_IC_1, S_NEU02_IC_1, S_NEU02_1, EnN,
    EnN_Out_1_2);
        I_NEURON_1_3: entity work.PLIN04 Port Map (Clock, Reset, LConst, S_INP00_0,
    S_INP01_0, S_INP02_0, S_INP03_0, S_NEU02_IC_1, S_NEU03_IC_1, S_NEU03_1, EnN,
    EnN_Out_1_3);
        --Layer No : 2
        I_NEURON_2_0: entity work.PLIN04 Port Map (Clock, Reset, LConst, S_NEU00_1,
    S_NEU01_1, S_NEU02_1, S_NEU03_1, S_NEU03_IC_1, S_NEU00_IC_2, OUT00_out, EnN,
    EnN_Out_2_0);
        I_NEURON_2_1: entity work.PLIN04 Port Map (Clock, Reset, LConst, S_NEU00_1,
    S_NEU01_1, S_NEU02_1, S_NEU03_1, S_NEU00_IC_2, S_NEU01_IC_2, OUT01_out, EnN,
    EnN_Out_2_1);
END ARCHITECTURE RTL;
```

EK-B: ANNGEN İLE HAZIRLANAN ÖRNEK VHDL KODU 2

Aşağıda ANNGEN ile üretilmiş ikinci örnek VHDL kodu görülmektedir. Bu kod da yine birinci örnekte olduğu gibi kısaltılarak verilmiştir.

```
----------------------------------------------------------------
-- Copyright 2011 by Duzce University, All rights reserved.
--
-- Design:    FpMult8
-- Name:İbrahim Şahin-N.Kemal Saritekin
--
-- Date  :    January 2011
-- Description:
-- Pipelined.
----------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY FpMult8 IS
     :
END FpMult8;
ARCHITECTURE Behavioral OF FpMult8 IS
:
END Behavioral;
----------------------------------------------------------------
-- Copyright 2011 by Duzce University, All rights reserved.
--
-- Design:    FpAdd8
-- Name:İbrahim Şahin-N.Kemal Saritekin
--
-- Date  :    January 2011
-- Description:
-- Pipelined.
----------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY FpAdd8 IS
     :
END FpAdd8;
ARCHITECTURE Behavioral OF FpAdd8 IS
:
END Behavioral;
----------------------------------------------------------------
-- Copyright 2011 by Duzce University, All rights reserved.
--
```

```
-- Design:     Generic Register
-- Name:İbrahim Şahin-N.Kemal Saritekin
--
-- Date  :     January 2011
-- Description:
-- Pipelined.
----------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY RegGen IS
      :
END RegGen;
ARCHITECTURE Behavioral OF RegGen IS
      :
END Behavioral;
----------------------------------------------------------------
-- Copyright 2011 by Duzce University, All rights reserved.
--
-- Design:     YSA 2 PURELIN NORMAL
-- Name:       İbrahim Şahin-N.Kemal Saritekin
--
-- Date  :     January 2011
-- Description:
-- Pipelined.
----------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity PLIN02 is
   :
end PLIN02;
architecture RTL of PLIN02 is
      :
END RTL;
----------------------------------------------------------------
-- Copyright 2011 by Duzce University, All rights reserved.
--
-- Design:     YSA 4 HARDLIMS NORMAL
-- Name:       İbrahim Şahin-N.Kemal Saritekin
--
-- Date  :     January 2011
-- Description:
-- Pipelined.
----------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
entity HLMS04 is
   :
end HLMS04;
architecture RTL of HLMS04 is
      :
END RTL;
----------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:-- Thu Jun 09 00:26:00 2011
-- Design Name:
-- Module Name:
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Additional Comments:
--
----------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY Deneme1 IS
        PORT (
                Clock : in STD_LOGIC;
                Reset : in STD_LOGIC;
                LConst : in STD_LOGIC;
                LVar : in STD_LOGIC;
                DataIn : in STD_LOGIC_VECTOR (31 downto 0);
                DataOutC : out STD_LOGIC_VECTOR (31 downto 0);
                DataOutV : out STD_LOGIC_VECTOR (31 downto 0);
                OUT00_out : out STD_LOGIC_VECTOR (31 downto 0)
                );
END ENTITY Deneme1;
ARCHITECTURE RTL OF Deneme IS
               --Internal data flow signal
               SIGNAL S_INP00_0 : STD_LOGIC_VECTOR (31 downto 0);
               SIGNAL S_INP01_0 : STD_LOGIC_VECTOR (31 downto 0);
               SIGNAL S_NEU00_1 : STD_LOGIC_VECTOR (31 downto 0);
               SIGNAL S_NEU01_1 : STD_LOGIC_VECTOR (31 downto 0);
               SIGNAL S_NEU02_1 : STD_LOGIC_VECTOR (31 downto 0);
               SIGNAL S_NEU03_1 : STD_LOGIC_VECTOR (31 downto 0);
               SIGNAL S_NEU00_2 : STD_LOGIC_VECTOR (31 downto 0);
               SIGNAL S_NEU01_2 : STD_LOGIC_VECTOR (31 downto 0);
               SIGNAL S_NEU00_3 : STD_LOGIC_VECTOR (31 downto 0);
               --Init Chain "IC" signals
               SIGNAL S_NEU00_IC_1 : STD_LOGIC_VECTOR (31 downto 0);
               SIGNAL S_NEU01_IC_1 : STD_LOGIC_VECTOR (31 downto 0);
```

```
        SIGNAL S_NEU02_IC_1 : STD_LOGIC_VECTOR (31 downto 0);
        SIGNAL S_NEU03_IC_1 : STD_LOGIC_VECTOR (31 downto 0);
        SIGNAL S_NEU00_IC_2 : STD_LOGIC_VECTOR (31 downto 0);
        SIGNAL S_NEU01_IC_2 : STD_LOGIC_VECTOR (31 downto 0);
        SIGNAL S_NEU00_IC_3 : STD_LOGIC_VECTOR (31 downto 0);
        --EnN Singal Definitions
        SIGNAL EnN_Out_1_0 : STD_LOGIC;
        SIGNAL EnN_Out_1_1 : STD_LOGIC;
        SIGNAL EnN_Out_1_2 : STD_LOGIC;
        SIGNAL EnN_Out_1_3 : STD_LOGIC;
        SIGNAL EnN_Out_2_0 : STD_LOGIC;
        SIGNAL EnN_Out_2_1 : STD_LOGIC;
        SIGNAL EnN_Out_3_0 : STD_LOGIC;
        SIGNAL EnN : STD_LOGIC;
BEGIN
    EnN <= '1';
        --Register Instantiations
        I_INPUT_0: entity work.RegGen Generic Map (32) Port Map (Clock, Reset, LVar,
    DataIn, S_INP00_0);
        I_INPUT_1: entity work.RegGen Generic Map (32) Port Map (Clock, Reset, LVar,
    S_INP00_0, S_INP01_0);
        --Neurons
        --Layer No : 1
        I_NEURON_1_0: entity work.PLIN02 Port Map (Clock, Reset, LConst, S_INP00_0,
    S_INP01_0, DataIn, S_NEU00_IC_1, S_NEU00_1, EnN, EnN_Out_1_0);
        I_NEURON_1_1: entity work.PLIN02 Port Map (Clock, Reset, LConst, S_INP00_0,
    S_INP01_0, S_NEU00_IC_1, S_NEU01_IC_1, S_NEU01_1, EnN, EnN_Out_1_1);
        I_NEURON_1_2: entity work.PLIN02 Port Map (Clock, Reset, LConst, S_INP00_0,
    S_INP01_0, S_NEU01_IC_1, S_NEU02_IC_1, S_NEU02_1, EnN, EnN_Out_1_2);
        I_NEURON_1_3: entity work.PLIN02 Port Map (Clock, Reset, LConst, S_INP00_0,
    S_INP01_0, S_NEU02_IC_1, S_NEU03_IC_1, S_NEU03_1, EnN, EnN_Out_1_3);
        --Layer No : 2
        I_NEURON_2_0: entity work.HLMS04 Port Map (Clock, Reset, LConst, S_NEU00_1,
    S_NEU01_1, S_NEU02_1, S_NEU03_1, S_NEU03_IC_1, S_NEU00_IC_2, S_NEU00_2, EnN,
    EnN_Out_2_0);
        I_NEURON_2_1: entity work.HLMS04 Port Map (Clock, Reset, LConst, S_NEU00_1,
    S_NEU01_1, S_NEU02_1, S_NEU03_1, S_NEU00_IC_2, S_NEU01_IC_2, S_NEU01_2, EnN,
    EnN_Out_2_1);
        --Layer No : 3
        I_NEURON_3_0: entity work.PLIN02 Port Map (Clock, Reset, LConst, S_NEU00_2,
    S_NEU01_2, S_NEU01_IC_2, S_NEU00_IC_3, OUT00_out, EnN, EnN_Out_3_0);

END ARCHITECTURE RTL;
```

ÖZGEÇMİŞ

Kişisel Bilgiler

Soyadı, Adı : SARITEKİN, Namık Kemal

Uyruğu : T.C

Doğum Tarihi Ve Yeri : 06.11.1970, Gediz/KÜTAHYA

Medeni Hali : Evli, iki çocuk babası

E-mail : tekinfizikster@gmail.com

Eğitim

Derecesi	**Eğitim Birimi**	**Mezuniyet Tarihi**
Lisans	Orta Doğu Teknik Üniversitesi	
	Eğitim Fak. Fizik Böl.	1993
Lise	Yunus Emre Öğretmen Lisesi	1987

İş Deneyimi

Yıl	**Yer**	**Görev**
2006-Halen	Düzce Fen Lisesi-Düzce	Fizik Öğretmeni
2001-2006	Torbalı Anadolu Lisesi-İzmir	Fizik Öğretmeni
1994-2001	M.A.Çulcuoğlu A.L.-Ş.Urfa	Fizik Öğretmeni
1993-1994	Gediz Anadolu Lisesi-Kütahya	Fizik Öğretmeni

Yabancı Dil

İyi derecede ingilizce

Hobiler

Kitap okumak, seyahat etmek, bilgisayar teknolojilerini takip etmek, masa tenisi oynamak.