KOCAELİ ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

FPGA İLE YAPAY SİNİR AĞININ DONANIMSAL GERÇEKLENMESİ

YÜKSEK LİSANS TEZİ

Elektronik ve Hab. Müh. Suhap ŞAHİN

Anabilim Dalı: Bilgisayar Mühendisliği

Danışman: Yrd.Doç.Dr. Yaşar BECERİKLİ

HAZİRAN 2004

KOCAELİ ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

FPGA İLE YAPAY SİNİR AĞININ DONANIMSAL GERÇEKLENMESİ

YÜKSEK LİSANS TEZİ

Elektronik ve Hab. Müh. Suhap ŞAHİN

Tezin Enstitüye Verildiği Tarih : 4 Haziran 2004

Tezin Savunulduğu Tarih : 12 Temmuz 2004

Tez Danışmanı	Üye	Üye
Yrd.Doc.Dr. Yaşar BECERİKLİ	Doç.Dr.A.Coşkun SÖNMEZ	Yrd.Doc.Dr.Ali TANGEL
(.....)	(.....)	(.....)

HAZİRAN 2004

FPGA İLE YAPAY SİNİR AĞININ DONANIMSAL GERÇEKLENMESİ

Suhap ŞAHİN

Anahtar Kelimeler: FPGAs, Yapay Sinir Ağları, Donanım Tanımlama Dilleri, Paralel Programlama, Tekrar Düzenlenebilir Hesaplamalar, Kayan Noktalı Aritmetik

Özet Yapay sinir ağlarının FPGA ile gerçeklenmesi programlanabilir sistemlerde esneklik sağlar. Eğer gerçek zamanlı uygulamalar için VLSI teknolojisi kullanılarak bir yapay sinir ağı tabanlı işlemci yapılmak istenirse, bu gerçekleştirim hem zaman hem de maliyet açısından oldukça masraflı olduğu görülecektir.

FPGA tabanlı YSA'da düşük sayı duyarlılığı kullanılarak yapılan tasarımlar,VLSI ile yapılan YSA tasarımlarına göre hem zaman hem maliyet açısından çok büyük avantajları olduğu gözlenir. Ayrıca düşük sayı duyarlılığı gerçek zamanlı yapay sinir ağları uygulamalarının test aşamasında sınıflandırma problemlerinde oldukça başarılı sonuçlar verdiği gözlenmiştir.

Tekrar düzenlenebilir FPGA programlanması ile özel amaçlı hızlı donanımlar çok geniş uygulamalar için kullanılabilecektir. FPGA'ların, geleneksel işlemcilerin sahip olmadığı hız, güvenlik ve paralel işlem yapabilme yeteneğine ve ayrıca VLSI teknolojisinin sahip olmadığı tekrar düzenlenebilirlik kabiliyetlerine sahip olması vasıtasıyla yapay sinir ağlarıyla çok uyumlu çalışmalar yapılabilmekte ve yeni yapay sinir ağı algoritmalarına ışık tutmaktadır.

Bu çalışmanın amacı FPGA kullanarak YSA nı donanım ile gerçeklenmesidir. Dijital sistem mimarisi, ileri sürümlü çok katmanlı YSA gerçeklemek için tasarlanmıştır. Tasarım mimarisi VHDL (Very High Speed Integrated Circuits Hardware Description Language) tanımlandı ve FPGA entegre devresi üzerinde gerçeklenmiştir. Tasarım FPGA demo kart üzerinde test edilmiştir.

NEURAL NETWORK HARDWARE IMPLEMENTATION USING FPGA

Suhap ŞAHİN

Key words: FPGAs, Artificial Neural Network, Hardware Description Language, Parallel Programming, Reconfigurable Computing, Floating Point Arithmetic.

Abstract: The FPGAs approach for neural network implementation provides flexibility in programmable systems. For the neural based instrument prototype in real time application, conventional specific VLSI neural chip desing suffer the limitation in time and cost.

With low precision artificial neural network desing, FPGAs have higher speed and smaller size for real time application than that of the VLSI design. In addition, artificial neural network based on FPGAs has fairly achieved with classification application.

The programmability of reconfigurable FPGAs yields the availability of fast special purpose hardware for wide applications. Its programmability could set the conditions to explore new neural network algorithms and problems of a scale that would not be feasible with conventional processor.

The goal of this work is to implementation a hardware of a neural network using Field Programmable Gate Array (FPGA). A digital system architecture is presented using Very High Speed Integrated Circuits Hardware Description Language (VHDL) and is implemented in FPGA chip. The design is tested on an FPGA demo board.

ÖNSÖZ ve TEŞEKKÜR

FPGA tabanlı yapay sinir ağları (YSA) diğer işlemcilerle gerçeklenen yapay sinir ağlarıyla karşılaştırıldığında ilk göze çarpan FPGA üzerinde gerçeklenen mimarinin çok amaçlı hesaplamaları yapabilmesidir. ASIC tümleşik devreler üzerinde gerçekleştirilen yapay sinir ağları (YSA) ise, sadece tek bir amaca hizmet verebilecek kabiliyete sahiptirler.

FPGA'ler programlama kabiliyetleri yönüyle DSP ve mikroişlemcilere göre sınırlıdırlar. Bununla birlikte bir çok CAD (Bilgisayar destekli çizim) aracı FPGA tasarımını desteklemekte ve bu sayede kodlanan devreler, lojik elemanlarıyla bir devre çizimi olarak gösterilmektedir. VHDL' in tasarımcılara verdiği destekle, tasarımlar çok daha çabuk ve etkili bir şekilde gerçekleştirilir.

Bu çalışmada FPGA üzerinde gerçekleşen bir yapay sinir ağı tasarımı, simülasyonu ve akış şeması verilmiştir. YSA(Yapay sinir ağı) tasarımı VHDL(Very high hardware desciription language) dili kullanılarak Xilinx ISE ve ModelSim programlarının yardımı ile yapılmıştır. YSA mimarisinde bulunan çarpma ve toplama işlemleri kayan noktalı sayılar kullanılarak yazılan algoritmalarla FPGA üzerine yüklenmiştir. Bölme işlemini yerine ise RAM yapısı kullanılmıştır. Kullanılan algoritmalarda kayan noktalı sayıların hassasiyeti IEEE 754 formatına uygun olarak 32 bit olarak alınmıştır. Kullanılan sayıların hassasiyet derecelerinin yüksek olması, algoritmaların FPGA üzerinde çok büyük yer kaplaması sonucunu doğurmuştur. Alan sorunu sebebiyle YSA mimarisi seri olarak tasarlanmıştır.

Tez çalışmalarına ayırdığım zamanı anlayışla karşılayıp beni destekleyen eşime, bir ömür boyunca desteklerini bir an için bile eksiltmeyen aileme ve bu güne kadar öğrettikleriyle bana katkıda bulunan herkese teşekkür ederim.

İÇİNDEKİLER

SİMGELER DİZİNİ ve KISALTMALAR

FPGA	Field Programable Gate Array
VHDL	Very High Application Specific Integrated Circuit Hardware Desciription Language
YSA	Yapay Sinir Ağları
YSH	Yapay Sinir Hücresi
CAD	Computer Aided Design
ASIC	Application Specific Integrated Circuit
SRAM	: Static Random Access Memory
PROM	: Programable Read Only Memory
EPROM	: Erasable Programable Read Only Memory
EEPROM	: Electrically Erasable Programable Read Only Memory
PLD	: Programable Logic Device
PAL	: Programable Array Logic
PLA	: Programable Logic Array
SPLD	: Simple Programable Logic Device
CPLD	: Complex Programable Logic Device
MPGA	: Mask Programable Gate Array
LUT	: Look-up Table
EDIF	: Electronik Design Interchange Format
CLB	: Configurable Logic Blok

ŞEKİLLER DİZİNİ

TABLOLAR DİZİNİ

BÖLÜM 1. GİRİŞ

Yapay sinir ağları(YSA), model(desen) tanıma, görüntü işleme ve tıbbi teşhis gibi alanlarda oldukça zor problemleri çözebilecek yeteneğe sahiptir. Biyolojik yapıdan esinlenilmiş YSA doğasında dağınık paralel işleme gerektirir. Bu sebeple, gerçek zamanlı ve çok yüksek hızlı uygulamalar yapay sinir ağları ile gerçeklenmeye çalışılır. Bu çalışmalar gerçek performanslarını ancak paralel çalışan mimariler üzerinde gösterebilirler.

YSA'nın gerçekleştirimi, yapay sinir hücresi (nöron) modelinin donanıma etkin bir şekilde haritalanması işlemidir. YSA konusundaki araştırmaları iki kategoride inceleyebiliriz.

Genel amaçlı YSA işlemcileri, çok çeşitlilikteki YSA'nın benzetimi için gerçekleştirilir. Genel amaçlı YSA'ı esneklik ve çeşitlilik sağlar. Özel amaçlı VLSI sistemleri özel bir YSA için tasarlanırlar. Özel amaçlı VLSI tasarımları yüksek hız ve yoğun hesaplama isteyen tipik uygulamaları sağlar. Bu alanda oldukça çeşitli uygulama çalışmaları ve teknikleri mevcuttur. (Ramache 1992, Burr 1993, Rucket et al 1993).

Özel amaçlı VLSI tümleşik devreler gerçek zamanlı uygulamalarda işlem hızını ve yoğunluğunu artırma problemlerinin üstesinden gelmiştir. Fakat çeşitli YSA mimarilerinden sadece biri gözetilerek gerçeklenen tümleşik devre üzerinde farklı YSA mimarilerinin kullanılamayışı ve üretim aşamalarının hem maliyetli hem de çok zaman almasından dolayı bu tarz tümleşik devreler yaygın olarak kullanım alanı bulamamışlardır.

VLSI tümleşik devreler üzerinde gerçeklenen uygulamanın sonradan değiştirilme şansı yoktur. VLSI tasarımlar uygulamaya özel olduğu için sadece tasarlandığı

uygulama için esneklik mevcuttur. FPGA'lar sayesinde uygulamaya özel tasarımlar yapılabilir olmuştur. Tasarım içindeki değişiklikleri birkaç saat içinde başarabilen FPGA sayesinde zamandan ve maliyetten önemli kazanımlar olmuştur (Cox and Blanz 1992).

1.1 Yapay Sinir Ağları Donanım Mimarisi

YSA modelinde, her yapay sinir hücresi(YSH) ağırlıklaştırılmış girişlerini toplar ve bu toplamı doğrusal olmayan bir fonksiyondan geçirerek sonuca ulaşır. Her giriş ya bir başka YSA'nın çıkışıdır ya da dış dünyadan alınmış bir giriştir. Yapısal parametreler, örneğin katman sayısı, YSH sayısı ve iletkenliği uygulamadan uygulamaya önemlilik arz edebilir.

Özel amaçlı YSA donanımları, günümüz teknolojisinde etkili gerçeklenmeli ve dikkatlice eğitilmelidir. Herhangi bir YSA gerçeklenmesini iki safhada inceleyebiliriz. Öğrenme aşaması, topoloji ve ağırlıkların geriye yayılım algoritmaları kullanarak kararlı hale getirilmesi. Sonradan düzeltme veya test etme safhası ki bu safhada ağ eğitimi sırasındaki ağırlıklarıyla ve yapısında sakladığı temel bilgilerle örnekleri tanır. Bu şekilde düşünülürse YSA gerçeklenmesi dağılmış öğrenme ve geri dönüşüm safhasıdır.

Öğrenme aşamasındaki alt yapı ihtiyaçları ve hesaplamalar test etme aşamasındaki alt yapı ihtiyaçlara ve hesaplamalara göre çok daha karmaşıktır. Donanımdaki ağ parametreleri ögrenme safhasında sabitlenir. Öğrenme sonrası, bu parametreler test görevi için donanıma yüklenir.

Eğer bir YSA'nı donanımsal olarak gerçeklemek istiyorsak; özellikle öğrenme aşamasında hesaplamalara katılan sayıların (işlem elemanının) duyarlılığının yer, güç ve performans üzerinde önemli etkisi olabilir. Etkili donanım gerçeklenmesi için uygun aritmetik çözümlemeyi uyumlu bir şekilde yerleştirerek baraşıya ulaşılabilir. Tecrübeler ve teorik çalışmalara dayanılarak YSH'ine ait duyarlılık sayısının çeşitlilik gösterebileceğini söyleyebiliriz (Stevenson et al 1990).

Araştırmaların sonuçlarına göre YSA'nın eğitimi aşamasında ağa girdi olarak verilecek eğitim setindeki sayıların ve ağın mimarisinin barındırdığı işlemlerde kullanılan sayıların sayı duyarlılığının fazla olması ağın eğitimi için gerekli olduğunun göstermiştir. Tam eğitilmiş ağın test aşamasında ise eğitim esnasında hesaplanan ağırlıklara ait sayıların duyarlılıkları düşük olabilir. Özellikle sınıflandırma için kullanılan ağlar bu halleriyle test yapabilecek yetenektedir (Lisa et al 1993).

İşlem hızını artırmak ve donanım karmaşasından kurtulmak için tipik YSA gerektirdiği işlem hesaplama hassasiyetini (ağın yeteneğini kaybetmeyecek şekilde) azaltarak donanım haritalanması optimize edilmelidir.

1.2 FPGA'lar Tabanlı Ağ Mimarisi

FPGA mimarisi üzerinde en etkili şekilde gerçekleştirilen ağlar sınıflandırma yapan ağlardır. Genellikle sınıflandırma uygulamasında en çok kullanılan YSA modeli ileri beslemeli ağ ile geriye yayılım öğrenme algoritmasıdır. YSH'leri ağda katmanlar halinde konumlandırılmışlardır. Bir katmanda bulunan herbir yapay sinir hücresine ulaşan girişler ya bir önceki katmandan ya da dışarıdan gelir. Test safhasında bir sinir ağı tarafından yapılan temel işlem toplamadır. Toplamadan sonra toplamlar bir doğrusal olmayan fonksiyondan geçirilirler (Yu and Deni 1994).

FPGA'de haritalanma yapılırken paralellikten mümkün olan en büyük ölçüde yararlanılması beklenir. Yer tasarrufu ve hız artırımı için YSA yapısında kullanılacak doğrusal olmayan fonksiyonun çeşitli girişlere vereceği çıkışlar bir tablodan elde edilebilir.

1.3 YSA'nın FPGA Kullanılarak Gerçeklenmesi Alanındaki Çalışmalar

Guccione ve Gonzalez (1993) bu makinalar için geleneksel bir programlama modeli önerdiler. Bu model hesaplamalarda vektör tabanlı veri-paralel yaklaşımıdır. Bu model, yüksek seviyeli dillerede (örneğin C) yazılan algoritmaları, yüksek

performanslı dijital devrelere dönüştürür. Bu makalede bu teknik ileri beslemeli yapay sinir ağının FPGA'de gerçekleştirimi için kullanılacaktır.
Zhu et al (1999) tablo (LUT-Look-up-table) tabanlı çarpma devreleri ile YSA'nın kalıtımsal paralelliğinin başarılı bir şekilde gerçeklenebileceğini göstermişlerdir.

Yu and Deni (1994) Kalp hastalarının sınıflandırılması için taşınabilir dijital bir sistem prototipi geliştirmişlerdir. Bu sistem eğitilmiş bir ağın Xilinx XC 4000 serisi FPGA'ın üzerinde gerçeklenmesiyle oluşturulmuştur.

Nichols et al (2002) ileri beslemeli bir yapay sinir ağını Xilinx Virtex 2E FPGA'yi üzerinde gerçekleciler. Bu gerçekleştirim sırasında kayan noktalı sayı duyarlılığı ile sabit noktalı sayı duyarlılığının karşılaştırlmasını yaptılar ve geri yayılım algoritması için en uygun sayı sisteminin 16 bit duyarlıklı sabit noktalı sayılar olduğunu kanıtladılar.

Bölüm 2'de YSA kavramı, YSA nın gelişimi ve YSA çekici kılan özelliklerden bahsedilmiştir. Ayrıca YSA'nın matematiksel modellemesi ve ileri beslemeli YSA hakkında bilgi verilmiştir.

Bölüm 3'de programlanabilir lojik elemanların gelişimi süreci anlatılmış. FPGA mimarileri hakkında bilgi verilmiştir. VHDL donanım tanımlama dilinin genel yapısı uygulanması anlatılmıştır.

Bölüm 4'de YSA'nın, VHDL kullanılarak FPGA üzerinde gerçeklenmesi anlatılmıştır. Gerçekleştirim esnasında kullanılan kayan noktalı sayıların aritmetiği hakkında bilgi verilmiştir. Gerçekleştirimin mimari yapısı ve işleyişi hakkında bilgi verilmiştir.

Sonuç ve öneriler bölümünde ise YSA nı gerçeşleştirmek için, kayan noktalı sayılar aritmetiğinin FPGA üzerinde kullanılmasının avantajları ve dezavantajları hakkında bilgi verilmiştir.

BÖLÜM 2. YAPAY SİNİR AĞLARI

2.1 Biyolojik Sinir Sistemi

Biyolojik sinir sistemi, merkezinde sürekli olarak bilgiyi alan, yorumlayan ve uygun bir karar üreten beynin (merkezi sinir ağı) bulunduğu 3 katmanlı bir sistem olarak açıklanır. Alıcı sinirler (receptor) organizma içerisinden ya da dış ortamlardan algıladıkları uyarıları, beyine bilgi ileten elektriksel sinyallere dönüştürür. Tepki sinirleri (effector) ise, beynin ürettiği elektriksel darbeleri organizma çıktısı olarak uygun tepkilere dönüştürür. Şekil 2.1 de bir sinir sisteminin blok gösterimi verilmiştir (Haykin 1999) (Arabib 1987).

Şekil 2.1. Biyolojik sinir sisteminin blok gösterimi

2.2 Yapay Sinir Ağı Nedir?

İnsan beyninin nasıl çalıştığı ve fonksiyonları uzun yıllar araştırılmıştır.1980 yılında beyinin fonksiyonları hakkında bilgi veren ilk eser yayınlanmıştır (Sharda 1994). Bu yıllardan sonra çalışmalar mühendislik alanlarına kaydırılmaya başlanmış ve günümüzdeki yapay sinir ağı temelleri oluşturulmuştur (Öztemel 2003).

Beynin üstün özellikleri, bilim adamlarını üzerinde çalışmaya zorlamış ve beynin nörofiziksel yapısından esinlenerek matematiksel modeli çıkarılmaya çalışılmıştır. Beynin bütün davranışlarını tam olarak modelleyebilmek için fiziksel bileşenlerinin

doğru olarak modellenmesi gerektiği düşüncesi ile çeşitli yapay hücre ve ağ modelleri geliştirilmiştir.

YSA kavramı beynin çalışma ilkelerinin sayısal bilgisayarlar üzerinde taklit edilmesi fikri ile ortaya çıkmıştır. İlk çalışmalar beyni oluşturan biyolojik hücrelerin, ya da literatürdeki ismiyle nöronların matematiksel olarak modellenmesi üzerine yoğunlaşmıştır. Bu çalışmaların ortaya çıkardığı bulgular, her bir YSH'nin komşu YSH'den bazı bilgiler aldığı ve bu bilgilerin biyolojik sinir hücresi dinamiğinin öngördüğü biçimde bir çıktıya dönüştürüldüğü şeklinde idi. Bu gün YSA olarak adlandırılan alan, birçok YSH'nin belirli biçimlerde bir araya getirilip bir işlevin gerçeklenmesinin yapısal olduğu kadar matematiksel yanıtlarını arayan bir bilim dalı olmuştur.

YSA, YSH'lerinin birbirleri ile çeşitli şekillerde bağlanmasından oluşur ve genellikle katmanlar şeklinde düzenlenir. Donanım olarak elektronik devrelerle ya da bilgisayarlarda yazılım olarak gerçeklenebilir. Beynin bilgi işleme yöntemine uygun olarak YSA, bir öğrenme sürecinden sonra bilgiyi toplama, hücreler arasındaki bağlantı ağırlıkları ile bu bilgiyi saklama ve genelleme yeteneğine sahip paralel dağıtık bir işlemcidir. Öğrenme süreci, arzu edilen amaca ulaşmak için YSA ağırlıklarının yenilenmesini sağlayan öğrenme algoritmalarını ihtiva eder.

İnsan beyninin çalışma mekanizmasını taklit etmeye çalışan bu sistemler, her ne kadar günümüz teknolojisinin ürettiği, birim işlem zamanı nanosaniyeler mertebesinde olan silikon kapılarla gerçeklenebilseler de insan beyninin birim işlem zamanından ve sinir hücrelerinin toplu biçimde ele alındıklarındaki işlevselliklerinden oldukça uzakta kalırlar. Örneğin bir görme eylemini düşünürsek, bu eylemde gerçekleştirilen bilgi-işleme hızı günümüz bilgisayarlarının asla yarışamayacağı ölçüttedir (Churchland and Sejnowski 1992). YSA, karar hızı açısından insan beyni ile yarışabilecek aşamaya henüz ulaşamamalarına rağmen, karmaşık eşleştirmelerin hassas biçimde gerçeklenebilmesi ve yapısal gürbüzlüğe sahip olmaları nedeniyle, gün geçtikce uygulama alanları genişlemektedir.

Yapay sinir ağlar konusu üzerinde çalışırken, bir ağ yapısının çözebileceği problem uzayının, insan beyninin çözebileceği problem uzayının oldukça kısıtlanmış bir alt kümesi olacağı gözden kaçırılmamalıdır. Diğer bir deyişle insan beyninin bilgiyi işlemedeki, kavramların ilişkilendirilmesindeki ve çıkarım mekanizmasındaki üstünlüğü günümüze kadar tartışılan yaklaşımlarla çürütülebilecek bir sav değildir.

2.3 Yapay Sinir Ağları Kavramını Çekici Kılan Özellikler

YSA'nın hesaplama ve bilgi işleme gücünü, paralel dağıtık yapısından, öğrenebilme ve genelleme yeteneğinden aldığı söylenebilir. Genelleme, eğitim ya da öğrenme sürecinde karşılaşılmayan girişler için de YSA'nın uygun tepkileri üretmesi olarak tanımlanır. Bu üstün özellikleri, YSA'nın karmaşık problemleri çözebilme yeteneğini gösterir.

2.3.1 Doğrusal olmama

YSA'ların yapıları gereği doğrusal ağlar olduğu gibi, daha çok doğrusal olmayan yönleriyle öne çıkmıştır. Ağın yada temel işlem elemanı olan hücrenin, doğrusallığı aktivasyon fonksiyonu Şekil 2.3 ile belirlenir. Bu özelliği ile YSA, özelikle doğrusal olmayan karmaşık problemlerin çözümünde en önemli araç durumuna gelmiştir.

2.3.2 Öğrenme

YSA' nın arzu edilen davranışı gösterebilmesi için amaca uygun olarak tasarlanması gerekir. Bu durum, hücreler arasında doğru bağlantıların yapılması ve bağlantıların uygun ağırlıklara sahip olması gerektiğini ifade eder. YSA'nın karmaşık ve lineer olmayan yapısı nedeniyle bağlantılar ve ağırlıklar önceden ayarlı olarak verilemez yada tasarlanamaz. Genellikle ağırlıklar, rasgele yada sabit bir değerde seçilir. YSA, istenen davranışı gösterecek şekilde ilgilendiği problemden aldığı eğitim örneklerini kullanarak problemi öğrenmelidir. Belli bir hata kriterine ve öğrenme algoritmasına

göre, ağırlıkların yenilenerek, artık değişmediği durumda öğrenmenin gerçekleştiği söylenebilir.

2.3.3 Uyarlanabilirlik

YSA, ilgilendiği problemdeki değişikliklere göre ağırlıklarını ayarlar. Yani, belirli bir problemi çözmek amacıyla eğitilen YSA, problemdeki değişimlere göre tekrar eğitilebilir, değişimler devamlı ise gerçek zamanda da eğitime devam edilebilir. Bu özelliği ile YSA, sinyal işleme, uyarlamalı sistem tanıma ve kontrol gibi alanlarda etkin olarak kullanılırlar. Uyarlanabilir sistemler çok kısa zaman aralıklarında ağırlıklarını bir çok değişikliğe uğratırlar, bu sebeple bozucu etkilere cevap vermeye yatkındırlar. Bu ise sistem performansını büyük ölçüde etkiler. Uyumluluk özelliklerini tam anlamıyla kullanmak için, sistemin alacağı ilk ağırlık değerleri çevreden gelen istenilen değişimlere cevap verecek kadar küçük ve sitemin bozucu etkilerden etkilenmemesini sağlayacak kadar büyük olmalıdır (Grossberg 1988).

2.3.4 Genelleme

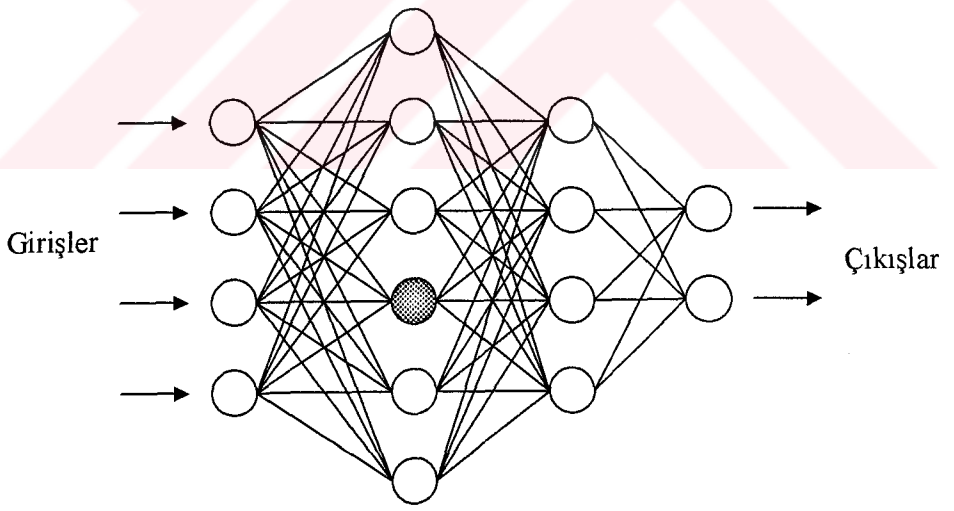
Ağ yapısının, eğitim esnasında kullanılan nümerik bilgilerden eşleştirmeyi betimleyen kaba özellikleri çıkarabilmesi ve böylelikle eğitim sırasında kullanılmayan girdiler için de anlamlı yanıtlar üretemesidir. YSA, ilgilendiği problemi öğrendikten sonra eğitim sırasında karşılaşmadığı test örnekleri için de arzu edilen tepkiyi üretebilir. Örneğin, karakter tanıma amacıyla eğitilmiş bir YSA, bozuk karakter girişlerinde de doğru karakterleri verebilir yada bir sistemin eğitilmiş YSA modeli, eğitim sürecinde verilmeyen giriş sinyalleri için de sistemle aynı davranışı gösterebilir (Haykin 1999).

2.3.5 Paralellik

YSA'nın paralelliği toplamsal işlevin yapısal olarak dağılmışlığıdır (Haykin 1994). Diğer bir deyişle bir çok nöron eş zamanlı olarak çalışır ve karmaşık bir işlev bir çok küçük nöron aktivitesinin bir araya gelmesinden oluşur.

2.3.6 Hata toleransı

YSA, çok sayıda hücrenin çeşitli şekillerde bağlanmasından oluştuğundan paralel dağıtık bir yapıya sahiptir ve ağın sahip olduğu bilgi, ağdaki bütün bağlantılar üzerine dağılmış durumdadır. Bu nedenle, eğitilmiş bir YSA'nın bazı bağlantılarının hatta bazı hücrelerinin etkisiz hale gelmesi, ağın doğru bilgi üretmesini önemli ölçüde etkilemez. Bu nedenle, geleneksel yöntemlere göre hatayı indirgeme yetenekleri son derece yüksektir (Haykin 1999). Koyu nöronun işlevsiz kalması partikten kaynaklanan görev paylaşımından dolayı başarımı dikkate değer ölçüde etkilemeyecektir (Bkz. Şekil 2.2).



Şekil 2.2. YSA'rında hata toleransı

2.3.7 Donanım ve hız

YSA, paralel yapıya sahiptir. Bu sebeple seri çalışan işlemciler üzerinde gerçeklenen mimariler gerçek performanslarını sergileyemezler. YSA'nın hızlı bilgi işleme yeteneğini artırmak ve gerçek zamanlı uygulamalarda kullanılabilmesini mümkün kılmak için büyük ölçekli entegre devre (VLSI) teknolojisinden yaralanılmaktadır (Mead 1989). Paralel mimariye sahip olan YSA, paralel çalışabilen işlemciler (FPGA) veya paralel mimariye uygun olarak geliştirilmiş devreler üzerinde gerçeklenirlerse seri çalışan işlenciler üzerinde gerçeklenen YSA'na göre çok daha hızlı çalışabilmektedir.

2.3.8 Analiz ve tasarım kolaylığı

YSA' nın, temel işlem elemanı olan hücrenin yapısı ve modeli, bütün YSA yapılarında yaklaşık aynıdır. Dolayısıyla, YSA'nın farklı uygulama alanlarındaki yapıları da standart yapıdaki bu hücrelerden oluşacaktır. Bu nedenle, farklı uygulama alanlarında kullanılan farklı mimarilerdeki YSA'lar, benzer öğrenme algoritmalarını ve teorilerini paylaşabilirler. Bu özellik, problemlerin YSA ile çözümünde önemli bir kolaylık getirecektir.

Sahip olduğu bu özellikler nedeniyle, günümüzde birçok alanda yapay sinir ağlarının uygulamalarına rastlamak olasıdır. Özellikle örüntü tanıma, işaret işleme, sistem tanıma ve nonlineer denetim alanlarında yapay sinir ağlarının değişik modelleri ve değişik öğrenme stratejileri başarı ile kullanılmıştır(Narendra and Parthasarathy 1990)

Ele alınan bir problemin YSA yaklaşımı ile çözüldüğünde tasarımcının önüne çeşitli secenekler çıkar. İlk secenek öğrenme mekanizması üzerindedir. Literatürde üç tip öğrenme stratejisinden bahsedilmektedir. Bunlar eğiticili öğrenme, eğiticisiz öğrenme ve pekiştirmeli (reinforcement) olarak isimlendirilmektedir (Haykin 1994, Chen 1996). Yaklaşımlar arasındaki temel farklılık istenilen çıkış değerlerinin mevcut olup olmamasıdır. Eğer bir eğitici , sistem çıkışlarının istenilen değerlerini

temin ediyorsa bu tip öğrenme birinci grupta yer alır. Tasarım koşulları istenilen değerlerin temin edilmesine müsaade etmiyorsa bu tip öğrenme ikinci grupta yer alır. Eğiticisiz öğrenme algoritmaları daha çok, sistemin geçmişte karşı karşıya kaldığı veri kümesinin içerdiği istatiksel bilgilerin çıkarılmasını amaçlar. Böylece çok elemanlı veri kümeleri içerisinde deneyim yoluyla bilgi genelleştirmesi yapılabilir. Örneğin bisiklet sürmeyi öğrenmeye çalışan bir çocuk önceki deneyimlerindeki yanlış hareketlerinin sonuçlarını gözleyerek bir sonraki denemede bu deneyimlerinden faydalanır.

Üçüncü tip (Pekiştirici öğrenmede) ise öğrenen sisteme bir eğitimci yardımcı olur. Fakat eğitimci her girdi seti için olması gereken çıktı setini sisteme göstermek yerine sistemin kendisine gösterilen girdilere karşılık çıktısını üretmesini bekler ve üretilen çıktının doğru veya yanlış olduğunu gösteren bir sinyal üretir (Öztemel 2003).

Tasarımda ikinci seçenek mimari üzerindedir ve iki alt başlıkta değerlendirilebilir. Bunlarda ilki veri akış yönüdür. Eğer ağ üzerinde bilgi akışı sürekli olarak ileri doğru ise bu yapıya sahip ağ modelleri ileri sürümlü olarak adlandırılır. Ağ yapısında geri besleme bağlantıları varsa bu tipteki sistemlere geribeslemeli denir (Teshnehlab and Wantanabe 1999).

Tasarımın sunduğu üçüncü önemli seçenek öğrenmedeki parametre güncelleme algoritmasıdır. Öğrenme algoritmaları öğrenme denen olguyu matematiğin kuralları ile ölçülebilir büyüklüklere dönüştürerek başarım ölçütünün oluşturulmasına ve bu ölçütün zaman içerisinde artılırmasını sağlayacak parametre değişikliklerinin hesaplanmasına dayanır. Burada parametre güncelleme işlemi için türetilen bilginin hangi yöntemelerle oluşturulduğu, tasarım esnekliğinin ana temasıdır.

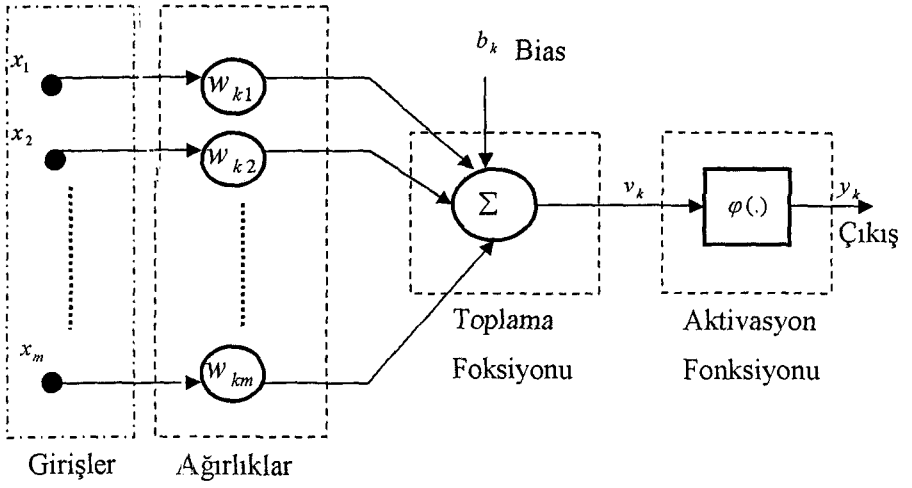
Bir başka seçenek ise parametre güncelleme işleminin zamanlamasıdır. Burada problemin dayattığı fiziksel gerçekliğin, zamanlamanın seçimi üzerinde bir koşul olabileceği vurgulanmalıdır. Eğiticili öğrenme yaklaşımında parametre güncelleme işlemi, normal çalışma esnasında, anlık gözlemlerden elde edilen bilgi ile yapılıyorsa buna eş zamanlı öğrenme denir. Eğer sinir ağı daha önceden belirlenen bir giriş/çıkış

eşleştirmesini gerçekleştirmeye çalışıyorsa buna zamandan bağımsız öğrenme denir (Piche 1994).

Son olarak parametre güncelleme işlemi için iki seceneğin varlığından bahsedilebilir. Eğer ağ parametreleri, eğitim çiftlerinin tamamının ağ üzerinden geçirilip her bir geçişte hesaplanan değişim miktarının toplamı ile güncelleniyorsa yığın (batch), her bir eğitim çifti için hesaplanan değişim miktarı o anda uygulanıyorsa ardışıl (sequential) güncellemeden bahsedilir.

2.4 Matemetiksel Yapay Sinir Hücresi Modeli

Yapay sinir ağını oluşturan yapay sinir hücreleri (nöronlar) tek başlarına ele alındıklarında çok basit işlevleri olan işlemcilerdir. Bu küçük işlemci birimini üç temel bölüme ayırabiliriz (Haykin 1999). Şekil 2.3' de bir yapay sinir hücresi gösterilmektedir. Diğer hücrelerden ya da dış ortamlardan hücreye giren bilgiler (yapay sinir girdileri) sinaptik bağlantılar üzerindeki ağırlıklarla çarpılarak bir birleştime fonksiyonuna uygulanmaktadır. Elde edilen toplam, yapay sinir hücrei aktivasyon fonksiyonundan geçirilerek çıkışlar elde edilir. Birleştirme fonksiyonu, bir hücreye gelen net girdiyi hesaplayan bir fonksiyondur ve genellikle net girdi, girişlerin ilgili ağırlıkla çarpımlarının toplamıdır. Birleştirme fonksiyonu, ağ yapısına göre maksimum alan, minimum alan, çarpım yada toplam fonksiyonu olabilir. Aktivasyon fonksiyonu ise birleştirme fonksiyonundan elde edilen net girdiyi bir işlemden geçirerek hücre çıktısını belirleyen ve genellikle doğrusal olmayan bir fonksiyondur.



Şekil 2.3. Bir yapay sinir hücresinin matematiksel modeli

Şekilden yola çıkılarak k. YSH için denklemler yazılırsa;

$$u_k = \sum_{j=1}^{m} w_{kj} x_j \tag{2.1}$$

$$v_k = u_k + b_k \tag{2.2}$$

$$y_k = \varphi(v_k) \tag{2.3}$$

Her bir girdideki değişim, YSH çıkışında bir değişime neden olmakta ve bu değişimin genliği, girişin etki derecesini belirleyen ağırlıga, toplayıcının eşik değerine ve aktivasyon fonksiyonuna bağlıdır. Yukarıdaki modeldende gözleneceği üzere eşik değeri girdilerden bağımsız olmasından dolayı bütün girişler sıfır olsa bile çıkışta bir $\varphi(0)$ değeri gözlenir.

2.4.1 Aktivasyon fonksiyonları

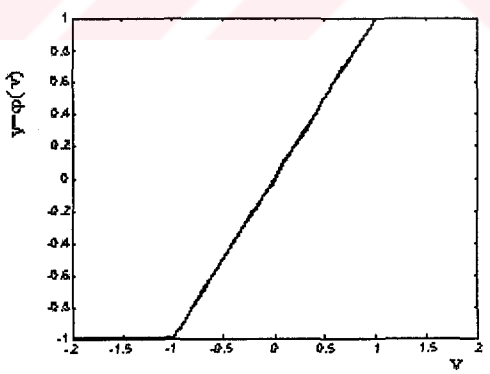
Hücre modellerinde, hücrenin gerçekleştireceği işleve göre çeşitli tipte aktivasyon fonksiyonları kullanılabilir. Aktivasyon fonksiyonları sabit parametreli yada

uyarlanabilir parametreli seçilebilir. Aşağıda, hücre modellerinde yaygın olarak kullanılan çeşitli aktivasyon fonksiyonları tanıtılmıştır.

2.4.1.1 Doğrusal ve doyumlu-doğrusal aktivasyon fonksiyonu

Doğrusal bir problemi çözmek amacıyla kullanılan doğrusal hücre ya da YSA'nın çıkış katmanında kullanılan doğrusal fonksiyon, hücrenin net girdisini doğrudan hücre çıkışı olarak verir (pureline). Doğrusal aktivasyon fonksiyonu matematiksel olarak y=v şeklinde tanımlanabilir. Doyumlu doğrusal aktivasyon fonksiyonu ise aktif çalışma bölgesinde doğrusaldır ve hücrenin net girdisinin belirli bir değerinden sonra hücre çıkışını doyuma götürür (limiter). Doyumlu doğrusal aktivasyon fonksiyonunun denklem 2.4'de matematiksel tanımı, Şekil 2.4'de ise grafiği görülmektedir (Haykin 1999).

$$y = \begin{cases} 1 & v > 1 \\ v & -1 < v < 1 \\ -1 & v < -1 \end{cases} \quad \text{ise} \qquad (2.4)$$



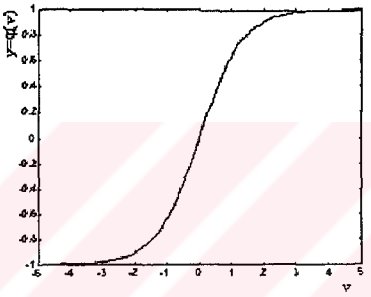
Şekil 2.4. Doyumlu doğrusal aktivasyon fonksiyonu

2.4.1.2 Sigmoid aktivasyon fonksiyonu

Şekil 2.5 de grafiği verilen çift yönlü sigmoid (tanh) fonksiyonu, türevi alınabilir, sürekli ve doğrusal olmayan bir fonksiyon olması nedeniyle doğrusal olmayan problemlerin çözümünde kullanılan YSA'larında tercih edilir. Çift yönlü sigmoid fonksiyonun tanımı denklem 2.5'de ve tek yönlü sigmoid (lojistik sigmoid) fonksiyonunun matematiksel ifadesi ise denklem 2.6'da verilmiştir.

$$\varphi(v) = a\frac{1-e^{-bv}}{1+e^{-bv}} \tag{2.5}$$

$$\varphi(v) = a\frac{1}{1+e^{-bv}} \tag{2.6}$$



Şekil 2.5. Sigmoid (tanh) aktivasyon fonksiyonu.

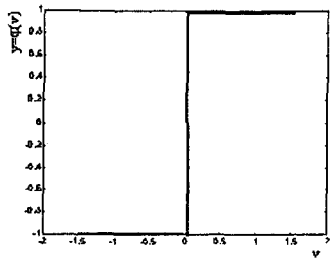
Sigmoid fonksiyonlarında a ve b katsayıları genellikle sabit olarak alınır ancak, YSA' nın eğitiminde öğrenme oranını hızlandırıcı etkilerinin olduğu belirlenmiştir. Ayrıca, a ve b katsayılarının YSA'nın eğitim sürecinde uyarlanmasıyla sabit katsayılı fonksiyona göre daha iyi bir performans elde edilebilmektedir. Bu uygulamada a ve b katsayıları 1 olarak kabul edilmiştir.

2.4.1.3 Eşik aktivasyon fonksiyonu

McCulloch-Pitts modeli olarak bilinen eşik aktivasyon fonksiyonlu hücreler, mantıksal çıkış verir ve sınıflandırıcı ağlarda tercih edilir, Şekil 2.6 Perceptron

(Algılayıcı) olarak adlandırılan, eşik fonksiyonlu (hardlimiter) hücrelerin matematiksel modeli aşağıdaki gibi tanımlanabilir (Haykin 1999).

$$y = \begin{cases} 1 & v \geq 0 \\ -1 & v < 0 \end{cases} \quad \text{ise} \tag{2.7}$$



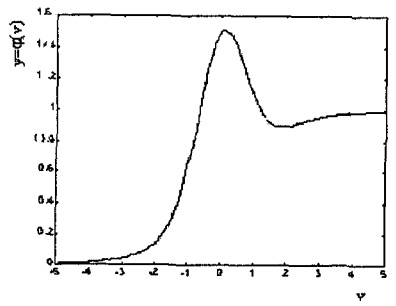
Şekil 2.6. Eşik aktivasyon fonksiyonu

2.4.1.4 Diğer aktivasyon fonksiyonları

Yukarıda verilen ve yaygın olarak kullanılan aktivasyon fonksiyonlarının dışında YSA' da çeşitli aktivasyon fonksiyonları kullanılmış ve aktivasyon fonksiyonlarına göre YSA'nın problemleri çözebilme performansları incelenmiştir. Denklem 2.7'da uyarlanabilir parametreli ve sigmoidal özellikli bir aktivasyon fonksiyonun matematiksel ifadesi ve Şekil 2.7'de ise grafiksel gösterimi verilmiştir. a ve b katsayıları, YSA' nın eğitim sürecinde uyarlanmak üzere Denklem 2.8 verilen aktivasyon fonksiyonuna sahip YSA'nın çeşitli problemlerin çözümünde etkin olduğu belirlenmiştir. Farklı problemlerde a1, b1 ve a2, b2 katsayıları farklı değerlere yakınsayarak YSA' nın eğitimini daha etkin kılmaktadır. Denklem 2.9'da normal sigmoid fonksiyonlarına göre YSA' nın eğitimini hızlandırıcı etkisi olduğu belirlenen karesel sigmoid aktivasyon fonksiyonunun matematiksel ifadesi verilmiştir. Denklem 2.10'de ise, a katsayısına bağlı olarak tanımlanan belirli bir aralıkta doğrusal ve bu aralıkların dışında sigmoidal değişen uyarlanabilir parametreli aktivasyon fonksiyonunun da çeşitli problemlerin YSA ile çözümünde etkin olduğu incelenmiştir. Uyarlanabilir a katsayısı 0-1 arasında değişmektedir (Efe ve Kaynak 1999).

$$y = \varphi(v) = a_1 e^{-b_1 v^2} + a_2 \frac{1}{1+e^{-b_2 v}} \tag{2.8}$$

$$y = \varphi(v) = \frac{1}{1+e^{-v}} \tag{2.9}$$



Şekil 2.7. Diğer aktivasyon fonksiyonları

$$y = \varphi(v) = \begin{cases} v & |v| < a \text{ ise} \\ (\pm)\left[(1-a)\tanh\left(\alpha + \frac{|v|-a}{1-a}\right)\right] & |v| > a \end{cases} \tag{2.10}$$

2.5 Yapay Sinir Ağı Yapıları

Yapay sinir ağları, hücrelerin birbirleri ile çeşitli şekillerde bağlanmalarından oluşur. Hücre çıkışları, ağırlıklar üzerinden diğer hücrelere ya da kendisine giriş olarak bağlanabilir ve bağlantılarda gecikme birimi de kullanılabilir. Hücrelerin bağlantı şekillerine, öğrenme kurallarına ve aktivasyon fonksiyonlarına göre çeşitli YSA yapıları geliştirilmiştir. Bu bölümde, çeşitli problemlerin çözümünde kullanılan ve kabul görmüş bazı YSA yapıları genel özellikleri ile tanıtılacaktır.

2.5.1 Yapay sinir ağı modelleri

İleri beslemeli sinir ağları,

- Çok Katmanlı Algılayıcı (Multi Layer Perceptron - MLP)
- Radyal Tabanlı Ağ (Radial-Basis Network - RBF)
- Kendini Ayarlayan Harita (Self-Organizing Map - SOM)

Geri beslemeli sinir ağları (recurrent),

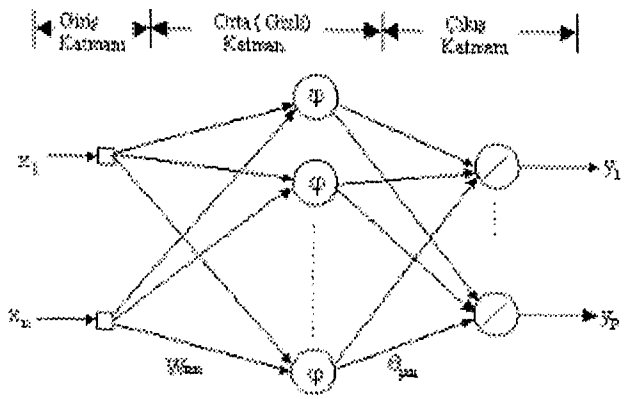
2.5.2 İleri beslemeli (Feed Forward) yapay sinir ağları (İBYSA)

Bir yapay sinir ağının öğrenmesi istenilen olayların girdi ve çıktıları arasındaki ilişkiler doğrusal olmayan ilişkiler olursa (örneğin XOR problemi) gibi tek katmanlı bir yapay sinir ağı bu probleme çözüm üretemez (Minsky and Papert 1969).

XOR problemini çözmek amacıyla yapılan çalışmalar sonucunda çok katmalı yapay sinir ağları geliştirilmiştir. Rumelhart ve arkadaşları tarafından geliştirilen bu modele hata yayma modeli veya geriye yayım modeli de denilmektedir (Rumelhart et al 1986).

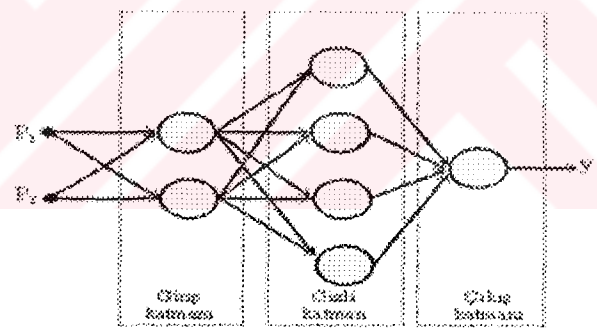
İleri beslemeli YSA' da, hücreler katmanlar şeklinde düzenlenir ve bir katmandaki hücrelerin çıkışları bir sonraki katmana ağırlıklar üzerinden giriş olarak verilir. Giriş katmanı, dış ortamlardan aldığı bilgileri hiçbir değişikliğe uğratmadan orta (gizli-hidden) katmandaki hücrelere iletir. Ağa sunulan giriş, orta ve çıkış katmanında işlenerek ağ çıkışı belirlenir. Bu yapısı ile ileri beslemeli ağlar doğrusal olmayan statik bir işlevi gerçekleştirir. İleri beslemeli 3 katmanlı YSA' nın, orta katmanında yeterli sayıda hücre olmak kaydıyla, herhangi bir sürekli fonksiyona istenilen doğrulukta yakınsanabileceği gösterilmiştir. En çok bilinen geriye yayma öğrenme (Backpropagation) algoritması, bu tip YSA ların eğitiminde etkin olarak kullanılmakta ve bazen bu ağlara geriye yayma ağları da denmektedir. Şekil 2.8 de

giriş, orta ve çıkış katmanı olmak üzere 3 katmanlı ileri beslemeli YSA yapısı verilmiştir



Şekil 2.8. İleri beslemeli 3 katmanlı YSA.

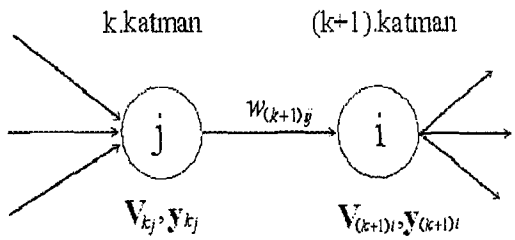
2.5.2.1 Çok Katmanlı Algılayıcı (Multi Layer Perceptron)



Şekil 2.9. 2-4-1 düzenine sahip bir MLP

Şekil 2.9' da gösterilen ağ yapısı, geri besleme bağlantıları olmaması dolayısıyla, veri akış yönünden ileri beslemeli ağ yapısındandır. Bu yapıda giriş katmanı giriş vektörünü gizli katman ulaştırmakla yükümlüdür ve lineer olmayan bir davranışa sahip değildir.

Ağ üzerindeki katmanlar k indisiyle sıralansın ve L adet gizli katman olduğu varsayılsın. Yani Şekil 2.9'daki ağ için L=1 olur, k=0 giriş katmanına, k=1 gizli katmana ve k=L+1 çıkış katmanına tekabul eder. Eğer k+1'inci katmanın i'inci yapay sinir hücresini k'ıncı katmanın j'inci sinir hücresine bağlayan bağlantının ağırlık değeri $w_{(k+1)ij}$ ile gösterilirse, ve k'ıncı katmanın j'inci yapay sinir hücresi çıkışı y_{kj} ile gösterilirse, k+1'inci katmandaki i'inci sinir hücresini net toplamı $v_{(k+1)j}$ ve çıkış değeri $y_{(k+1)j}$ aşağıdaki şekilde hesaplanır. Deklemlerde görülen n_k değişkeni, k'ıncı katmandaki yapay hücre sayısını simgelemektedir.



Şekil 2.10. Ağ içerisinde katmanların ve yapay sinir hücrelerinin sıralaması

$$v_{(k+1)i} = \sum_{j=1}^{n_k} w_{(k+1)ij} y_{kj} \tag{2.11}$$

$$y_{(k+1)i} = \varphi(v_{(k+1)}) \tag{2.12}$$

Hatanın geriye yayılımı yönteminin kullanılabilmesi için aktivasyon fonksiyonun türevlenebilir olması gerekmektedir.

2.6 Hata Geriye Yayma Yöntemi ile Parametre Güncelleme

Ses tanıma problemlerinden (Moon and Hwang 1997) nonlineer sistem tanıma ve denetim problerine kadar (Efe ve Kaynak 1999) yapay sinir ağları ile çözüm üreten bir çok alanda başarı ile kullanılan bu yöntem, kuadratik bir maliyet fonksiyonunun zaman içerisinde, ağ parametrelerinin uyarlanması ile minimizasyonuna dayanmaktadır.

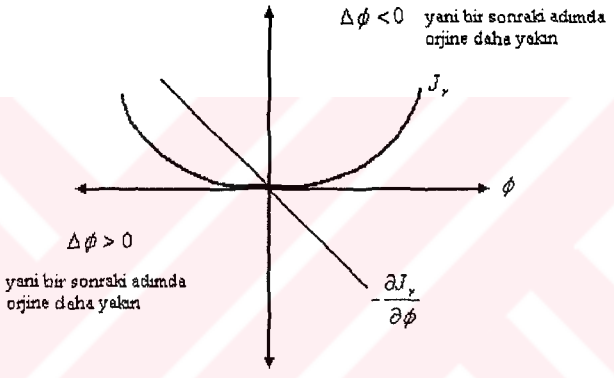
2.6.1 Yöntemin dayandığı metedoloji

Hata geriye yayılma yönteminin temel felsefesi Denklem 2.13 ile verilen tek paremetreli (ϕ) bir maliyet fonksiyonunun (J_r) en küçük değeri aldığı noktanın Denklem 2.14 ile verilen kural ile iteratif olarak bulunabilmesine dayalıdır.

$$J_r = \frac{1}{2}\phi^2 \tag{2.13}$$

$$\Delta\phi = -\eta \frac{\partial J_r}{\partial \phi} \tag{2.14}$$

$$\phi(n+1) = \phi(n) + \Delta\phi \tag{2.15}$$



Şekil 2.11. Eğim düşümü yönteminin grafiksel yorumu

Şekil 2.11'de parmetrenin negatif olduğu bölgede sağa, pozitif olduğu bölgede sola doğru bir hareket söz konusudur. Burada önem kazanan bir nokta, η değişkennin değeridir. Ptatikte öğrenme katsayısı veya adım büyüklüğü olarak bilinen bu değer çok küçük ise, hata uzunca bir süre içerisinde orjine doğru yakınsarken, büyük bir değer orjin etrafında salınımlara hatta ıraksamaya neden olabilir. Katman ve nöron sayısı gibi adım büyüklüğünün seçimi de bir çok uygulamada deneme yanılma ile yapılır.

2.6.2 Matematiksel türetim ve analiz

Hata geriye yayılma yönteminin türetimi için Denklem 2.16 ile verilen maliyet fonksiyonu minimize edilmelidir. Bu amaçla, bir önceki bölümde değinildiği gibi Denklem 2.17 ile verilen parametre güncelleme fonksiyonu belirlenecektir. Denklem 2.17'de ∇_w sembolü w parametresine göre kismi türevi göstermektedir.

Yöntemin türetimi, çıkış katmanındaki nöronların parametreleri için farklı, gizli katmandaki nöronların parametreleri için farklı bir formülasyon ortaya çıkarır. Bu nedenle bu iki durum birbirinden farklı olarak ele alınmıştır. Türetim için Şekil 2.12 ile verilen çıkış katmanı göz önüne alınsın. Bu katmanın k+2'inci katman olduğu ve $n_{(k+2)}$ sayıda nöron içerdiği varsayılacak ve türetim esnasında aşağıdaki değişkenler kullanılacaktır.

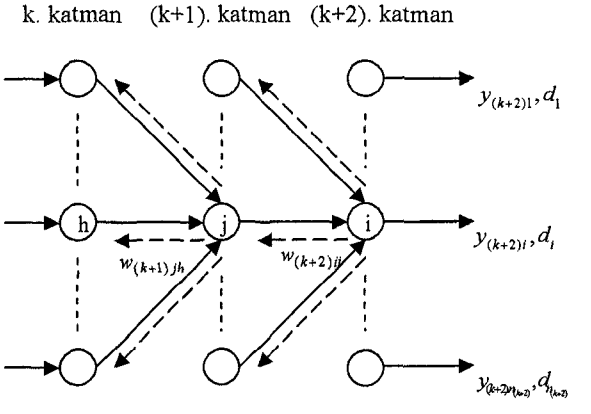
J_r : Maliyet fonksiyonu

d_i : Ağın i'inci çıkışı için istenen çıkış değeri

$y_{(k+2)i}$: k+2'inci katmanın i'inci yapay sinir hücresi çıkışında gözlenen değer

$w_{(k+2)ij}$: k+2'inci katmandaki i'inci sinir hücresi ile (k+1)'inci katmandaki j'inci sinir hücresini birleştiren ağırlık

$v_{(k+2)i}$: k+2'inci katmanın i'inci sinir hücresi girişinde oluşan net toplam



Şekil 2.12. Çıkış katmanında hatanın geriye yayılması

Ağın gi rişine uygulanan veriler ileri yayımlanarak ağın o andaki çıkışı üretilir. Maliyet fonksiyonu ise hedef ile ağın çıkışı arasındaki fark kullanılarak üretilir.

$$J_r = \frac{1}{2}\sum_{i=1}^{n}(d_i - v_i)^2 \tag{2.16}$$

Amaç Denklem 2.16'daki maliyet fonksiyonunu minimize etmektir.Bu maliyet fonksiyonu ise ağırlıkları (w) değiştirecektir. Bu sebeple maliyet fonksiyonunun ağırlıklara göre kısmi türevi alınmalıdır.

$$\Delta w = -\eta \nabla_w J_r \tag{2.17}$$

Daha sonra çarpanlar sırasıyla açılırsa:

$$\frac{\partial J_r}{\partial w_{(k+2)ij}} = \frac{\partial J_r}{\partial y_{(k+2)i}} \frac{\partial y_{(k+2)i}}{\partial v_{(k+2)i}} \frac{\partial v_{(k+2)i}}{\partial w_{(k+2)ij}} \tag{2.18}$$

$$\frac{\partial J_r}{\partial y_{(k+2)i}} = -(d_i - v_{(k+2)i}) \tag{2.19}$$

$$\frac{\partial y_{(k+2)i}}{\partial v_{(k+2)i}} = \frac{d\varphi(v_{(k+2)i})}{dv_{(k+2)i}} = \varphi'(v_{(k+2)i}) \tag{2.21}$$

$$\frac{\partial v_{(k+2)i}}{\partial w_{(k+2)ij}} = \frac{\partial}{\partial w_{(k+2)ij}}\left(\sum_{j=1}^{n_k} w_{(k+2)ij} y_{(k+1)j}\right) = y_{(k+1)j} \tag{2.22}$$

Eğer Deklem 2.23 da verilen kısmi türeve delta(δ) adını verirsek, çıkış katmanındaki yapay sinir hücresi için delta değerlerinin genel hali Denklem 2.24'de, parametrelerdeki değişim oranı da Denklem 2.25 de gösterilmiştir.

$$\delta_{(k+2)i} = -\frac{\partial J_r}{\partial v_{(k+2)i}} \tag{2.23}$$

$$\delta_{(k+2)i} = (d_i - y_{(k+2)i})\varphi'(v_{(k+2)i}) \tag{2.24}$$

$$\Delta w_{(k+1)ij} = \eta \delta_{(k+2)i} y_{(k+1)j} \tag{2.25}$$

Çıkış katmanındaki her bir ağırlığı yenilemek için ise 2.26 formülü kullanılabilir.

$$w_{(k+2)ij}(n+1) = w_{(k+2)ij}(n+1) + \Delta w_{(k+2)ij}(n) \tag{2.26}$$

Gizli katman hücrelerinin parametrelerini güncellemek için ise denklem 2.18'i tekrar yazalım.

$$\frac{\partial J_r}{\partial w_{(k+2)ij}} = \frac{\partial J_r}{\partial y_{(k+2)i}} \frac{\partial y_{(k+2)i}}{\partial v_{(k+2)i}} \frac{\partial v_{(k+2)i}}{\partial w_{(k+2)ij}} \tag{2.27}$$

Denklem 2.27 deki kısmi türevi oluşturan terimler Şekil 2.12 de görüldüğü gibi bir çok yollardan gelebilirler. Bu durum Denklem 2.27'indeki zincir kuralının ilk teriminin açık hali olan Denklem 2.28'de de görülmektedir. Aynı terimi biraz daha açarsak Denklem 2.29 ve Denklem 2.30'u elde ederiz.

$$\frac{\partial J_r}{\partial y_{(k+1)j}} = \sum_{i=1}^{n_{(k+2)}} \frac{\partial J_r}{\partial v_{(k+2)i}} \frac{\partial v_{(k+2)i}}{\partial y_{(k+1)j}} \tag{2.28}$$

$$\frac{\partial J_r}{\partial y_{(k+1)j}} = \sum_{i=1}^{n_{(k+2)}} \left(\frac{\partial J_r}{\partial v_{(k+2)i}} \frac{\partial}{\partial y_{(k+1)j}} \left(\sum_{j=1}^{n_{(k+2)}} w_{(k+2)ij} y_{(k+1)j} \right) \right) \tag{2.29}$$

$$\frac{\partial J_r}{\partial y_{(k+1)j}} = \sum_{i=1}^{n_{(k+2)}} \frac{\partial J_r}{\partial v_{(k+2)i}} w_{(k+2)ij} \tag{2.30}$$

Çıkış katmanlarında olduğu gibi gizli katmanlarda da delta(δ) değeri tanımlanabilir:

$$\delta_{(k+2)i} = \frac{\partial J_r}{\partial v_{(k+2)i}} \tag{2.31}$$

Bu denklemin tanımlanması ile 2.27 denkleminin ilk terimi 2.32 denklemindeki şekilde yazılabilir. 2.33 ve 2.34 denklemleri ise 2.27'daki ikinci ve üçüncü terimlerin daha açık ifade edilmeleridir. Bu üç terim birleştirilirse 2.35 denklemi türetilir ve 2.35'deki parametre güncelleme kuralına erişilir.

$$\frac{\partial J_r}{\partial y_{(k+1)j}} = -\sum_{i=1}^{n_{(k+2)}} \delta_{(k+2)i} w_{(k+2)ij} \tag{2.32}$$

$$\frac{\partial y_{(k+1)j}}{\partial v_{(k+1)j}} = \frac{\partial \varphi(v_{(k+1)j})}{\partial v_{(k+1)j}} = \varphi'(v_{(k+1)j}) \tag{2.33}$$

$$\frac{\partial v_{(k+1)j}}{\partial w_{(k+1)jh}} == y_{kh} \tag{2.34}$$

$$\delta_{(k+1)j} = -(\sum_{i=1}^{n(k+2)} \delta_{(k+2)i} w_{(k+2)ij}) \varphi'(v_{(k+1)i}) \tag{2.35}$$

$$\Delta w_{(k+1)jh} = \eta \delta_{(k+1)j} y_{kh} \tag{2.36}$$

BÖLÜM 3. PROGRAMLANABİLİR LOJİK ELEMANLARIN MİMARİSİ VE PROGRAMLAMA TEKNİKLERİ

Sayısal işlem sistemleri, sayısal verileri işlemek için tasarlanmış, hızlı donanımlara ve bu donanımlara işlevsellik kazandıracak esnek yazılımlara ihtiyac duyan yapılardır. Sayısal işlem sistemi oluşturmada, donanım ve yazılım tabanlı olmak üzere iki geleneksek yöntem uygulanır.

Donanım tabanlı yöntem : Sayısal verileri işlemek için ağırlıklı olarak özel tüm devreler (ASIC) kullanılır. Bu tip devreler özel bir fonksiyonu gerçekleştirmek amacıyla üretildiğinden, bu fonksiyonları etkin ve hızlı bir şekilde gerçeklerler. Ancak işlevleri sınırlıdır ve sadece ilgili oldukları uygulamaya yönelik üretilmişlerdir. ASIC devreler doğru ve hızlı sonuç vermesine rağmen çözüm ürettiği problemin çeşitli türevleri için kullanılamayacaklardır. Yeni problemler için yeni donanımlara ve yeni ASIC yapılara ihtiyac duyulur. Bu da maliyet artışına ve zaman kaybına neden olur(Villasenor ve William 1997).

Yazılım tabanlı yöntem : Bu yöntemde tasarım değişikliklerine daha esnek olan mikroişlemciler kullanılır. Mikroişlemcinin koşturduğu yazılımlar değiştirilerek, hiçbir donanım değişikliğine gidilmeden tasarım ortamına yeni fonksiyonlar eklenebilir. Mikroişlemciler üzerinde koşan, yazılım uygulamaları aynı anda birçok işlemi yerine getirmek için tekbir işlemcinin genel kaynaklarını kullanırken, yavaş fakat esnek yazılımlar ile çalışırlar. Fakat sıradan bir uygulama için komutların bellekten okunması, onların yorumlanıp yerine getirilmesi, sistemin performansı ve hızını oldukça düşürmektedir.

Günümüde sayısal işlem sisitemlerinin kullandığı alandaki hızlı gelişmeler, üretim tamamalandıktan sonra da esnek genel amaçlı olacak şekilde tasarlanan işlem sisitemlerini ortaya çıkarmıştır. Özellikle iyi tasarlanmış ve işlemci yükünü azaltan, paylaşan donanımlar, performans artışı için iyi bir çözüm olabilir. Bununla beraber,

işlevleri uygulama sırasında değiştirilebilen programlanabilir devre elemanlarının kullanımı da avantaj getirecektir. Bu devre elemanları ile gerçekleştirilen donanımlar, ASIC gibi devre elemanları ile yapılan klasik donanımlara göre daha işlevseldirler(Hutchings ve Wirtin 1995).

Tekrar düzenlenebilen işlem sistemleri (Reconfigurable Computing System) olarak ta adlandırılan bu sistemler, esnek ve genel amaçlı yapıları sayesinde yeni bir üretim aşamasına ihtiyaç duymadan, değişen protokollere, sistem özelliklerine ve kullnıcı ihtiyaçlarına kısa sürede cevap verebilirler. Yine bu özellikleri sayesinde tekrar düzenlenebilen işlem sistemleri, donanım tabanlı sayısal işlem sistemlerine göre daha esnek ve yazılım tabanlı sayısal işlem sistemlerine göre daha hızlı sayısal tasarım ortamı oluşturarak, bu iki sistem arasındaki boşluğu doldururlar (Cmpton and Haouck 2002).

Tekrar düzenelebilir sayısal işlem sistemlerinin, ihtiyaç duyduğu esnek donanımlar, Alan Programlamalı Kapı Dizileri(Filed Programable Gate Array, FPGA) kullanılarak karşılanır (Villasenor ve William 1997). Özellikle SRAM (Statik Rastgele Erişimli Bellek) tabanlı FPGA'ler tekrar düzenlenebilirlik kabiliyetleri ve yüksek performanslı uygulamalardaki yeterlilikleri sayesinde, genel amaçlı sayısal donanımların tasarımında anahtar rol oynarlar.

Yakın tarihten itibaren sayısal işlem sistemleri üzerinde FPGA'ler kadar etkili olan başka bir kavram ise HDL'dir (Donanım tanımlama dili). Donanım tanımlama dillerinin kullanılması system-on-chip (SoC) teknolojisini beraberinde getirmiştir. Sayısal işlem sistemleri tasarımında system-on-chip (SoC) teknolojisi özellikle yer ve enerji sorunlarının yoğun olarak yaşandığı alanlarda kullanılmaktadır.

SoC teknolojisinde, sistem içerisindeki birimler çoğunlukla bir donanım tanımlama dili yardımıyla ifade edilirler. Sistemin tanımlama aşamasını takiben derleme ve davranışsal benzetim adımları gerçekleştirilir. Sistemden beklenen cevapların elde edilmesiyle sistem üzerinde zamansal benzetim aşamasına geçilir. Bütün birimler, sentezleme ve yerleştirme işlemi sonunda tekrar programlanabilir bir tümleşik

devreye aktarılır. Sistemin uygulama aşamasında ise gerek büyük kapasiteleri gerekse de esnek yapılarından dolayı FPGA tümleşik devreleri tercih edilmektedir.

Günümüz teknolojisinde FPGA ve VHDL (Very High Speed Integrated Circuit HDL) gittikçe önem kazanmaya başlamışlardır. VHDL ile SoC teknolojisi kullanılmasıyla tanımlanan ve sentezlenen sistemler FPGA ile gerçeklenerek gerçek gücü ortaya koymaktadırlar.

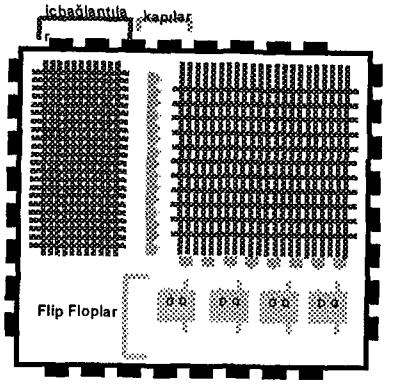
3.1 Programlanabilir Lojik Elemanların Gelişimi

Sayısal tasarımlarda kullanılan ilk programlanabilir lojik devre elemanı salt okunabilir bellek(PROM) elemanıdır. Bu elemanda, belleğin adres girişleri, lojik devrenin girişlerine, adreslenmiş gözdeki bilgiler de, lojik fonksiyonun çıkışlarına karşılık düşer. Genellikle karmaşık lojik fonksiyonlar için verimsizdirler.

Programlanabilir yapıların sonraki türleri PLD'lerdir (Programlanabilir Lojik Devre). PLD'ler; SPLD(Basit Programlanabilir lojik eleman), CPLD(Karmaşık Programlanabilir lojik eleman),MPGA(Maske programlanabilir kapı dizileri), FPGA(Alan Programlamalı kapı dizileri) olmak üzeri beş bölümde incelenebilir.

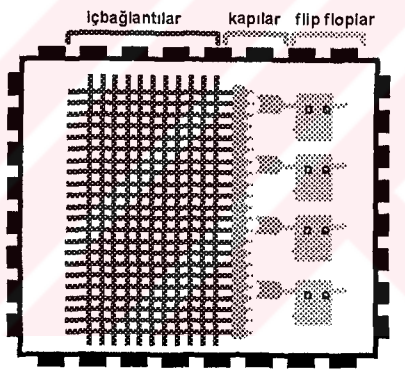
SPLD'ler PLA(Programlanabilir Lojik Dizi) ve PAL (Programlanabilir Dizi Lojiği) olmak üzere iki kısma ayrılırlar.

PLA : PLA mimarisi Şekil 3.1 de görüleceği gibi iki tane programalanabilir düzleme sahiptir. Bu iki programlanabilir düzlem AND (VE) ve OR (VEYA) kapılarının kombinasyonlarıyla oluşmakta ve AND işlemini birçok OR kapısı üzerinde paylaştırma esasına dayanmaktadır. Bu mimari oldukça esnektir fakat iki tane programalanabilir düzleme sahip olması üretim ve yollanma (maping) gecikmelerini artırmaktadır (Bkz.Şekil 3.1).



Şekil 3.1. PLA lojik devre elemanın iç mimarisi

PAL : Pal mimarisinde ise sadece bir tane programlanabilir düzlem bulunmakadır. Bu mimaride programlanabilir AND(VE) ve sabit OR(VEYA) matrisleri bulunmaktadır. PAL'ler bu özellikleri ile ucuz ve yüksek hızlı performans sağlamaktadırlar (Bkz.Şekil 3.2).

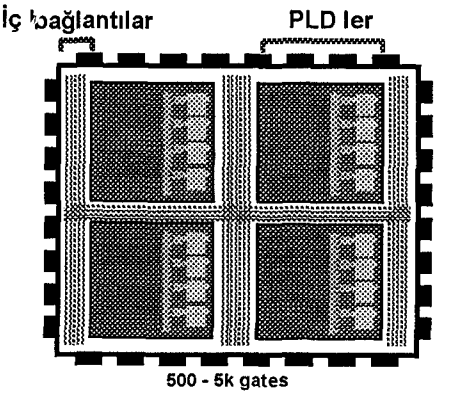


Şekil 3.2. PAL lojik devre elemanın iç mimarisi

PLA ve PAL' ler Basit programlanabilir lojik devreler (SPLD) olarakta adlandırılırlar.

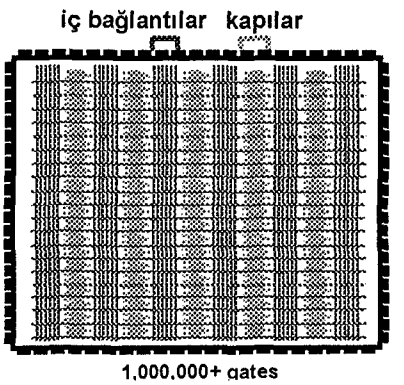
SPLD'lerin sınırlı kapasiteleri daha yüksek kapasiteli programlanabilir devreler olan CPLD'lerin doğmasına neden olmuştur. CPLD'ler, SPLD benzeri birçok bloğun bir araya getirilmesiyle oluşmuş yapılardır.

CPLD sadece bireysel PLD'lerin aynı çip üzerinde toplanmış ve bu bireysel PLD'lerin birbirlerine bağlanmak için arabağlantı yapılarının çip üzerinde düzenlenmiş halidir (Bkz. Şekil 3.3).



Şekil 3.3. CPLD loj ik devre elemanın iç mimarisi

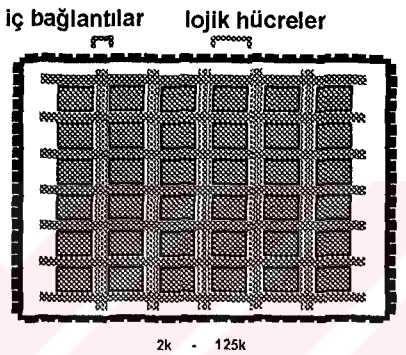
Programlanailir devre elemanlarını gelişim sürecinde CPLD'lerden sonra Maske programlamalı kapı dizileri (Mask Programable Gate Array, MPGA) tanıtılmıştır. MPGA, kullanıcıların lojik devresindeki özelliklere ve bağlantılara göre özel olarak üretilmiş tranzistor dizilerinden oluşur. Kullanıcının isteğine göre üretildiğinden dolayı fabrikasyon süreci hem uzun hemde masraflıdır. MPGA ler tam programlanabilir lojik tümdeverler olmasalar da programlanabilir türevleri olan FPGA lerin gelişmesinde ilk aşama olmuşlardır (Bkz. Şekil 3.4).



Şekil 3.4. MPGA lojik devere elemanın iç mimarisi

3.2 Alan Programlamalı Kapı Dizileri (FPGA)

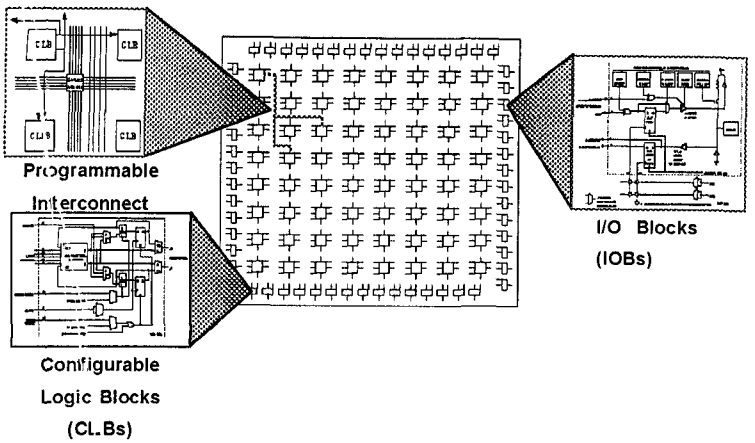
CPLD ve MPGA'ların üstünlüklerini kullanmak ve sakıncalarını ortadan kaldırmak amacıyla Xilinx firması, 1985 yılında, FPGA'leri piyasaya sunmuştur(Venkatesan 1994, Willert 200). FPGA'lar programlanabilir lojik bloklar ve ara bağlantılardan oluşur. Kullanıcının tasarladığı devreye göre, FPGA üreticisi tarafından sağlanan bir yazılım sayesinde lojik bloklar ve aralarındaki bağlantılar programlanır. Tasarım sırasında kullanıcıya sağladığı esneklik, düşük maliyet ve hızlı örnek üretme özelliği FPGA'ları sayısal tasarım ortamlarının vazgeçilmezi haline getirmiştir (Bkz. Şekil 3.5).



Şekil 3.5. FPGA mimarisi

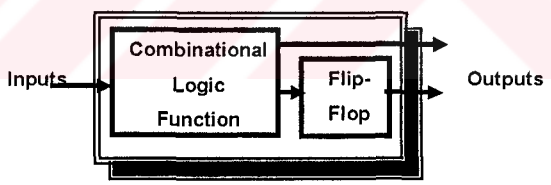
3.2.1 FPGA'lerin mimarisi

FPGA'lar temelde üç bloktan oluşur, lojik bloklar(CLB), giriş-çıkış blokları(I/O) ve bağlantı blokları.



Şekil 3.6. FPGA'i oluşturan bloklar

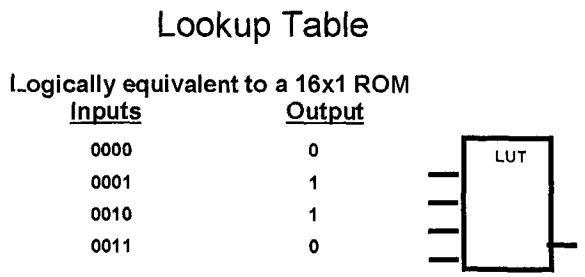
Lojik Bloklar: Şekil 3.6 ile gösterilen lojik bloklar(CLB) boolen fonksiyonlarının gerçeklendiği yapılardır; küçük taneli (fine-grain) ve kaba taneli (coarse-grain) olarak adlandırılan iki sınıfa ayrılırlar. Bu sınıfandırmada; CLB'nin oluşumunda kullanılan tranzistor sayısı, lojik bloğun gerçekleyebileceği boolen fonksiyon sayısı veya lojik bloğun giriş-çıkış sayısı büyüklük ölçütü olarak kullanılabilir (Hartenstein et al 2000).



Şekil 3.7. CLB iç yapısı

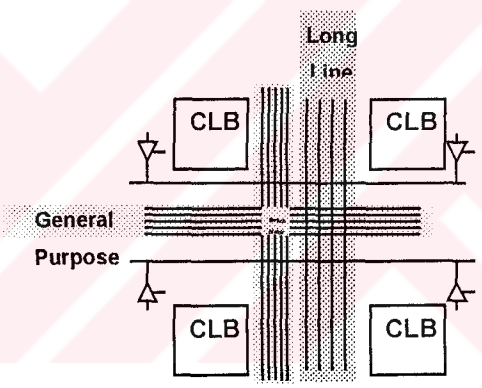
Şekil 3.7'de iç yapısı verilen CLB, küçük taneli bloklar, genellikle iki girişli bir lojik kapıya veya bir kaç girişli bir çoklayıcıya eşlik eden saklama elemanından oluşur.

Kaba taneli lojik blokların yapıları çok çeşitlilik göstermelerine karşın yaygın olarak Şekil 3.8 ile gösterilen doğruluk tablosu (LUT) veya çoklayıcı (Multiplexer) gibi daha büyük saklama elemanlarından oluşur.



Şekil 3.8. CLB'yi oluşturan LUT elemanı

Bağlantı Blokları : Şekil 3.9 da verilen bağlantı blokları lojik bloklarla giriş-çıkış blokları arasındaki bağlantıyı sağlayan yapılardır. Bu yapılar, yollandırma kanalları ve programlanabilir anahtarlardan oluşur.



Şekil 3.9. bağlantı blokları

FPGA mimarileri bağlantı kanallarının yapısına göre; Simetrik Dizi Mimarisi (Symmetrical Array), Sıra Tabanlı Mimari (Row-Based Array) ve Kapı Denizi Mimarisi (Sea of Gate) olmak üzere üç ana gruba ayrılır (Brown 1992)

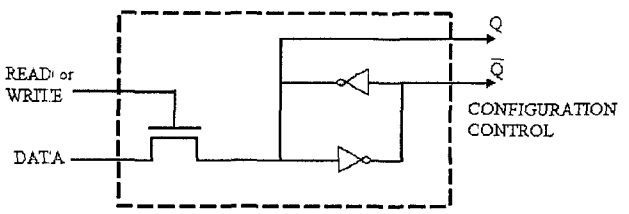
3.2.2 FPGA'lerin programlama teknolojileri

FPGA'ler için üç tip programlama teknolojisi genel olarak kullanılır. Bu üçü de elemanın mimarisinde yansıtılan ilgili alan ve performans etmenlerine bağlı

kullanılır. Buna göre FPGA'ler eleman mimarisi ve programlama konfigürasyonuna göre kategorize edile bilir.

3.2.2.1 Statik RAM programlama teknolojisi

Bir SRAM programlı FPGA'de, normal işlem boyunca program statik hafıza hücresinde tutulur. Şekil 3.10 Hafıza statik RAM hücrelerinden yapılmıştır ve bunun için çip sık sık "SRAM programlanabilir" olarak ifade edilir.

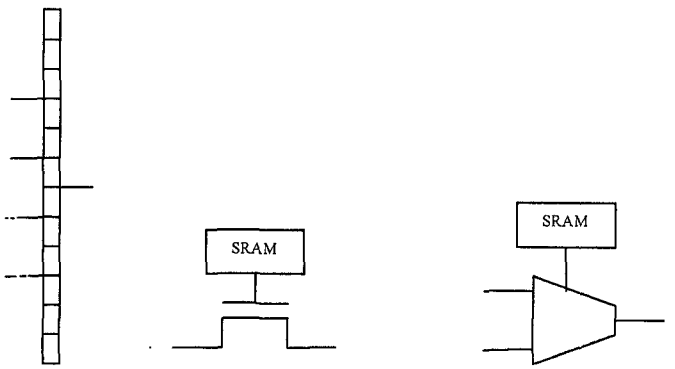


Şekil 3.10. Bir Statik RAM hafıza hücresi

Tümleşik devre üzerinde SRAM hafıza hücreleri için ayrılmış yer yoktur, SRAM hücreleri kontrol ettikleri lojik elemanlar arasında dağıtılmışlardır. SRAM hücreleri ile üç ayrı inşa bloğu kontrol edilir. Look-up tablosu, adres hatlarını kontrol eden fonksiyon girişleriyle hafıza hücrelerinden yapılmış önemli inşa bloklarından biridir. Look-up tablosu Şekil 3.11(a)'da gösterilmiştir. Diğer inşa bloğu programlanabilir ara bağlantı noktası (PIP) diye isimlendirilir ve Şekil 3.11(b)'deki gibidir. PIP bir hafıza hücresinin kontrol ettiği bir geçiş hücresidir. Konfigüre edilebilir bir ara bağlantı olarak kullanılır. Üçüncü inşa bloğu ise Şekil 3.11(c)'de gösterilen bir hafıza hücresi tarafından kontrol edilen multiplexer'dır.

SRAM programlama teknolojisi mimarisinin en büyük dezavantajı geniş alan kaplamasıdır. Diğer dezavantajı ise gücünün kesilmesiyle barındırdığı bilgilerin silinmesidir.

Ancak, SRAM parogramlama teknolojisi iki önemli avantaja sahiptir; sadece standart tümleşik devre teknolojisinin sağladığı hızlı tekrar-programlanabilirlik ve hatta karmaşık lojik devreler için bile düşük güç tüketimidir.



Şekil 3.11. (a)LUT, (b)PIP, (c)Multiplexer

SRAM tabanlı FPGA'ler, tekrar düzenlenebilirlik kabiliyetleri ve yüksek performanslı uygulamadaki yeterlilikleri nedeniyle sayısal işaret işlemede temel devre elemanı olarak yaygın şekilde kullanılmaktadırlar (Hadley and Hutchings 1995). Ayrıca tasarım tamamlanır tamamlanmaz test etmek mümkün olduğundan uygulamalarda oldukça avantaj sağlamaktadırlar (Goslin 1995).

3.2.2.2 Antisigorta-tabanlı programlama teknolojisi

Bu teknolojide, FPGA programlanmadan önce, yollanma kanalları arasındaki bağlantılar kurulmamış durumdadır. Programlama sırasında uygulanan gerilimle gerekli bağlantılar oluşturulmuş olur. Böylece kullanılan bağlantı sayısının kullanılmayan bağlantı sayısından daha az olması sebebiyle programlama süresi kısalır (Trimberger 1994).

Programlama için gerekli gerilimin entegre içine dağıtılmasını sağlayan tranzistorlar yarıiletken yonga üzerinde geniş alan kaplamasına karşın, diğer teknolojilerle karşılaştırıldıklarında antisigorta'lar daha küçük bir alana ihtiyaç duyarlar. Bu

teknolojiyi kullanan FPGA'ler bir kez programlanabilme özelliğine sahip olduklarından ilk örnek üretim için pahalı bir çözüm olmaktadırlar.

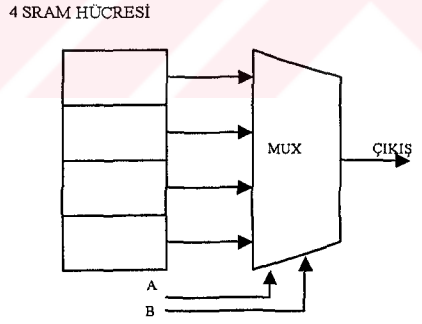
3.2.2.3 EPROM ve EEPROM programlama teknolojisi

EPROM ve EEPROM programlamada kullanılan yapı EROM belleklerde kullanılan yapıya benzemektedir. Bu teknolojiyi kullanan FPGA'ler tekrar programlanabilir. Ancak, bu tip yapıları programlamak için özel devreciklere ihtiyac duyulur bu sebeple devre üzerinde programlanamazlar.

3.2.3 FPGA'ların lojik hücre mimarisi

3.2.3.1 Doğruluk tablosu tabanlı yapı

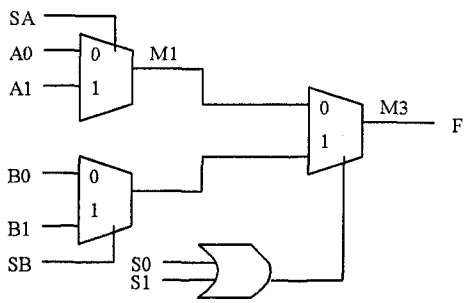
Bu yapının temel bloğu, LUT(Look Up Table) adı verilen ve m(m≥2) değişkenli her boolen foksiyonunu gerçekleyebilen bir yapıdır (Bkz.Şekil 3.12).



Şekil 3.12. İki girişli LUT yapısı

3.2.3.2 Çoklayıcı tabanlı yapı

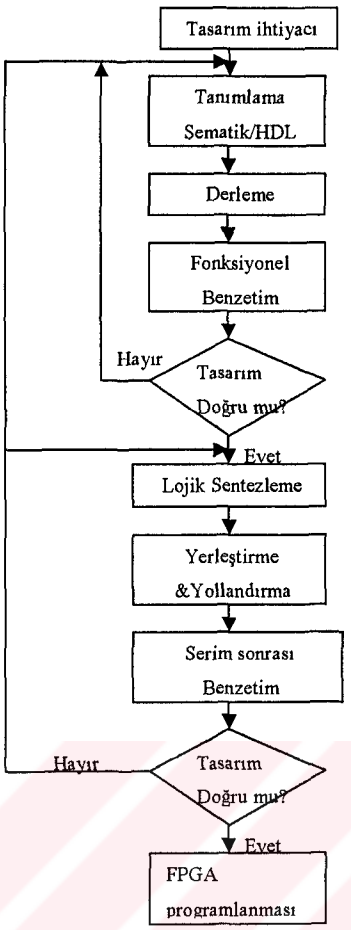
Bu yapının temel bloğu çeşitli konfigürasyonlardan ve olabildiğince az VE ve VEYA gibi lojik kapılardan oluşur. Şekil 3.13



Şekil 3.13. Çoklayıcı tabanlı lojik hücre yapısı

3.3 FPGA Kullanılarak Gerçekleştirilen Devrelerin Tasarım Süreci

Tasarım süreci, gerçeklenecek devre fonksiyonlarının, sözle veya şematik olarak tanımlanması ile başlar. Sözde tasarımda genellikle yüksek seviyeli donanım tasarım dilleri (Hardware Description Language, HDL) kullanılır. Şematik tanımlamada ise bir çok firma tarafından geliştirilmiş şematik editör programlarından faydalanılır. Tanımlama ne şekilde olursa olsun, derleme işlemi sonrasında, tüm tanımlamalar, standart bağlantı listesi (netlist) biçimine çevrilir (Nur 2000). Yapılan tanımlamaların, istenilen fonksiyonları yerine getirip getirmediği fonksiyonel benzetim (functional simulation) yapılarak test edilir. Benzetim sonucuna göre, tanımlamada gerekli değişiklikler yapılır. Şekil 3.14'de tasarım sürecinin akış şeması verilmiştir.



Şekil 3.14. FPGA kullanılarak yapılan tasarım sürecinin akış diyagramı

3.4 VHDL Donanım Tasarım Dili

3.4.1 Giriş

Lojik devre tasarımının başlangıç dönemlerinde, tasarım lojik kapılar seviyesinde yapıldı. Ancak tasarımların karmaşıklaşması ve bunların lojik kapılar kullanılarak gerçeklenme zorluğu nedeniyle çeşitli yazılım dilleri tasarlanmaya başlandı.Bu dillere genel olarak HDL (Hardware Description Languages) adı verilir. Bu dillerden en çok kullanılanı VHDL (Very High Speed Application Specific Integrated Circuit Hardware Description Language) yazılım dilidir. VHDL yazılım dili kullanılarak, her

türlü lojik devre tasarlanabilir. Tasarımın işlevi yazılım programı içerisinde bulunan belirli test bölümleri vasıtasıyla kontrol edilebilir. Bu sayede sistem, donanım yapısı kurulmadan bilgisayar ortamında test edilebilir. Testlerin donanıma ihtiyaç duyulmadan yapılabilmesi tasarım süresinin kısalmasını sağlar (Stanley and Patricia 1996).

3.4.2 VHDL ve donanım tasarımı karşılaştırılması

VHDL dili kullanılarak tasarım yapılmasının standart donanım tasarım yöntemine göre bazı önemli üstünlükleri vardır.

Tasarım süresi: Teknolojinin hızla gelişimi ile beraber gerçeklenen bir devrenin kullanım ömrü azalmaktadır. Bu, devrenin tasarım zamanının kısıtlanması anlamına gelir. Böyle durumlarda, devrenin optimum tasarımlanmış olmasının ötesinde tasarım süresinin kısalığı ön plana çıkar. VHDL dili kullanılarak tasarım yapılması özellikle bu noktada tasarımcıya önemli yararlar sağlar. VHDL dili doğrudan donanım tasarımına göre, tasarım süresi açısından çok daha kısa sürede sonuçlanır.

Tasarım esnekliği: Teknolojideki değişimle birlikte kullanılan elamanların yapıları değişmektedir. Yapı değişiklerinin daha önce yapılmış tasarımlarda çalışabilmesi için, kullanılan tasarım ortamının buna uygun olabilmesi gerekir. VHDL dili fonksiyon bağımlı olarak çalışır. Dönüştürücü programlar yardımıyla yazılımın donanım yapısı oluşturulur. Teknoloji değişimleri durumlarında sadece bu dönüştürücü programların yeni teknolojiye uygun hale getirilmiş olması yeterli olacaktır.

Tasarım kolaylığı: Genel olarak VHDL dili kullanarak yapılan tasarımlarda, klasik donanım tasarımına göre, donanım bilgisine daha az ihtiyaç duyulur.

Yenileme kolaylığı: Teknolojideki hızlı değişim, gerçeklenen devrelerin değişim sürelerini azaltmıştır. Bu nedenle yapılacak değişimlerin hızlı bir şekilde

gerçekleştirilebilmesi gerekmektedir. VHDL dili ile yapılan tasarımlarda değişim esnekliği tasarımcının yazılım gücü ile sınırlıdır.

3.4.3 VHDL dili mimari yapıları

VHDL dili 3 mimari yapıdan oluşur.Bunlar aşağıda açıklanmaktadır.

3.4.3.1 Davranışsal mimari

Davranışsal mimaride, sistemin yapması gereken işlemler, 'process' yapıları kullanılarak yapılır. Ancak tasarımın nasıl gerçekleneceği konusunda bilgi verilmez. VHDL programı derleyicisi, varsa uyarılarını ya da hatalarını, tasarımın ne kadar yer kaplayacağını kullanıcıya bildirir. Bu nedenle bazı durumlarda programda yapılan küçük değişimler, gerçekleme alanı bakımından büyük değişiklere neden olabilir. Bunun için tasarım yapılırken sürekli olarak donanım düşünülmelidir.

Davranışsal mimaride 'process' ile birlikte duyarlılık ifade eden parametreler parantez içinde yazılır. Verilen parametrelerde değişim olması durumunda, program yukarıdan aşağıya doğru gerçekleştirilir. Bir mimaride tek 'process' olmak zorunluluğu yoktur. Çoklu yapılarda tüm 'process'ler aynı anda gerçeklenir ancak 'process'içindeki işlemler yukarıdan aşağı doğru yapılır. Böylelikle aynı program içerisinde bağımsız bloklar oluşturulabilir.

3.4.3.2 Veri akışı mimarisi

Veri akışı mimarisinde devre tasarımı; karşılaştırıcılar, toplayıcılar, kod çözücüler ve basit lojik kapılar kullanılarak tasarlanır. Program içerisindeki satırlar tamamen eşzamanlı olarak çalışır. Davranışsal mimariden farklı olarak duyarlılık ifade eden parametreler, program satırlarında ki eşitliklerin sağ tarafında kalan parametrelerdir. Bu tür mimaride davranışsal mimariye göre dışarıdan bakıldığında yapılacak işlemin anlaşılırlığı daha azdır. Aynı zamanda yapılacak işlemin gücü büyüdükçe her fonksiyonu standart elemanlarla ifade etme zorluğu nedeniyle, genelde kullanıcı

davranışsal mimariyi tercih eder. Ancak davranışsal mimariye göre, devrenin kaplayacağı alan bakımından daha kontrollüdür.

3.4.3.3 Yapısal mimari

Yapısal mimari temel olarak tamamen kullanıcının kontrolündeki bir tasarım biçimidir. Bu tasarımda tüm bağlantılar yazılımcı tarafından tanımlanır ve işaret isimleri verilerek belirlenir. Ayrıca kullanılacak elemanlar yazılımcı tarafından 'component' olarak oluşturulur ve tasarımda kullanılır.

Yapısal mimari donanım tasarımında yapılacak çizimin, yazılım ile gerçeklenmesi olarak görülebilir. Bu nedenle sistem karmaşıklığı arttıkça bu tasarımın kullanımı çok zorlaşacaktır. Küçük projelerde ya da genelde aynı yapıların tekrar ettiği sistemlerde kullanılabilir. En büyük avantajı tasarımın kaplayacağı alanın kullanıcı kontrolü altında olmasıdır.

Örnek olarak bir karşılaştırıcının VHDL kullanılarak üç farklı mimari ile yazımı aşağıda verilmektedir.

Bildirim bölümü her mimari için aynı şekilde yapılır. Fonksiyonların bildirildiği bölümler farklılık gösterir.

```
entity karşılaştırıcı is
port    (X,Y : in std_logic;
Z: out std_logic);
end karşılaştırıcı;

--Devre işleyişlerinin verildiği bölümler
--Davranışsal mimari

architecture davranış of karşılaştırıcı is
```

```
begin
process (X,Y)
begin
if (X=Y) then
C<='1';
else
C>='O';
end if;
end process;
end davranış;

--Veri akış mimarisi
architecture veri_akışı of karşılaştırıcı is
begin
C<=not(A xor B);
end veri akışı;

--Yapısal mimari
architecture yapısal of karşılaştırıcı is
component XOR
port (G1,G2:in std_logic;
Ç:out std_logic);
End component;
component NOT
port  (G: in std_logic;
Ç:out std_logic);
end component;
signal I:std_logic;
begin
U0: XOR port map(X,Y,I);
U1: NOT port map(I,Z);
End yapısal;
```

3.4.4 VHDL temel özellikleri

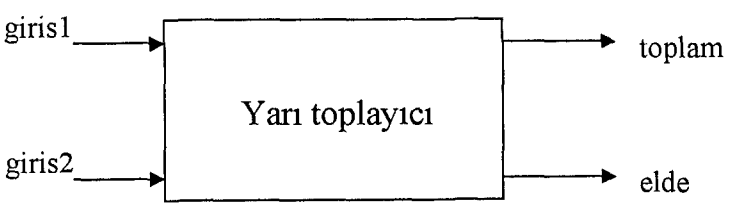
Bu bölümde VHDL dilinin bazı önemli özellikleri ve temel kullanım kuralları verilecektir.

3.4.4.1 Yapısal ve davranışsal tanımlamalar

VHDL dilinde 'component' tanımlamaları 'entity' birimleri içerisinde gerçekleştirilir. Her 'component' gerçeklemesi iki bölümden oluşur. Birinci bölümde yapının hangi giriş çıkış ucundan oluşacağı bildirimi yapılır. Örnek olarak bir yarı_toplayıcının bildirim yapısı şu şekilde ifade edilir.

```
etity yari_toplayici is
port(
giris1 : in bit;
giris2: in bit;
toplam : out bit );
end yari_toplayici;
```

Gerçeklenen yazılım ile Şekil 3.15'de gösterilen giriş ve çıkışlar oluşturulur.



Şekil 3.15. Yarı toplayıcı giriş ve çıkışları

Bu bildirimde kullanılan port tanımlamaları 4 farklı tipte yapılabilir.

- in : Sadece okunabilir. Giriş için kullanılır.
- out : Sadece yazılabilir. Çıkış için kullanılır.
- buffer : Okunabilir yazılabilir. Devrenin içinden bir süreni olabilir.
- İnout : Okunabilir yazılabilir. Giriş ve çıkış için kullanılabilir.

İkinci bölümde ise 'component' davranış biçimi ifade edilir. Örnek yarı toplayıcıya ilişkin davranışsal tanımlama bölümü aşağıdaki gibi yapılabilir.

```
architecture davranissal_tanimlama of yari_toplayici is
begin
process
toplam <= giris1 xor giris2 after 10 Ns;
elde <= giris1 and giris2 after 10 Ns;
wait on giris1,giris2;
end process;
end davranissal_tanimlama;
```

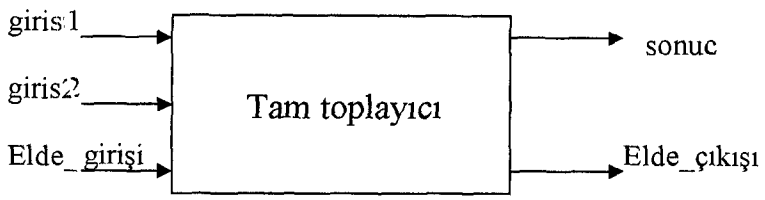
Bu yapı içerisinde kullanılan 'after' ifadeleri, gerçeklemede herhangi bir etki yapmaz. Sadece simülasyonlarda gecikme amacıyla kullanılırlar.

Yapısal tanımlamanın belirtilmesi ile yarı toplayıcı oluşturulmuştur. Bu şekilde oluşturulan yapıların başka bir yazılım tarafından kullanılmaları durumunda ifade edilen port yapıları yeni sistem için, sistem içi işaretleri gösterir ve VHDL dilinde signal olarak ifade edilirler. Aşağıda bununla ilgili olarak bir tam toplayıcı örneği verilmiştir.

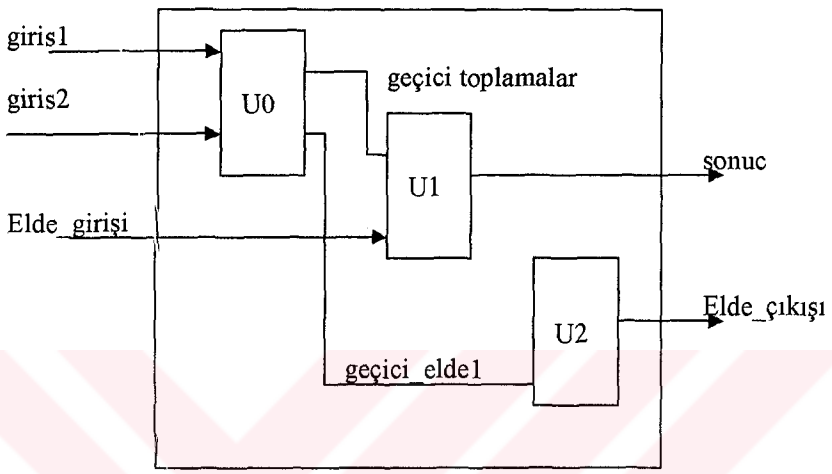
```
entity tam_toplayici is
port(
giris1 : in bit; giris2 : in bit; elde_giris : in bit;
sonuc : out bit; elde_cikis : out bit);
end tam_toplayici
```

```
architecture yapi of tam_toplayici is
signal gecici_toplam : bit;
signal gecici_elde1 : bit;
signal gecici_elde2 : bit;

component yari_toplayici
port(giris1 : in bit; giris2 : in bit;
toplam : out bit; elde : out bit);
end component;
component or_kapisi
port( g1: in bit; g2 : in bit; C1 : out bit);
end component;

begin
U0 : yari_toplayici
port map(
giris1 => giris1, giris2 => giris2,
toplam => gecici_toplam, elde => gecici_elde1);
U1 : yari_toplayici
port map(
giris1 => gecici_toplam, giris2 => elde_girisi,
toplam => sonuc, elde => gecici_elde2);
U2 : or_kapisi
port map(
g1 => gecici_elde1, g2 => gecici_elde2,
C1 => elde_cikisi);
end yapi;
```

Yukarıda örnekte görüldüğü gibi kullanılan her 'component' bir tanımlayıcı ile etiketlenmiştir. Bu etiketten sonra kullanılan 'component' lerin yapı içerisindeki işaret yerleri ve bağlantıları ifade edilir. Tüm bu tanımlamalar sonucunda VHDL dili Şekil 3.16 ile oluşturulan tanımlamayı, Şekil 3.17 ile gösterilen yapı biçiminde gerçekleyecektir.



Şekil 3.16. Tam toplayıcı giriş çıkışları



Şekil 3.17. Tam toplayıcı iç yapısı

3.4.5 Veri türleri ve nesneler

VHDL dilinde değer tutan her şey bir nesne olarak görülür. Her nesne de kendi yapısını tanımlıyan bir türe sahiptir.

3.4.5.1 Veri türleri

VHDL'de, sıkça kullanılacağı düşünülen belirli veri türleri yazılım içerisinde oluşturulmuştur. Örneklerde görülen 'bit' deyimi bahsedilen türlerden biridir. Ancak VHDL'de bu konu ile ilgili olarak esnek bir yapı mevcuttur.Kullanıcı kendi veri türünü oluşturabilir.aşağıda bazı veri türü tanımlama örnekleri bulunmaktadır. Bu

örnekler genel olarak 'integer' yapısında olan tür tanımlamalarını gösteren örneklerdir :

type x is range-10 to 10;
type y is range 40 downto 30;
type z is range-1E10 to 1E10;

'integer' türünde de tanımlamalarının yanı sıra kayan nokta türünde tanımlamaları da yapılabilir. Bu tanımlamalarda sınır tam sayılardan oluşmaz. Aşagıda bununla ilgili bir örnek mevcuttur.

Type k is range -2+0.1E-20 to 2+0.1E20;

Bu iki farklı tür tanımlamaları dışında kalan tüm tanımlamalarda 3.gurup tür tanımlamaları olarak düşünülür. 3. gurup tür tanımlamalarına en iyi örnek olarak 3 seviyeli lojik tasarım düşünüldügünde, ortaya çıkacak 3.seviyeyi tanımlama sorunu cevap verir. Burada 3 seviyeli parametresi, '0','1','Z'degerlerinden oluşturulacak şekilde tanımlanması durmunda 3 seviyeli yapı tanımlanmış olur. Bununla ilgili örnekler aşagıda verilmiştir.

type 3 seviye is('0','1','Z');
type mantık is(False,True);

3.4.5.2 Nesneler

VHDL'de 3 farklı nesne tanımlamaları bulunmaktadır.Bunlar,işaretler (signals),degişkenler (variables) ve sabitlerdir(constants).Tüm tanımlamaların farklı özellikleri aşağıda sıralanmıştır.
İşaretler donanımsal bağlantıyı ifade eden nesne türleridir. Degişimleri bağlı oldukları işaretlerin degişimleri ile gerçekleşir. Doğrudan değerinin degişimine yönelik işlem yapılmaz. Degişkenler, işaretlerden farklıdır. Değerinin değişimi için işlem yapılmalıdır. İşlem sonucunda elde edilen değişim sonucu, bir sonraki yazıma

kadar kalır. Son tanımlama biçimi ise sabitlerdir. Bunlar bu iki tanımlamadan faklı olarak bir kez değer alırlar ve bu değeri sürekli olarak korurlar. Bu anlamda bir ROM gibi düşünülebilir. Aşağıda tanımlamalarla ilgili örnekler sunulmuştur.

Constant x:Integer :=1 6#FFFF# ;
Variable y :Boolean;
Signal z :Bit ;

3.4.6 Arayüz listeleri

Arayüz listeleri VHDL dili içerisinde 4 farklı yapıda kullanılır. Bunlar 'entity' tanımlamaları, 'block' ifadeleri ve altprogram tanımlamalarıdır.Tüm tanımlamaları oluşturan elemanlar parantez içerisinde yer alacaktır. Her eleman aşağıdaki verilen 3 vasıfla birlikte tanımlanır.

Nesne sınıfı (sıgnal, constant yada variable)
Verinin akış yönü (in, out, inout yada buffer)
Veri türü (Bit vb...)

Her tanımlama türü için izin verilen nesne sınıfları ve modlar bulunmaktadır.

3.4.7 VHDL dili ana yapıları

VHDL içerisinde,bazı önemli yapılar sıkça kullanılır. Bunlar aşağıda sıralanmaktadır.

3.4.7.1 'Entity' tanımlamaları

'Entity'tanımlamaları her tasarım içerisinde mutlaka bulunması gereken bir yapıdır.

Genel yazım kuralı aşağıdaki gibidir.

entity kimlik is
generic interface listesi ;
port interface listesi;
Tanımlamalar
begin
ifadeler
end kimlik ;

Bu tanımlamalar içerisinde bildirimler yapılır. Bu bildirimler, tür bildirimleri, alttür bildirimleri, sabitlerin bildirimleri, dosya bildirimleri, altprogram bildirimleri ve 'use'ifadeleridir.

3.4.7.2 'Architecture' yapıları

'Architecture' yapıları genel yazım biçimi aşağıdaki gibidir.

architecture kimlik of kimlik hedefi is
bildirimler
begin
ifadeler
end kimlik;

Yapı içerisinde ifade edilen bildirimler bölümü kapsamında temel tanımlamalar, işaret tanımlamaları, altprogram yapısı, nitelik tanımlamaları ve özellikleri, 'component'tanımlamaları mevcuttur.

3.4.7.3 Alt programlar

Altprogramlar VHDL' nin önenli yapılarındandır. Program içerisinde tekraren çağrılabilirler. VHDL dili 'procedure' ve 'function' olmak üzere iki tür altprogram yapısını desteklker. Birinci tür altprogramlar dönüş değeri almazlar. Altprogram tanımlamaları sadece 'interface' bilgisini içerir. Altprogram yapısı ise 'interface' bilgisi, yerel tanımlamalar ve ifadeleri içerir.Altprogram tanımlaması ile altprogram yapısı arasındaki fark, 'entity' bildirimi ve 'architecture' yapısı arasındaki fark gibidir. Altprogram bildirimleri aşağıdaki gibi yapılır.

procedure kimlik interfece listesi
function kimlik interface listesi return dönüş türü

Bu bildirimlerde, 'interface' listelerinin olma zorunluluğu bulunmamaktadır.Yani altprogramlar parametresiz olabilir.Bununla ilgili olarak bazı örnekler verilmiştir.

```
procedure A ;
function B return byte;
procedure C(x:inout İnteger):
function D(x:İnteger )return byte;
procedure C(variable x:inout İnteger);
function D(constant x:in İnteger )return byte;
```

Altprogram tanımlamaları aşağıda ifade edildiği gibi yapılabilir. Her iki altprogram türü için aynı yapı geçerlidir.

```
altprogram özelliği is
bildirimler
begin
ifadeler
end kimlik;
function deneme (X,Y: byte) return byte
begin
```

```
if (X>2)
return X;
else
return Y;
end if;
end deneme ;
```

3.4.7.4 'Package' ve 'Use' yapıları

Tüm yazılım sistemlerinde olduğu gibi birden çok yerde kullanılan elemanları ortak olarak kullanmak için belirli yapılara imkan verilmiştir.VHDL dilinde ortak kullanım oluşturma yapısını 'package'meydana getirir.Tanımlama biçimi ve ilgili bir örnek aşağıda verilmiştir.

```
package kimlik is
tanımlamalar
end kimlik ;

package lojik is
type 3 seviye is ('0','1','Z');
constant bilinmeyen değer :3 seviye:='0':
function tümleme (
giriş: 3 seviye )
return 3 seviye;
end lojik ;
```

Bu örnekte içerisinde 'fuction' tanımı bulunan bir 'package' bildirim yapısı verilmiştir. Genel yapıları ise

```
package body kimlik is
bildirimler
end kimlik;
```

Şeklinde gösterilebilir. Verilen örneğin yapı yazılımı ise aşağıdaki gibiolabilir.

```
package body lojik is
function tümle(
giriş:3 seviye)
return 3 seviye is
begin
case giriş is
when '0'=>
return '1';
when '1'=>
return '0';
when 'Z'=>
return 'Z';
end case;
end tümle;
end lojik;
```

Bu yapılarının kullanımları 'use' ifadeleri ile gerçekleştirilir. 'use'yazımından sonra kullanılacak 'package' adı yazılır. Hemen ardından nokta ve sonrasında ise yapı içerisinde kullanılacak tanım yazılır. Başka kullanılacak yapılar varsa virgülden sonra aynı mantık içerisinde yazılır. 'package' içindeki tüm tanımlamalar kullanılacak ise bu durumda isim yazımı ardından nokta ve sonrasında 'all'ifadesi yazılarak, tüm tanımlamalar alınmış olacaktır. Bununla ilgili olarak yukarıdaki örnekteki yapıyı kullanan bir kod yazılabilir.

```
use lojik.all ; Tüm tanımlamalar alındı.
entity tümle is
port (X : in 3_seviye ; Y : out 3_seviye);
end b tümle;

arhitecture tümle_yapısı of tümle is
```

```
begin
process
begin
Y<=tümle(X) 10 ns;
wait on X;
end process;
end tümle_yapısı;
```

3.4.7.5 Tasarım kütüphaneleri

VHDL'de yazılan tanımlamalar, yazım kurallarına uyduğu doğrulandıktan sonra sentezleme ve sirnülasyonda kullanılmak üzere kütüphanelerde saklanır. Bir kütüphane tek başına analiz edilebilir. Ancak birbirine bağımlı yapıların belirli bir sırasıyla analiz edilmesi oluşabilecek hataları engeller

BÖLÜM.4. UYGULAMA

Bu çalışmada geriyayılım ağı Xilinx Spartan 2E entegre devresinde gerçeklenmiştir. Bu tümleşik devre 200000 lojik kapı, 2352 dilim ve 14 blok RAM den oluşur. Bir dilim, 4 girişli look-up tablosu ve 2 flip-flop'dan oluşan lojik mimaridir.

Ağ öncelikle Matlab'da eğitilimiştir. Eğitim sonunda elde edilen ağırlıklar VHDL paket dosyasına yerleştirilmiştir. Bu dosya diğer VHDL dosyalarıyla beraber sentezleme ve gerçekleştirim aşamalarından geçirilmiştir. Bu adımlar Xilinx ISE yazılım araçları kullanılarak gerçeklenmiştir. Simülasyon aşaması ise ModelSim tarafından gerçeklenmiştir.

4.1 Veri Gösterimi

Donanım gerçeklenmesine başlamadan önce giriş, ağırlıklar ve aktivasyon fonksiyonu için sayı duyarlılığı düşünülmelidir. Tasarımın sayı duyarlılığını artırmak FPGA'in kaynaklarının üstel olarak artırılması demektir.

Öğrenme aşamasında sayıların duyarlılığının mümkün oldukça yüksek alınması çok önemlidir. Bununla beraber test aşamasında düşük duyarlılıklı sayılar kabul edilebilir (Stevenson et al 1990). Sonuç hatası sınıflandırma uygulamaları için önemsenmeyecek kadar küçüktür (Blake et al 1998, Krips et al 2002, Ossoinig et al 1996).

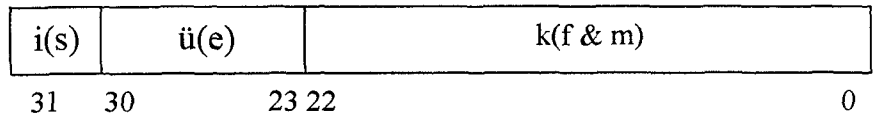
YSA sistemlerinde genellikle analog verileri işlenir. Oysa ki FPGA dijital bir devre elemanıdır. FPGA giriş değerleri olarak sadece dijital (0 veya 1)verileri kabul eder. Uygulama safhasında yapılması gereken ilk çalışma; analog bir sistemin dijital değerlerle ifade edilmesidir. Analog hesaplama sistemini dijital sayı değerleriyle

hesap etmek için bir çok sayı formu geliştirilmiş ve satandartlaştırılmıştır. Fakat bunlardan duyarlılık bakımından en göze çarpanı IEEE'nin, kayan noktalı sayılar ve sabit noktalı sayılar fomatlarıdır. Bu projede, hassasiyet derecesinin daha yüksek olmasından dolayı IEEE 754 formantındaki (IEEE 1981) tek duyarlıklı kayan noktalı sayilar ile çalışılmıştır. Bu format sayesinde gerçel sayılar, tek duayrlılıkta (32 bit) veya çift duyarlılıkta (64 bit) gösterilme imkanına sahiptir. Gerçel sayılar bilgisayarda 32 bit formata çevrildikten sonra FPGA'e girilmektedir. Tabiki FPGA içindeki bütün sayısal işlemler kayan noktalı sayılar formatına uygun olarak yapılmak zorundadır. Bu noktada kayan noktalı sayılara ait dört işlem algoritmalarının VHDL ile gerçeklenmesi gerekmektedir.

4.1.1 Kayan noktalı sayılar aritmetiği

Bir çok algoritma sayı gösterimindeki dinamiklik ve hasas işlem yapma kapasitesi sabebiyle kayan noktalı sayıları tercih etmektedir. Birçok avantajına rağmen kayan noktalı sayılar sıradan bir uygulama için bile aşırı derecede donanım kaynağı tüketmektedirler.

4.1.2 Kayan noktalı (Floating Point) sayıların gösterimi



i (sign) : işaret biti, 1 bit ile ifade edilir.(31)

ü (exponenet) : üs bitleri , 8 bit ile ifade edilir.(30-23)

k (fraction & mantisa) : kesir bitleri, 23 bit ile ifade

Şekil 4.1. IEEE 754 32 Bit Kayan sayı formatı

Gerçel sayıları Şekil 4.1 de görülen 32 bit IEEE format çevirmek için;

$$v = -1^{s} 2^{(e-127)} (1.f) \tag{4.1}$$

formülü kullanılabilir. Formülde üs(exponenet için kullanılan 127 sayısı bias değeridir.

Örneğin, -3.625(ondalık) veya -11.101(ikilik) sayısı flaoting point formatta aşağıdaki gibi ifade edilir.

-3.65 = -11.101 = -1.1101x2^1 burada, s =127+1, 1.f =1.1101, s =1 dir.

$$v = -1^{1} 2^{(128-127)} (1.1101) \tag{4.2}$$

burada;

s = 1(onluk), e = 128(onluk) ve f = 680000(onaltılık).

Sonuçta -3,625 sayımız, flaoting point formatta, C0680000(onaltılık) şeklinde gösterilir.

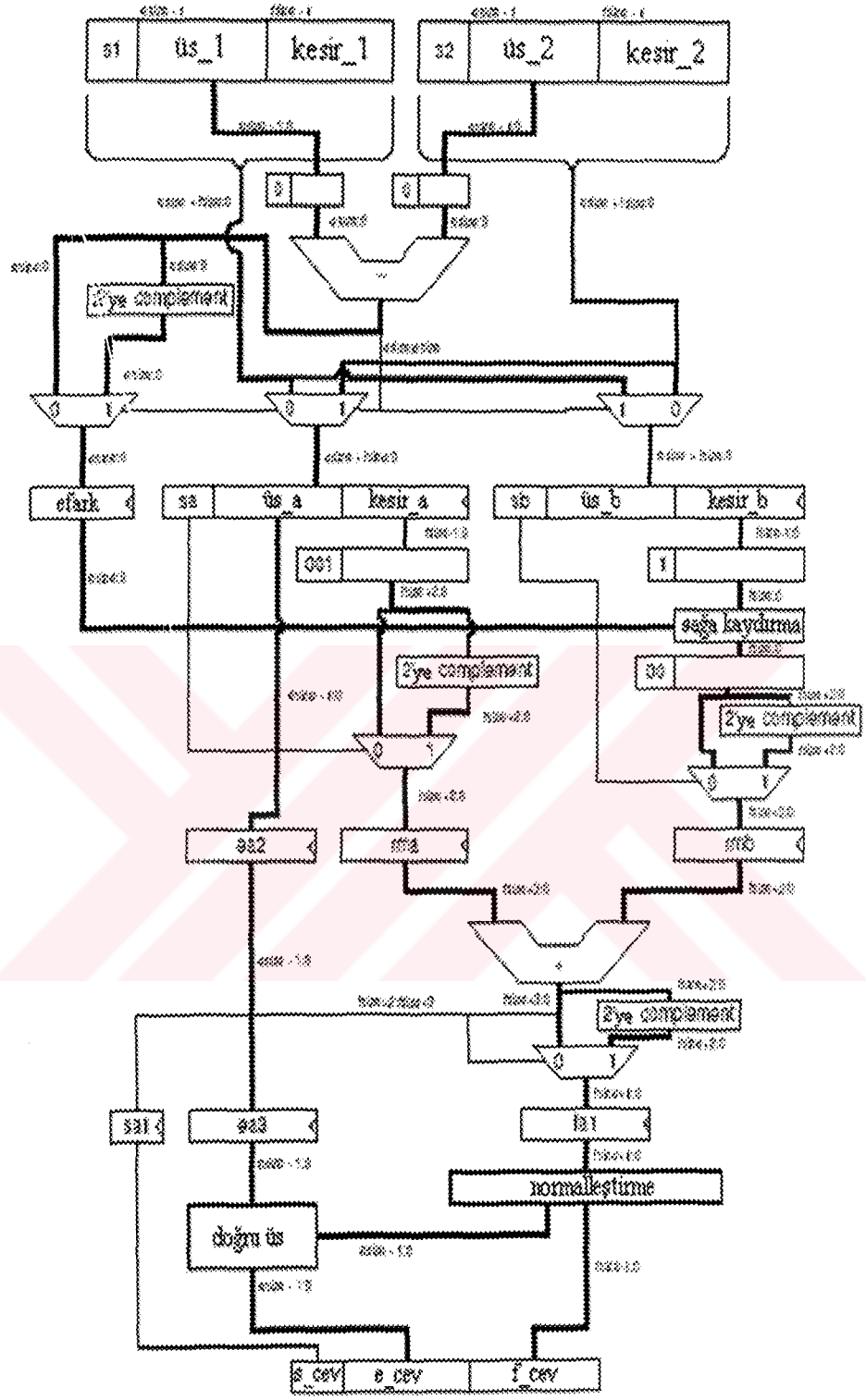
4.1.3 Toplama ve çıkarma

Toplama veya çıkarmaya işlemine tabi tutulacak sayılar F_1 ve F_2 olsun.

$$F_{toplam} = F_1 + F_2 \tag{4.3}$$

$$F_{minus} = F_1 - F_2 = F_1 + (-F_2) \tag{4.4}$$

şeklinde yazılabilir. Dolayısıyla sadece toplama algoritmasını gerçeklemek çıkarma yapmak içinde yeterli olacaktır. Aşağıda toplama algoritması verilmiştir (Bkz. Şekil 4.2).



Şekil 4.2. Kayan Noktalı sayıların toplanması

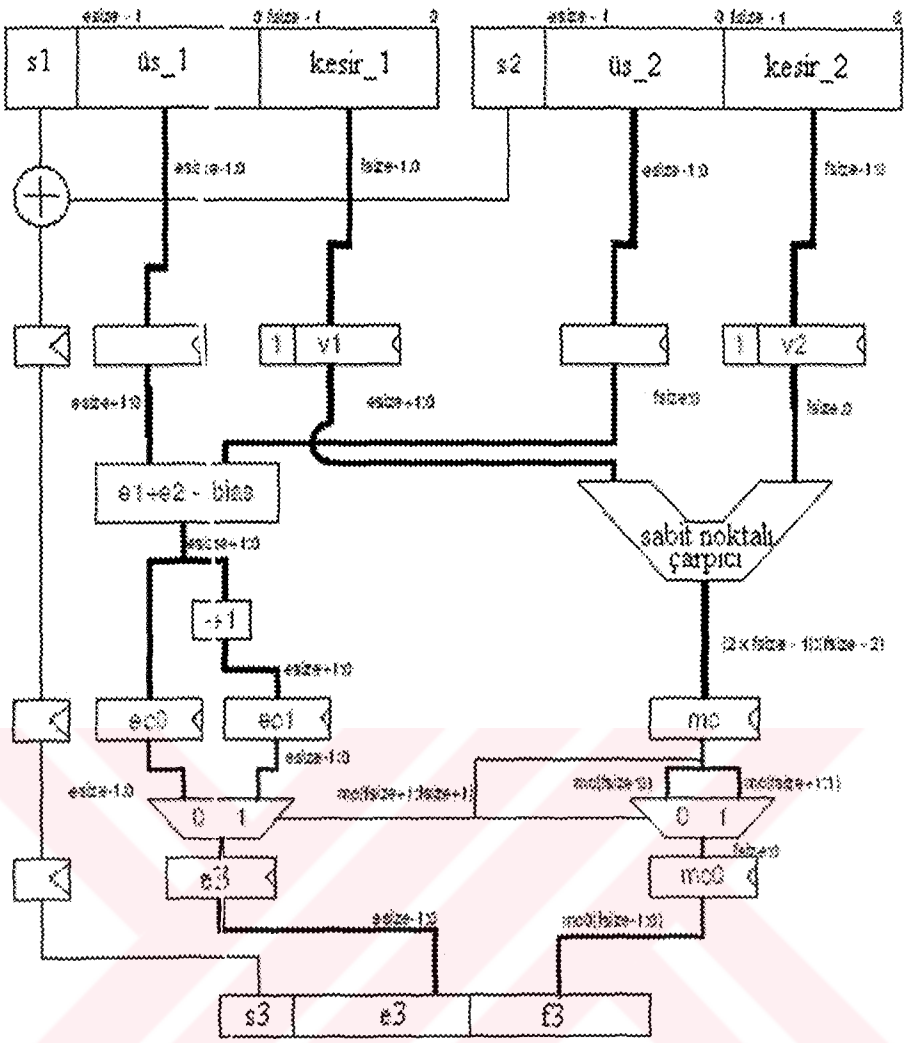
Şekil 4.2' de görüldüğü gibi toplanacak olan sayı1 ve sayı2 sayıları 1 bit işaret 8 bit üs ve 23 bit kesir olmak üzere toplam 32 bit ile ifade edilir. İlk aşamada giriş sinyallerinde büyük olan sayıa ve küçük olan sayıb olarak değiştirilir. İkinci aşamada üs_1'den üs_2 'i çıkarılır. kesir_a'ya sol taraftan 001 kesir_b'ye sol taraftan 1 eklenir ve kesir_b (üs_1-üs_2) kadar sağa kaydırılır. Sonra kesir_a ve kesir_b' arasında tamsayı toplaması yapılır. Sonra normalleşme yapılarak sonuç bulunmuş olur.

4.1.4 Çarpma

Toplama veya çıkarmaya işlemine tabi tutulacak sayılar F_1 ve F_2 olsun.

$$F_{çarpma} = F_1 + F_2 \tag{4.5}$$

şeklinde yazılabilir. Toplama algoritması Şekil 4.3 ile verilmiştir.



Şekil 4.3. Kayan Noktalı sayıların çapılması

Çarpma işleminde toplama işlemi gibi çarpılacak olan sayı1 ve sayı2 sayıları 1 bit işaret 8 bit üs ve 23 bit kesir olmak üzere toplam 32 bit ile ifade edilir. İlk aşamada s1 ve s2 exor işlemine tabi tutulur. üs_1 ve üs_2'i toplama işleminden geçirilir. kesir_1 ve kesir_2 kısımları sabit noktalı çarpıcıdan geçirilerek sonuca ulaşılır.

4.2 Yapay Sinir Hücresi Yapısı

Bu çalışmada mimari içindeki işlem birimleri tek duyarlılıklı kayan noktalı sayılar düşünülerek oluşturulmuştur. Bu sebepten dolayı her YSH'nin kendine ait çarpma, toplama ve aktivasyon fonksiyonu olması imkanı ortadan kalkmış olmaktadır.

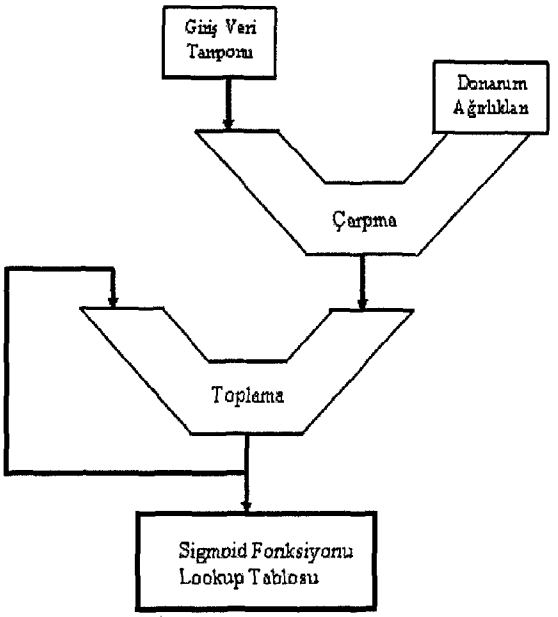
Mimaride bir toplayıcı ve bir çarpıcı vardır. Ve bu işlem birimleri her YSH tarafından zaman bölümlemeli olarak paylaşılır.

4.3 Ağ Mimarisi

Tamamen paralel mimariye sahip bir ağı FPGA ile gerçeklenmesi olasıdır. Tamamen paralel mimariye sahip ağ hızlıdır fakat esnek değidir. Bu tip ağ mimarilerinde YSH'ne düşen çarpıcı sayısı YSH bağlantılarıyla eşit miktardadır. Bu sebepten dolayı her katman için değişik yapay sinir hücresi mimarileri tasarlanmalıdır. Çünkü çarpıcı YSH yapısında en çok kullanılan elemandır. Tam paralel ağ için ikinci dezavataj kapı kullanımındaki artıştır (Krips 2002)

Amaç, gerçek hayata en uygun ve çalışma anında katman ve YSH sayısını değiştirebilen YSA'larını FPGA yapısında gerçeklemek olmuştur. Bu sebepten dolayı bu çalışmada paralellik tamamen göz ardı edilmiştir. Paralel yapının göz ardı edilmesi FPGA tümleşik devresinin en önemli özelliklerinden birinin kullanılamaması anlamına gelemektedir. Fakat oluşturulan YSA mimarisi oldukça esnek ve hassas ve yoğun işlemleri de çok rahatlıkla gerçekleştirebilen bir yapı elde edilmiştir.

Oluşturulan YSA mimarisinin akış diyagramı aşağıda verilmiştir.



Şekil 4.4 Gerçeklenen YSA'nın Akış Diyagramı

Diyagramdan da görüleceği üzere dış dünyadan gelen girişler(veriler) bir giriş tamponunda tutulmaktadır. Sonra bu girişler seri bir şekilde eğitilmiş ağın ağırlıklarıyla çarpma işlemine tabi tutulmaktadır. Çıkan sonuçlar ait oldukları YSH'ne göre depolanmaktadır. Her YSH ait ağırlıklarla çarpılmış girişler kendi aralarında seri bir şekilde toplanıp depolanmaktadır. En sonunuda bu toplamalar Sigmoid fonksiyonunu gerçekleyen bir look-up tablosuna giriş olarak verilirler ve çıkan sonuçlar depolanır.

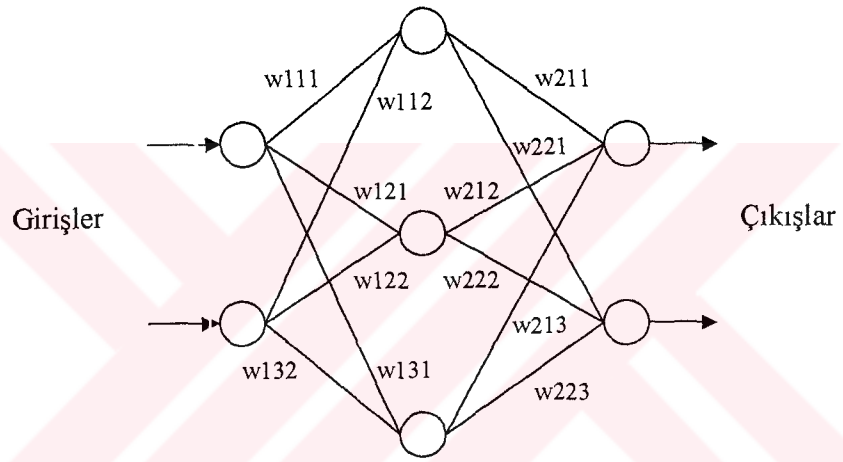
Şekil 4.4 yapı YSA mimarisinin sadece bir katmanını ifade edebiliyor. Eğer birden çok katmanlı bir YSA mimarisi oluşturulmak istenirse bu akış diyagramının katman sayısınca işletilmesi yeterli olacaktır.

Oluşturulan mimari hem katmanları hemde YSH lerini seri olarak işleyebilmektedir. Yani ağın çıkışındaki gecikme, katman sayısı ile bir katmanın işleme süresi çarpılarak bulunabilir. Bu yapının en büyük avantajı ise tek duyarlılıklı kayan noktalı sayıları kullanması sayesinde istenilen herhangi YSA yapısını önemsenmeyecek kadar küçük hata değerleriyle gerçekleyebilmesidir. Ayrıca VHDL kodları üzerinde

herhangi bir düzeltme yapılmadan dışarıdan nekadar giriş sayısı ve katman sayısı istediğimizi belirtebiliriz.

4.3.1 Üç katmanlı (2-3-2) YSA'nın modellenmesi

Gerçeklenmek istenen YSA ileri beslemeli bir ağdır. Ağa ait ağırlıklar bir ROM' da tutulmaktadır. Aktivasyon fonksiyonu olarak Denklem 4.6'da verilen sigmoid fonksiyonu kullanılmıştır. Sigmoid fonksiyonunun -10 ile 10 aralığında 0.2 adımlarla alınan 100 değere verceği cevap hesaplanmış ve bir ROM'a yazılmıştır. Şekil 4.5'de gerçeklenen ağın mimari verilmektedir.



Şekil 4.5 Üç katmanlı YSA

$$\varphi(v) = \frac{1}{1+e^{-v}} \tag{4.6}$$

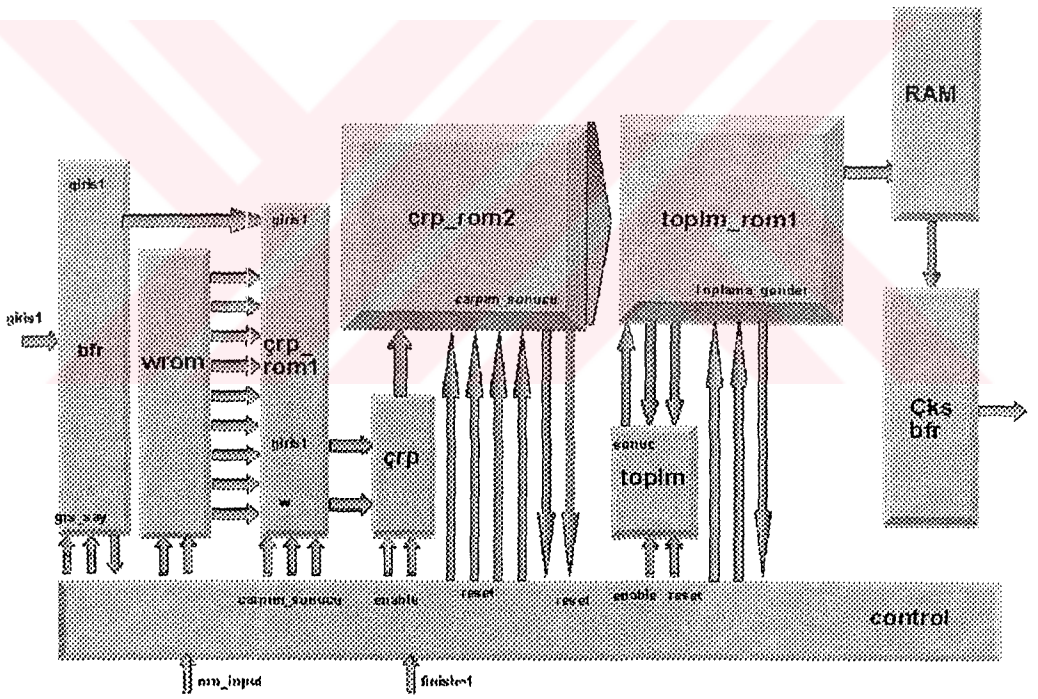
Şekilde verilen ağırlıklar (w) Tablo 4.1'de gösterilmiştir.

Tablo 4.1 YSA gerçeklenmesinde kullanılan ağırlıklar

w111	w121	w131	w112	w122	w132
0.5	1	0.5	1	0.5	1
w211	w221	w212	w222	w213	w223
0.5	1	0.5	1	0.5	1

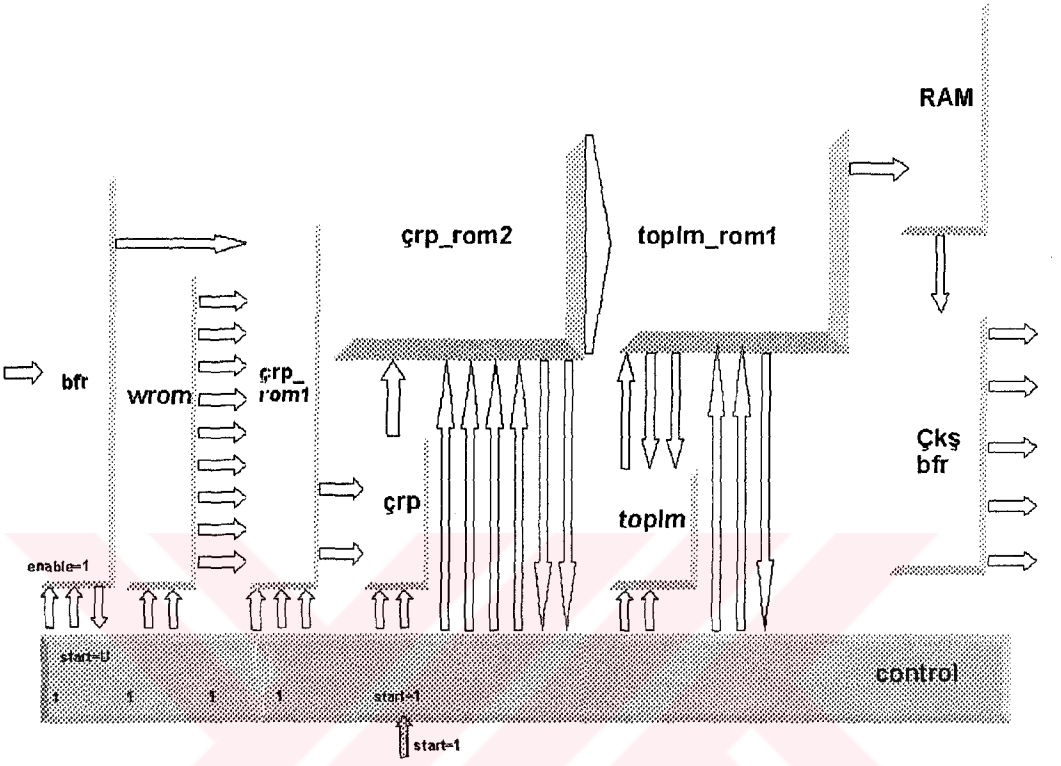
4.3.1 Üç katmanlı (2-3-2) YSA çalışması

Şekil 4.6'de gösterildiği üzere girişler kullanılan donanımın yetersizliğinden dolayı seri bir şekilde girilmiştir. Girişleri ile beraber ek olarak bitir ve YSH sayısı adında iki giriş daha girilmek zorundadır.



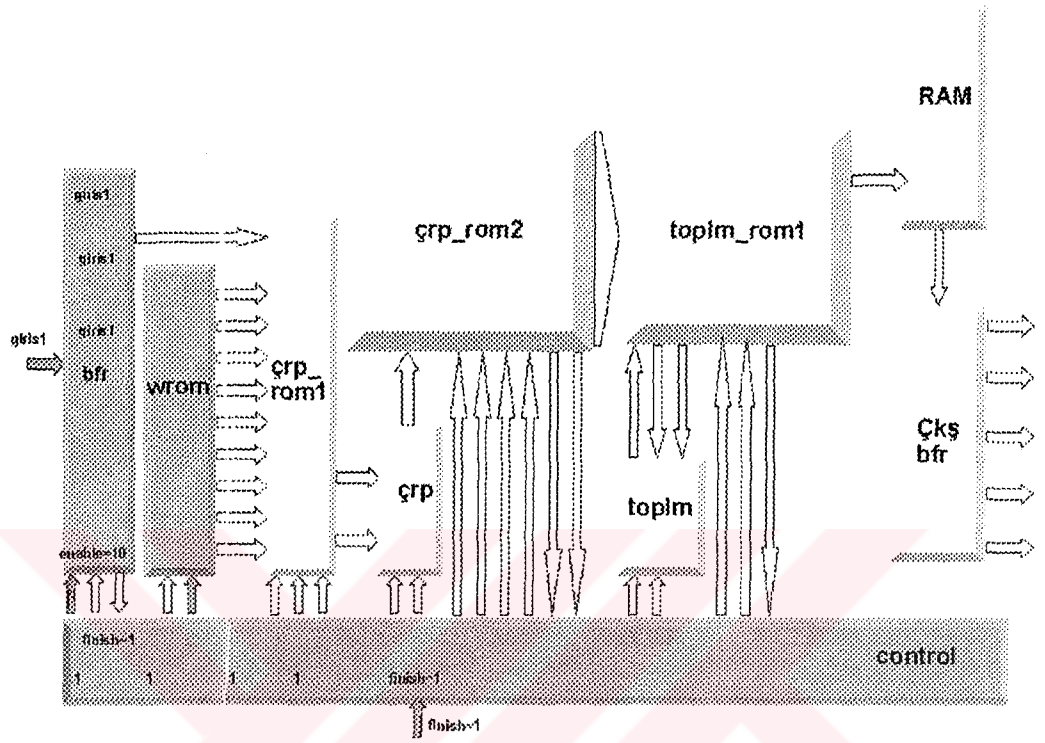
Şekil 4.6 Gerçeklenen YSA mimarisi

Girişlere başlamadan önce başla tuşuna (bitir girisi =1) basılır sonra istenildiği kadar giriş girilir ve bitir tuşuna basılır. Bitir tuşunu alan sistem giriş sayısını otomatik olarak hesap eder.



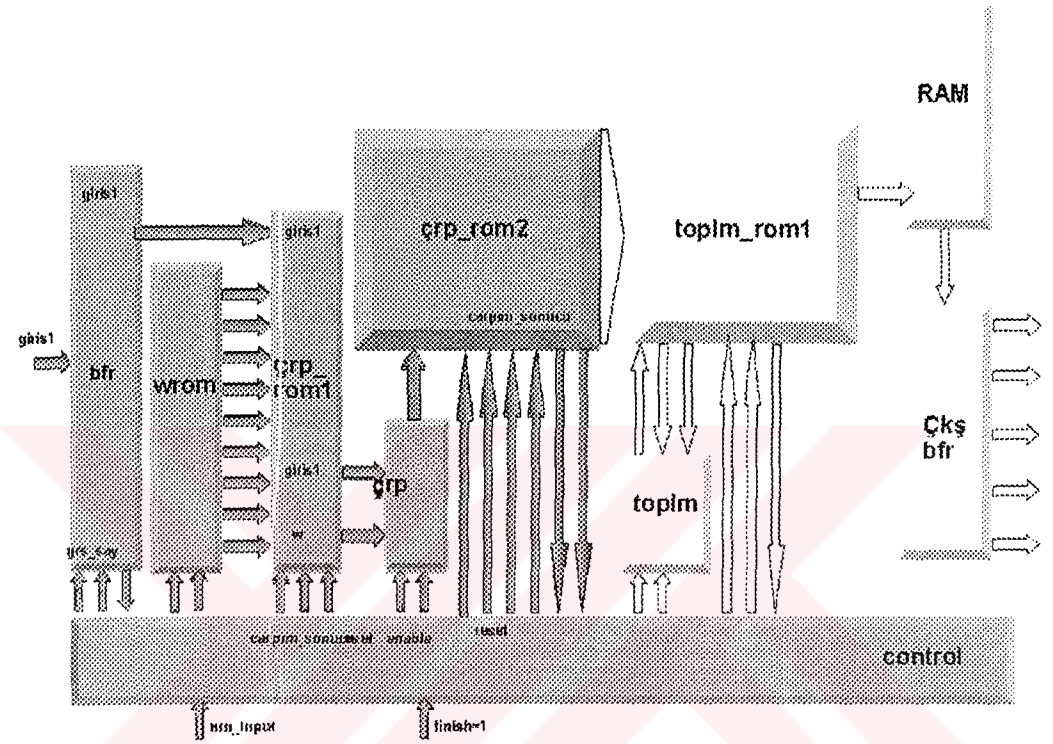
Şekil 4.7 Girişler sisteme girilmesi ve sayılarının hesap edilmesi

BFR'deki giriş sayısı ile YSH sayısı çarpılarak elde eldilen sayı kadar ağırlık WROM (Tablo 4.1'deki değerleri saklar)'dan ayrılır ve ilk girişle alakalı ağırlıklar sisteme verilir.



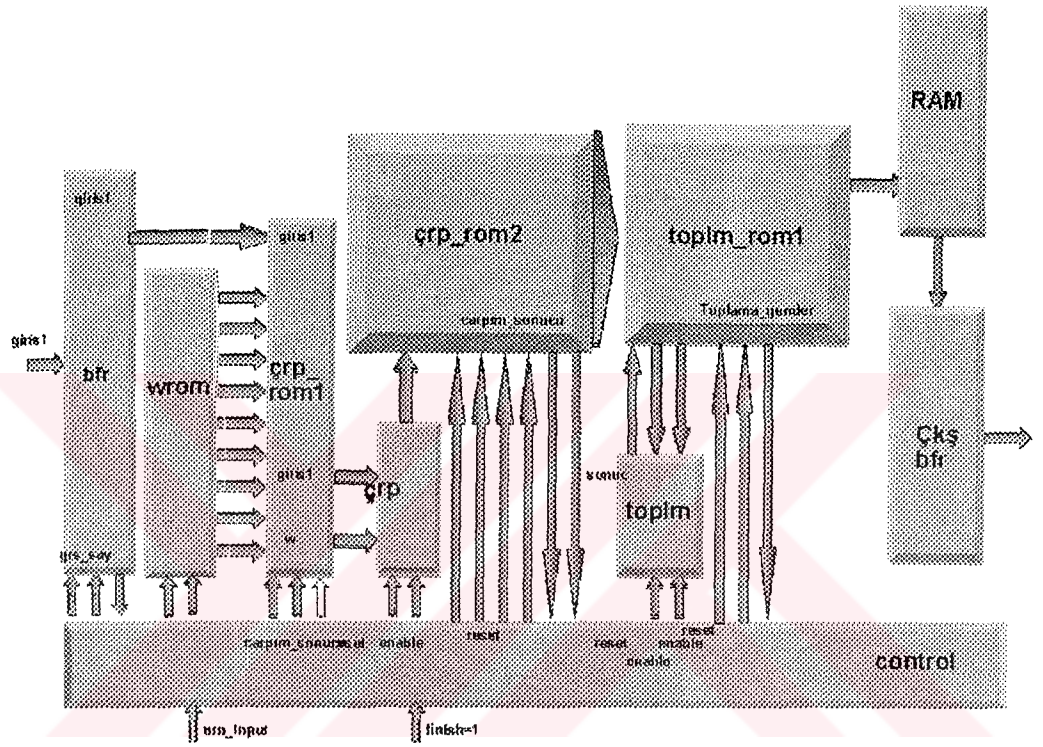
Şekil 4.8 Girişlere ait ağırlıkların sisteme verilmesi

İlk giriş kendisi ile alakalı ağırlıklarla beraber çarpma öncesi bir ÇRPROM1'de tutulur ve sonra bu ağırlıklarla seri bir şekilde çarpılır. Çarpım sonuçları, ÇARROM2'de bulunan ait oldukları YSH'nin alanına yazılırlar. İlk giriş ile ağırlıkların ÇRPROM2'de depolanması işlemi bitirildiğinde ikinci giriş için aynı işlemler tekrarlanır.



Şekil 4.9 Girişler ile ağırlıkların çarpılması

Sonunda toplama elemanına girmeye hazır girişler YSH'rine ait biaslar da göz önüne alınarak seri bir şekilde toplanırlar. Her YSH'ne ait toplamlar sigmoid look-up tablosuna (RAM) giriş olarak verilirler. RAM çıkışları bir ÇKBFR'de depolanır. Depolama işlemi bittiğinde artık bu ağ katmanının çıkışları diğer ağ katmanı için giriş olmuş durumundadır. Bütün bu işlemlerin zamanlamasını ve kontrolünü bir kontrol birimi üstlenmiştir.



Şekil 4.9 Toplama sonuclarının Ram'e aktarılarak çıkışa verilmesi

SONUÇLAR VE ÖNERİLER

YSA'nı bir çok uygulamada kullanırken karşılaşılan en büyük engel öğrenme aşamasındaki yavaş eğitimdir. Gerçek zamanlı bir çok uygulama hızlı öğrenme aşamasında hızlı bir eğitim ve test aşamasında da hızlı hatırlayabilmek ister. Bu tip problemlerin olabilecek çözümlerinden biri de YSA'larını tekrardüzenlenebilir tümleşik devreler üzerinde gerçeklemektir.

Aslında tekrar düzenlenebilir hesaplamanın manası (tek bir üniteye ait silikon alanındaki yüksek performans) işlem yoğunluğ artışı ve bunu ötesinde genel amaçlı hesaplama platformu sağlamasıdır (Watanabe et al 1993). FPGA programlanabilen bir lojik elemandır ve yazılım gibi esnek tasarım önerirler, fakat performans hızı ASIC den daha yavaştır. Tekrar düzenlenebilirlik yeteneği üretildiği zamandan itibaren sonsuz kezdir. FPGA teknolojisindeki artış ile birlikte bu elemanların tekrar düzenlenebilirlik isteyen hesaplamalarda sıkça kullanılmasına yol açmıştır. Fakat bu uygulamalarda seçilen sayıların duyarlılık derecesi iki aşamalı bir problemi de beraberinde getirir. İlk problem FPGA tabanlı bir YSA'nın egitim hızı ve doğruluk derecesi için makul değerde sayısal duyarlılık ne olması gerektiğidir. İkinci problem sie bu sayı duyarlılığı ve alan maliyeti (ki bu maliyet doğrudan duyarlılık artımıyla ilişklidir.) arasındaki dengenin nasıl kurulacağıdır.

Uygulamaya başlamadan önce sorulması gereken soru şudur. Özel uygulamalar için doyumun olamayacağını garanti eden yeterli hassasiyette sahip olan uygun sayı duyarlılığını nasıl seçeceğim? Hasas hesaplamalar içeren uygulamalar için 32 bit kayan noktalı sayılar ideal duyarlılık olabilir. Genel amaçlı hesaplama uygulamalarının bir çoğunda kayan noktalı sayılar yüksek nicelendirme

kabiliyetlerinden dolayı kullanılırlar. Fakat FPGA'daki sınırlı donanım kaynakları kayan noktalı sayılar yerine daha az donanım kaynağı tüketen sayı sistemlerinin (örneğin sabit noktalı sayılar) kullanımını zorunlu kılar.

5.1 Alan Tasarrufu ve Duyarlılık Oranı

Genel amaçlı uygulamalar için yapılan tekrar düzenlenebilir hesaplamaların yoğunluk dezavantajlarının üstesinden gelmenin bir yolu da donanım kaynakların etkin bir şekilde kullanmaktır.

Burada alan tasarrufunun ve optimal duyarlılık oranının ölçütü en aza indirgenmiş donanım alanı kullanımı ile birlikte kalite performansından ödün vermemektir. Bu iki düşünce minimum duyarlılık oranı olarakta birleştirilebilir.

Sayı duyarlılığının azaltımı sitemdeki hataları çoğaltacaktır. Kullanılabilir minimum sayı duyarlılığı, performans azaltımına direnebilen uygulamalarda hesaplama belirsizliklerini ortadan kaldırabilecek maksimum duyarlılık miktarını belirleme sorunudur. Ayrıca, dinamik hasaplama oranları sınırlandırılabilir ki bu sınırlama doyumluluğa yol açar. Bu nedenle donanım için kullanılabilen duyarlılık oranı ve uygun sayısal gösterim genellikle uygulamaya ve kullanılan algoritmaya göre değişir.

Neyseki, yapay sinir ağlarında kullanılan geri yayılım algoritmaları için duyarlılık oranı deneysel olarak geçmişte belirlenmiştir. 16 bit sabit noktalı sayı duyarlılığının geri yayılım algoritması için minimum izin verilen duyarlılık oranı olduğu kanıtlanmıştır. Ağın öğrenme kabiliyetinden ödün vermeden donanım kullanılarak yapılan geriye yayılım algoritması için izin verilen mimimum duyarlılık oranı 16 bit sabit noktalı sayılardır.

16 bit duyarlılık FPGA tabanlı yapay sinir ağı uygulamalarının bir çoğunda kullanılmaktadır ve donanım kaynaklarının kullanım sıkıntısını kısmen de olsa ortadan kaldırmıştır. Bu uygulamalardaki hassasiyet hatası önemsenmemiştir.

Hassasiyet hatalarını dağıtma kaygısı olmadan, 32 bit duyarlılık kullanımı ise öğrenme aşamasında kullanılmalıdır. İşlem yoğunluğu az olan test aşaması olan uygulamalarda 32 bit gösterimi kullanılabilir.

Bu aşamada, sabit noktalı sayılar yoğun işlem yapabilme avantajı sağlar fakat bunun yanında kayan noktalı sayılar hassas ve dinamik hesaplamaları yaparken ortaya çıkabilecek doyum riskini ortadan kaldırmaktadır.

Sabit sayılar üzerinde kayan noktalı sayıların hassasiyet avantajaları eski kuşak Xilinx 4020E FPGA üzerinde uygulandı ve başarılı sonuçlar alındı (Rucket 1993).

Ancak yapay sinir ağı açısından bir 32 bit sabit noktalı sayı, 32 bit kayan noktalı sayı kadar esnek olamaz. Gelecek yıllarda FPGA mimarisinde ve beraberindeki geliştirme araçlarındaki hızlı gelişim ve bununla birlikte aritmetik devre tasarımındaki (yer/zamanoptimizasyonu) ilerlemeler 32 bit duyarlıklı kayan noktalı sayıları hem eğitim aşamasında hemde test aşamasında yapay sinir ağlarında kullanımına olanak verecektir.

5.2 Çözüm Yöntemi

VHDL ile yazılmış kayan noktalı sayılara ait algoritmalar uygun YSA mimarisine uygun şekilde yerleştirilerek bir YSA gerçekleştirilebilir. Eğer sayı duyarlılığı 8 veya 16 bit alınırsa alan bakımından fazla problem çıkmayacaktır. 8 bit veya 16 bit sayı duyarlılığı ile sınırlı miktarda paralele işleyebilen YSH'si FPGA'içine yerleştirilebilir. Fakat 8 bit veya 16 bit gerçek uygulamalar yeterli hassasiyeti yakalayamayabilirler. YSA ile gerçek bir uygulama yapılmak isteniyorsa veya YSA eğitimininde FPGA üzerinde yapılması isteniyorsa kesinlikle 32 bit kayan noktalı sayı duyarlılığı kullanılmak zorundadır. 32 bit sayı duyarlılığı kullanımı FPGA'in kısıtlı kaynakları sebebiyle YSA oldukça zordur.

32 bit sayı duyarlılığı FPGA üzerinde YSA mimarisinde kullanılmak isteniyorsa çözümlerden bir taneside YSA mimarinin paralelliğinden ödün vermektir. Bu

çalışmada, FPGA üzerinde, toplama, çarpma gibi aritmetik işlemleri seri fakat diğer bütün işlemleri paralel yapabilen bir YSA mimarisi gerçeklenmiştir. Melez (paralel ve seri) mimari ile YSA paralel işlem yapabilme gücü paralel işlem yapabilen FPGA sayesinde kurunmuş ve 32 bit duyarlılık sayılarla işlem yapabilme yeteneği YSA'na kazandırılmıştır.

5.2.1 Sayısal test ve karşılaştırma

Fpga tabanlı YSA'ları diğer işlemcilerle gerçeklenen YSA'larıyla karşılaştırıldığında ilk göze çarpan FPGA üzerinde gerçeklenen mimarinin çok amaçlı hesaplamalar yapabilmesidir. Oysaki PIII Cpu üzerinde gerçekleştirilen genel amaçlı bir YSA sonsuz büyüklükte olacaktır.

Aşağıda, gerçeklenen YSA'nın çalışması bir sayısal bir örnekle açıklanmıştır. Bu anlatım sadece bir katman içindir. Üç katman için yapılacak işlem ise yapının üç kez dönmesini sağlamak olacaktır. Bu uygulama için sistemin giriş değerlerine ek olarak 2:3:1 (ağın kaç katmandan oluştuğunu ve katmanlardaki YSH sayısı) değerleri de sisteme girilmelidir. Bu değerler sisteme teker teker girilir ve bu değerler sistem kaç kez döneceğini ve her katmanın özelliğini nasıl ayarlayabileceğini belirler.

1. Ağa giriş değerleri verilir.

g1 = 0.5 ve g2 = -0.25

2. g1 ve g2 değerlerinin kayan noktalı karşılıkları hesaplanır.

g1 = 01000000000000000000000000000000

g2 = 11000000100000000000000000000000

3. İlk olarak YSH sayısı sisteme girilir ve başla butonuna basılıp girişler teker teker girilir. Bitir btonuna basıldığında sistem giris sayısını 2 olarak belirler. Bitir butonuna basılmasıyla birlikte g1 ve WROM daki g1 ile alakalı ağırlıklar (w111, w121, w131) ÇRPROM1'e alınır.Toplam kullanılacak ağırlık sayısı giriş sayısı ile

YSH sayısının çarpımıyla bulunur. ÇRPROM1'deki g1 girişi ve ilgili ağırlıklar teker teker ÇARPMA(çarpılarak)modülünden geçirilerek ÇRPROM2'ye yazılırlar.

çrp1 = g1 x w111 = 0.5 x 1 = 0.5 = 01000000000000000000000000000000

çrp2 = g1 x w121 = 0.5 x 0.5 = 0.25 = 01000000100000000000000000000000

çrp3 = g1 x w131 = 0.5 x 1 = 0.5 = 01000000000000000000000000000000

4. ÇRPROM1'deki ağırlıklar çarpıldıktan sonra aynı işlemler (3. nolu işlem) ikinci giriş için tekrarlanır.

çrp4 = g2 x w112 = -0.25 x 0.5 = -0.125 = 10111110000000000000000000000000

çrp5 = g2 x w122 = -0.25 x 1 = -0.25 = 11000000100000000000000000000000

çrp6 = g2 x w132 = -0.25 x 0.5 = -0.125 = 10111110000000000000000000000000

5. Bu çarpımlar(çrp1...çrp6) farklı YSH'lerine ait girişler oldukları için ÇRPROM2'de ait oldukları YSH ait alana yazılırlar. Sonradan aynı YSH2ne ait çarpımlar bias(uygulamadaki değeri 1 'dir.) ve kendi aralarında toplanır (TOPLAMA elemanında geçirilir).

tlp1 =bias+çrp1+çprp4=1+0.5-0.125 =1.375= 00111111101100000000000000000000

tlp2 = bias+çrp2+çrp5 = 1+ 0.25- 0.25 = 1 = 00111111100000000000000000000000

tlp3 =bias+çrp3+çprp6=1+0.5-0.125=1.375 = 00111111101100000000000000000000

6. Toplama elemanından teker teker çıkan bu toplamalar RAM de saklı olan sigmoid fonksiyonunda geçirilerek ÇKBFR saklayıcısına yerleştirilirler. ÇKBFR daki bu değerler bir sonraki katman için giriş değerleridir artık onun için g1, g2 ve g3 olarak yazılacaklardır

g1(sigmoid(tlp1)) = 0.768525 = 01001001001110111010000011010000

g2(sigmoid(tlp2)) = 0.731059 = 01001001001100100111101100110000

g3(sigmoid(tlp3)) = 0.768525 = 01001001001110111010000011010000

7. ÇKBFR da değerler depolandığı anda diğer katman için sistem tarafından başla,bitir sinyallari tekrar dan üretilir. Yeni girişler sisteme verilmeden gereken birimler resetlenir.

Tablo 4.2 'de FPGA gerçeklenen sistem çıkışı ile YSA nın çıkışı ve aralarındaki hata oranı gösterilmiştir.

Tablo 5.1 YSA sisteminin simülasyon sonuçlarının karşılaştırılması

	YSA	TD-YSA	HATA
G1	2.3274321322	2.268109	0.059323
G2	1.1637160661	1.1340545	0.029661

5.3 Sonuç

Fpga mimarisindeki hızlı gelişme rağmen 32 bit IEEE kayan noktalı sayılar ile FPGA tabanlı tamamen paralele çalışabilen YSA'nın gerçeklenmesi oldukça zordur. Bu sebepten dolayı gerçeklenen YSA ya 8 bit veya 16 bit sayı duyarlılığı düzeyinde kalmaktadır. Buna rağmen çeşitli çözümler kullanılarak (paralel ve seri çalışabilen YSA, tek bir YSA'nı gerçekleyen birden fazla paralel çalışabilen FPGA v.b.) 32 bit sayı duyarlılığı ile çallışan YSA'ları FPGA'ler üzerinde gerçekleştirilmektedir.

Tezde gerçeklenen YSA'ya ait VHDL kodları CD-ROM'da ek olarak verilmiştir.

KAYNAKLAR

1. ARABIB, M.A. 1987. Brains Machines and Mathematics. 2nd edition, Springer-Verlag, New York.

2. BLAKE, J.J., MAGUIRE, L.P., MCGINNITY, T.M, ROCHE, B., MCDAID L.J. 1998. The Implement of Fuzzy Systems, Neural Networks using FPGAs, Infomation Sciences, Vol.112, pp 151-168

3. BROWN, S.D., FRANCIS, R.J., VRANESIC Z.G. 1992. Field Programmable Gate Arrays, Kluwer Academics Publishers

4. BURR , J. 1993. Digital Neurochip Dsign in Parallel Digital Implementations of Neural Networks, Prentice Hall Inc., pp 223-1214.

5. CHEN, C. H. 1996. Fuzzy Logic and Neural Network Handbook, IEEE Press, Vol.9, pp 9.17-9.20.

6. CHUCLAND, P.S., SEJNOWSKI, T.J. 1992. The Computational Brain , MIT Press, Cambridge, London..

7. CMPTON, K., HAOUCK, S. 2002. Reconfigurable Computing : A Survey of System and Software, ACM Computing Surveys.

8. COX, C., BLANZ, W. 1992. GABGLION-A Fast Field Programmable Gate Array Implementation of a Connectionist Classifier, IEEE Journal of Solid-satate Circuits, Vol.27, No.3, pp 288-299.

9. EFE M.Ö., KAYNAK O. 1999. A Coparative Study of Neural Network Structure in Identification of Nonlinear Systems, Mechatronics, Vol.9, No.3, pp 287-300.

10. GOSLIN, G.R. 1995. Using Xilinx FPGA's to design custom digital processing devices, Proceedings of DSPX, pp 565-604.

11. GROSSBERG, S. 1988. Neural Network and Neural Intelligence, MIT Press, Cambridge, London.

12. GUCCIONE, S. A., GONZALEZ, M. J. 1993. A Neural Network Implementation Using Reconfigurable Architectures, Will Moore and Wayne Luk, Abingdon EE&CS Books, Abingdon, England

13. HADLEY, J. D., HUTCHINGS, B. L. 1995. Desing Methologies for Partially Reconfigured Systems, Proceeding of the IEEE Workshop on FPGAs for Costum Computing Machines, Los Alamitos, California, pp 78-84.

14. HARTENSTEIN, R., HERTZ, M., HOFFMANN, T., NAGELDINGER,U. 2000. Generation of Design Suggestion for Coarse-Grain Reconfigurable Architecture, 10th International Workshop on Field Programmable Logic and Applications.

15. HAYKIN, S. 1999. Neural Networks A Comprehensive Foundation. 2nd edition, Prentice Hall Publishing, New Jersey 07458, USA, Vol.1, pp 6-7.

16. HAYKIN, S. 1994. Neural Networks A Comprehensive Foundation, Prentice Hall Publishing, New Jersey , USA, Vol.1, pp 1-14.

17. HUTCHINGS, B.L., WIRTLIN, M.J. 1995. Implementation Approaches fo Reconfigurable Logic Applications, 5th International Workshop on Filed Programable Logic and Applications, Oxford, England, pp 419-428.

18. IEEE Task P754. 1981. A Proposed Standard for Binary Floating-Point Arithmetic, IEEE Computer, Vol.14, No.12, pp 51-62.

19. KRIPS, M., LAMMERT, T., KUMMERT A. 2002. FPGA Implementation of Neural Netwprk for Real-Time IEEE Internation Workshop on Electronic Design, Test and Applications

20. LISA, F., CARRABINA, J., VICENTE, C., Avellana N., VALDERRAMA, E. 1993. Two-bit Weights Are Enough to Solve Vechicle License Recognition Problem, Proc. IEEE International Conference Neural Networks, pp 1242-1246.

21. MEAD, C.A. 1989. Analog VLSI and Neural Systems, Reading, Addision-Wesley

22. MINSKY, M., PAPERT, S. 1969. Perceptrons., MIT Press.

23. MOON, S., HWANG, J.N. 1997. Robuts Speech Recognition Based on Joint Model and Feature Space Optimization of Hidden Markov Models, IEEE Transaction on Neural Networks, Vol.8, No.2, pp 194-2004

24. NARENDRA, K. S., PARTHASARATHY K. 1990. Identification and Control of Dynamical Systems Using Neural Networks, IEEE Transactions on Neural Networks, Vol.1, No.1, pp 4-27.

25. NICHOLS, K.R., MOUSSA, M.A, AREIBI, S.M. 2002. 15th International Conference on Computer Applications in Industry and Engineering (CAINE-2002), San Diego, California USA.

26. NUR, T. 2000. Tekrar Düzenlenebilir İşlem KatıTasarımı, Yüksek Lisans Tezi, İ.T.Ü. Fen Bilimleri Enstitüsü, İstanbul.

27. OSSOINIG, H., REISINGER E., STEGER C., WEİSS R. 1996. Design and FPGA-Implementation of a Neural Network, Proceedings of the 7th International Conference on signal Processing Applications & Technology, pp 939-943, Boston, USA.

28. ÖZTEMEL, E. 2003. Yapay Sinir Ağları, Papatya Yayıncılık, İstanbul, pp 37.

29. PICHE, S.W. 1994. Steepest Descent Algorithms for Neural Network Controllers and Filters, IEEE Transactions on Neural Networks, Vol.5, No.2, pp 198-212.

30. RUCKET, U., FUNKE, A. and PINTASKE, C. 1993 Acceleratorboard for Neural Associative Memories, Neurocomputing, Vol.5, No.1, pp 39-49.

31. RUMELHART, D.E., HINTON, G.E, WILLIAMS, R.J. 1986. Learninig representations by backpropogation errors, Nature, Vol.323, pp 533-536.

32. SHARDA R. 1994. Neural Networks for The MS/OR Analyst: An Application Bibliography, Interfaces, Vol. 24, No.2, pp 116-130.

33. STANLEY, M., PARTICIA, L. 1996. A Guide to VHDL, by Kluwer Academic Publishers.

34. TESHNEHLAB, M., WANTANABE K. 1999. Intelligent Control Based on Flexible Neural Networks, Kluwer AcademicPublishers, Dordhrecht, Vol.2, pp 25-37.

35. TRIMBERGER, S.M. 1994. Field programable Gate Arrays, Kluwer Academic Publihers, New York.

36. VENKATESAN, R. 1994. FPGA Implementation of Residue Number System, Master Thesis, University of Windsor. Dept. of E.E.

37. VILLASENOR, J., WILLIAM, H.M. 1997. Reconfigurable Computing, Scirntific American.

38. WILLERT, C. 2000. The Evolution of Programable Logic Design Technology, Xilinx Inc.

39. YU, X., DENI D. 1994. Implementing Neural Networks In FPGAs, The Institution of Electrical Engineers, IEE published, Savoy Place, London WC2R 0BL, UK.

40. ZHU, J., GUNTHER, B.K.,1999. Towards an FPGA Based Reconfigurable Computing Environment for Neural Network Implementations, Proceedings of the Ninth International Conference on Artificial Neural Networks (ICANN'99). In IEE Conference Proceedings 470, pp.661-667, IEE.

ÖZGEÇMİŞ

1977 yılında Erzincan'da doğdu. İlk, orta ve lise öğrenimini Erzincan'da tamamladı. 1095 yılında girdiği Kocaeli Üniversitesi Elektronik ve Haberleşme Mühendisliği Bölümün'den 2000 yılında mezun oldu. Halen Kocaeli Üniversitesi, Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümün'de Öğretim Görevlisi olarak görev yapmakta olup; evlidir.