YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

ASENKRON MOTORUN DSP (SAYISAL İŞARET İŞLEYİCİ) TABANLI BİR KONTROL SİSTEMİ KULLANILARAK YSA (YAPAY SİNİR AĞLARI) İLE PERFORMANSININ ARTTIRILMASI

Elk. Yük. Müh. Kayhan GÜLEZ

F.B.E. Elektrik Mühendisliği Anabilim Dalında Hazırlanan

DOKTORA TEZİ

Tez Savunma Tarihi : **10 Haziran 1999**
Tez Danışmanı : **Prof. Dr. Halit PASTACI (YTÜ)**
Jüri Üyeleri : **Prof. Dr. Sezgin ALSAN (MÜ)**
: **Doç. Dr. Mehmet KORÜREK (İTÜ)**

İSTANBUL, 1999

İÇİNDEKİLER

Sayfa

SİMGE LİSTESİ

1., 2., 3. ve 6. Bölümler için simgeler

s, r	Stator ve rotoru ifade eden indisler
d, q	Rotor akısı ile senkron olarak dönen eksenler
α, β	Stator sabit eksenler
I	Akım
I_0	Mıknatıslanma akımı
P	Güç
T	Moment genel gösterilim
T_M	Maksimum moment
T_B	Kilitli-rotor momenti
T_{TM}	Nominal tam yük momenti
T_e, T_d	Elektriki moment, gürültü momenti
sa, sb, sc	Stator a, b, c fazları
i_{sa}, i_{sb}, i_{sc}	Stator a, b, c faz akımları
i_{sd}, i_{sq}	Stator d ve q eksen akımları
$i_{s\alpha}$, $i_{s\beta}$	α ve β eksen akımları
v	Gerilim
v_a, v_b, v_c	a, b, c faz gerilimleri
ψ_r	Rotor akısı
ψ_{rd}, ψ_{rq}	Rotor d ve q eksen akıları
w	Rotor açısal hızı
w_s	Senkron açısal hızı
w_e, w_m, w_r	Rotor akı, rotor elektriki, rotor şaft açısal hızı
w_{sl}	Kayma açısal hızı
w_d	Doğal frekans
θ	Rotor akı açısı
θ_r, θ_m	Rotor elektriki, rotor şaft açısı
ϕ	Güç faktörü açısı
n	Rotor hızı genel gösterilim
n_s	Senkron hız
n_r	Rotor hızı
n_{max}	Kontrol edilebilir maksimum hız
R_e	Eşdeğer direnç
R_s	Stator direnci
R_r	Rotor direnci
L_s	Stator endüktansı
L_r	Rotor endüktansı
L_m	Karşılıklı endüktans
s	Kayma
p	Kutup sayısı
τ_r	Rotor zaman sabiti
J	Eşdeğer şaft atalet momenti katsayısı
B	Sürtünme veya sönüm katsayısı
σ	Toplam kaçak faktörü
ξ	Sönüm faktörü
K_t	Eşdeğer moment sabiti
K_p	Oransal sabit
K_i	Integrasyon sabiti
K	PWM darbelerinde azaltma yapan bölme kazancı

T	Inverter çalışma sürelerinin bir peryodu
t_c	Cevap zamanı
t_{top}	Algoritmanın tamamlanma zamanı
ss	Enkoder slot sayısı
m	İkili dijit sayısı

4. Bölüm için simgeler

w	YSA ağırlıkları
α	Öğrenme oranı
ε	Momentum oranı
Δ	Genelleştirilmiş Delta kuralını simgeler
t	Öğrenme çevrimlerinin sayısı

KISALTMA LİSTESİ

ADC	Analog Dijital Dönüştürücü
DAC	Dijital Analog Dönüştürücü
DSP	Digital Signal Processing (Sayısal İşaret İşleme)
IEEE	Institute of Electrical and Electronics Engineers
İ.T.Ü.	İstanbul Teknik Üniversitesi
Y.T.Ü.	Yıldız Teknik Üniversitesi
YSA	Yapay Sinir Ağları
ti	Texas Instruments
AC	Alternatif Akım
DC	Doğru Akım
PWM	Pulse With Modulation
UVPWM	Uzay Vektör Pulse With Modulation
TMS320C50	Texas Instruments Firmasının C50 DSP'sinin piyasa kodu
AAD	Analog Arabirim Devresi
TLC320C40	Texas Instruments Firmasının C40 AAD piyasa kodu
PI	Oransal ve İntegrasyon kontrolörü
PID	Oransal, İntegrasyon ve Türev'e dayalı kontrolör
VLSI	Very Large Scale Integration
BJT	Bipolar Transistör
GTO	Gate Turn-off Transistör
IGBT	Izolated Gate Bipolar Transistör

ÖNSÖZ

Bu tezin hızlı bir şekilde ilerlemesi için gerekli kolaylığı gösteren Y.T.Ü. Elektrik-Elektronik Fakültesi, Elektrik Mühendisliği Bölümü, Kontrol ve Kumanda Sistemleri Anabilim Dalı Başkanı Hocam Prof. Dr. Sayın Halit PASTACI'ya teşekkürlerimi sunarım.

Karşılaşılan malzeme eksikliklerinin giderilmesinde ve teknik desteğin verilmesinde her türlü yardımı sağlayan Texas Instruments Firması'nın Avrupa Şubesi'ne teşekkürü bir borç bilirim.

Yapılan ölçümlerin karşılaştırmalı analizinin sunulabilmesi için benzer ölçümlerin alınmasında her türlü kolaylığı gösteren ABB Firması Kartal Fabrikası İnverter Teknik Servis Sorumlusu Elektrik Mühendisi Sayın Sabri DEMİRYÜREK'e ,

Deneysel çalışmalarda DSP program yapısı için gerekli olan asenkron motorun parametrelerinin elde edilmesinde değerli yardımlarını esirgemeyen Türk Elektrik Motorları A.Ş. İşletme Müdürü Elektrik Yük. Mühendisi Sayın Raif ALTUMN'a teşekkürü bir borç bilirim.

Yapılan deneylerde, donanım ve yazılımların oluşturulmasında yardımlarını esirgemeyen kıymetli arkadaşlarım Elektronik Yük. Mühendisi Sayın C. Bülent FİDAN'a, Elektronik Yük. Mühendisi Sayın S. Vakkas ÜSTÜN'e, Elektronik Yük. Mühendisi Sayın Turgay TEMEL'e, Elektronik Yük. Mühendisi Sayın Serkan AYDIN'a teşekkürü bir borç bilirim.

Teorik bilgilerin toplanması ve belirli bir sistematiğe oturtulmasında her türlü yardımı sağlayan Elektrik Mühendisi Sayın İbrahim B. KÜÇÜKDEMİRAL'a, Elektrik Yük. Mühendisi Sayın Recep YUMURTACI'ya teşekkürlerimi sunarım.

ÖZET

Yapay Sinir Ağları (YSA) günümüz teknolojisinde kontrol, elektrik makinalarında arızaların erken tanısı, sayısal işaret işleme gibi farklı birçok alanlarda başarı ile kullanılmaktadır. YSA gibi paralel işlem yapma kabiliyetine sahip böyle bir sistemden bağımsız olarak Sayısal İşaret İşleyiciler de (DSP-Digital Signal Processors) sistem kontrolü, haberleşme, digital motor kontrolü ve askeri projelerin bir çoğunda kendisine uygulama alanı bulmuştur. YSA'nın matematiksel formüllerden bağımsızlığı ve sonuca etkili bir şekilde çok kısa bir zaman dilimi içinde yaklaşması, DSP'nin de YSA'ya işlemesi için gerekli verileri oldukça hızlı bir şekilde (nano saniyeler mertebesinde) sağlaması, böyle bir asenkron motor kontrol -performansı arttırıcı diğer bir etken olan vektör kontrolü tercih edilmiştir- uygulamasında, prototip olarak ele alınan sistemin performansını arttırıcı her türlü özelliğe sahip olmasından dolayı, diğer yaklaşım metodlarıyla benzer bir sisteme yapılacak kontrole göre bunu çok daha fazla avantajlı ve güvenilir kılmaktadır. Ayrıca sistemde, DSP üzerinden alınan sayısal değerler YSA ile güncelleştirilmekte ve bu güncelleştirilmiş eğitme ağırlıkları DSP tarafından tekrar sistemin performansını arttırmak için kullnılmaktadır. Bu da benzer sistemlere göre kullanılan sistemin diğer bir üstün yanını temsil etmektedir.

Yapılan çalışmada, asenkron motorun performansının arttırılması işlemi DSP tabanlı bir kontrol sistemi ile gerçekleştirilirken YSA bünyesinde İleri-besleme ağı ve Hızlı Hatanın Geriyeyayılımı Algoritması (Fast Error Backpropagation Algorithm) kullanılmıştır. Konunun akış yapısına bağlı kalınarak kontrol işleminin gerçeklenmesinde kullanılan asenkron motor, vektör kontrolü, YSA ve DSP gibi konuların her biri ayrı başlık altında incelemeye alınmış olup, sonuç bölümünde ise alınan deney ölçümlerine göre kontrol işleminin nasıl yapıldığı blok diyagram şeklindeki devre şemaları, ölçülen değişkenlerin zamanla değişim diyagramları ve ağ mimarileriyle gösterilmekte, YSA'yı eğitme de kullanılan sayısal bilgi dosyası, sonuç tabloları şeklinde verilmekte ve kullanılan program yapıları ve açık devre şemaları ise ekler bölümünde sunulmaktadır.

Kontrol sisteminden alınan sayısal değerler klasik Hatanın Geriyeyayılımı Algoritması'na (Error Backpropagation Algorithm) da uygulanmış olup her iki algoritmanın teorik anlatımı YSA bölümü altında, elde edilen sonuçların Hızlı Hatanın Geriyeyayılımı Algoritması ile karşılaştırmalı analizi ise sonuç bölümünde verilmektedir. Yine sonuç bölümünde, alınan bu sayısal değerlerdeki değişkenlerin yüksek dereceli durumlarına bağlı olarak tasarlanan yüksek mertebeden ağ sonuçları da sunulmaktadır.

Motor olarak, 0.55 kW, 2.6A, 220V, 50 Hz, Cosϕ=0.79 olan bir 3-fazlı sincap kafesli asenkron motor kullanılmıştır.

Sistemin performansını arttırmak için YSA'ya giriş değerleri olarak hem klasik Hatanın Geriyeyayılımı Algoritması hem de Hızlı Hatanın Geriyeyayılımı Algoritması için motorun rotor hızı (n) açısal hız şeklinde (w), akımı (I) ve gücü (P), çıkış değeri olarak momenti (T) girilerek 300000 iterasyon sonucunda klasik Hatanın Geriyeyayılımı Algoritması için %0.0011, Hızlı Hatanın Geriyeyayılımı Algoritması için %0.0007 ve yüksek mertebeden YSA'da, klasik Hatanın Geriyeyayılımı Algoritması için %0.0009 ve Hızlı Hatanın Geriyeyayılımı Algoritması için %0.0004 gibi çok az bir sistem hatasıyla motorun istenilen yük-momentine istenilen sürede (nano saniyeler mertebesinde) ulaştığı görülmektedir.

Elde edilen sonuçların test edilmesi yani karşılaştırmalı analizinin yapılabilmesi açısından, ABB firmasının Kartal Fabrikası İnverter Teknik Servisi'nde de gelişmiş bir İnverter modülü olan ACS600 üzerinden benzer ölçümler yapılmış olup, YSA'ya uygulanan değişkenler aynı kalmak

üzere, klasik Hatanın Geriyeyayılımı Algoritması için %0.0012 ve Hızlı Hatanın Geriyeyayılımı Algoritması için %0.0003 ve yüksek mertebeden YSA'da, klasik Hatanın Geriyeyayılımı Algoritması için %0.0010 ve Hızlı Hatanın Geriyeyayılımı Algoritması için %0.0003 olarak benzer şekilde çok az bir sistem hatasıyla motorun istenilen yük-momentine istenilen sürede (nano saniyeler mertebesinde) ulaştığı görülmüştür. Bu da yapılan çalışmdan alınan ölçümlerin doğruluğunu göstermektedir.

DSP'nin veri haberleşmesinin sağlanması ve YSA'nın eğitme işleminin yapılması elde edilen sonuçların YSA test fazı üzerinden tekrar sisteme aktarılması için bir Pentium-200 MMX bilgisayar kullanılmıştır. DSP işlemci olarak Texas Instrument firmasının TMS320C50 kartı ile uygulama yapılmıştır. YSA ve DSP algoritmalarının çalıştırılması için YSA'da C++ ve DSP'de Assembler dilleri ile yazılımlar gerçekleştirilmiştir.

Anahtar Kelimeler: Asenkron Motor, Vektör Kontrolü, Sayısal İşaret İşleme, Yapay Sinir Ağları ile Kontrol, Analog-Dijital Dönüştürücü, Sistem Otomasyonu.

ABSTRACT

Artificial Neural Networks (ANN) are successfully used in a lot of areas such as control, early detection of electric machine faults, digital signal processing in our daily technology. Digital signal processors (DSP) spreadly found plenty of application areas like system control, communication, digital motor control and military projects as they are independent on the parallel data processing system of ANN. It is a very preference reason ANN's mathematical independence and their effectiveness in a very short time to approaching to the results, and also DSP ensures the necessary data (nano second degree) to process to ANN faster than analog control systems. These results give a few but important advantages and do the system more reliable according to the other approaching methods. Because it has such a prototype system every feature to increase induction motor performance -vector control applicated to the system as another effect to increase the performance of the motor- it can be used for all rotating machine applications. In the system, the values taken from DSP are also updated by ANN. The updated training weights are again used to increase the performance of the system. This shows another advantage of the system according to other similar systems.

In the study, it is used a DSP-based control system with feed-forward ANN (multilayer perceptron) and Fast Backpropagation Algorithm to increase the performance of the induction motor. It is firstly explained induction motors, vector control, ANN and DSP under separate chapters to follow the subject flow respectively. It is shown the circuits of the system how to be done and how to work with block circuit diagrams, measured variable ones and network architectures. The training set of ANN is given as data file tables according to the experimental results in the last chapter. The softwares of ANN and DSP and complete circuit diagrams are given in addition parts.

The values taken from the control system were also applicated to classic Error Backpropagation Algorithm. The theoretical description of this two algorithms are given in ANN chapter and it is shown a comparison of the algorithms in the matter of experimental results in the last chapter. It is put forward the results of high-order network according to the high-order degree conditions of the same values in the same chapter.

It was used 0.55 kW, 2.6A, 220V, 50Hz, Cosϕ=0.79 3-phase squirral-cage induction motor.

In the thesis, for both clasic Error Backpropagation and Fast Error Backpropagation Algorithms, as the input values to ANN, rotor speed (n) as angular speed (w), current (I) and power (P), as the output value to it torque (T) of the motor to increase the performance of the system are made and discriminated with success rate under %0.01 generalised system error, %0.0011 for classic Error Backpropagation Algorithm, %0.0007 for Fast Error Backpropagation Algorithm and to high-order network, %0.0009 for classic Error Backpropagation Algorithm and %0.0004 for Fast Error Backpropagation Algorithm for 300000 iterations. It is considerably important success to access this performance increment, that is, under changing the value of the load, the load-torque rises to the same value performance in a very short time(nano seconds degree).

It was done similar measurements by using an advanced technology ACS600 Inverter module in Inverter Technical Service of ABB's Kartal Factory in the matter of testing our experimental results. Input and output variables of ANN are the same ones. Again, the performance of the system is increased and discriminated with success rate under %0.01 generalised system error, %0.0012 for classic Error Backpropagation Algorithm, %0.0003 for Fast Error Backpropagation Algorithm and to high-order network, %0.0010 for classic Error

Backpropagation Algorithm and %0.0003 for Fast Error Backpropagation Algorithm for 300000 iterations. These results show the truth of our measurements.

It was used a Pentium-200 MMX computer to ensure the data communication of DSP, train ANN set and give the taken ANN test phase results to the system again. The system application was done by Texas Instruments TMS320C50 DSP kit. The software languages of ANN and DSP are C++ and Assembler respectively.

Key words: Induction Motor, Vector Control, Digital Signal Processing, Control with Artificial Neural Networks, Analog-Digital Converter, System Automation.

1. GİRİŞ

1.1 Konunun Tanıtılması

Yapay Sinir Ağları (YSA) ile tasarım teknolojisi başarılı bir şekilde hata tespiti (fault detection) (Mangum, 1990; Chow, 1991, 1993; Gülez, 1995), kontrol ve işaret işleme (signal processing) gibi birçok farklı alanlarda kullanılmaktadır. YSA'nın ilk kullanımı 1940'lara kadar götürülebilir. Ancak, YSA çalışmalarında teorinin yanında gerçek zaman uygulamaları açısından son yıllarda ciddi ilerlemeler kaydedilmiştir. Yapılan çalışmalarda kullanılan modellerden biri ileri-besleme (feed-forward multilayer perceptron) ağıdır. Kohonen ağı, Hamming ağı, Hopfield ağı, Radyal tabanlı ağ v.b. çalışmalar diğer ağ modellerine iyi birer örnek teşkil etmektedir. Yapılan çalışma, ileri-besleme ağ yöntemi ve Sayısal İşaret İşleyici (DSP-Digital Signal Processor) kullanımından hareketle asenkron motorun performansının arttırılmasını sağlamak ve değişken yük altında en iyi momenti en kısa zamanda yakalamak için YSA ve DSP teknolojisinin niçin ve nasıl uygulandığını anlatmağa çalışmaktadır.

Burada, asenkron motorun sistem kontroluna dayalı performansının arttırılması işleminde olduğu gibi belirli kabuller çerçevesinde geliştirilen sistemin tüm döner makinaları kapsam dahilinde tutarak, uygulamanın gerçekleştirilmiş olması fabrika otomasyonu açısından da önemli bir gelişmedir. Yapılan çalışmalar başlı başına üzerinde çalışılan konuda bir problem teşkil etmekte iken, bu problemlerin herhangi birinin çözümüne YSA ile yapılacak iterasyonel bir yaklaşım çok fazla iterasyon sayısı içermesine rağmen çok kısa sürede ve verilen girişlere çok düşük hata payıyla cevap vermesinden dolayı -doğruluğunun yüksek olmasından dolayı- en iyi çalışma tekniklerinden biri olarak kabul edilmektedir.

Temeli 1800 yıllarına dayanan Otomatik Kontrol Teorisi yüzyılın başında gerçekleşen ve insanlık tarihinde yeni bir dönem açan endüstriyel devrimin önemli konularından birisi olup, 1950 ve 1960 yıllarındaki uzay çalışmalarında önemli rol oynamış ve günümüzde elektrik-elektronik mühendisliğinin anabilim dallarından biri olmuştur. Bu teori zamanla elektrik-elektronik mühendisliğinin de dışına taşmış ve son yıllarda elektronik alanında meydana gelen baş döndürücü gelişmeler sayesinde, ekonomi, biyoteknik ve biyomedikal konularında da uygulanır hale gelmiştir. Günümüzde Otomatik Kontrol Teorisi'nin uygulama alanlarından bazıları; füzeler, uydular, robotlar, uzay araçları, uçak ve gemilerde bulunan otomatik pilotlardır. Bütün bu gelişmelerden motor kontrolu da ve dolayısıyla fabrika otomasyonu da gerekli payı almıştır.

Otomatik Kontrol Teorisi temel olarak ideal model ile eldeki mevcut sistemin karşılaştırılmasına dayanmaktadır. Mevcut sistem -bu bir robot olabileceği gibi, motorun kontrol üniteside olabilir- ve ideal sisteme aynı girdiler verilir. Mevcut sistemin girişe olan tepkisi, ideal sistemin aynı olaya karşı olan tepkisi ile karşılaştırılır. Neticede de hata bulunmuş olur. Yani:

Hata= Model - Mevcut Sistem (1.1)

Hatayı düzeltmek için mevcut sistemin içindeki parametreler belirli bir oranda değiştirilir ve tekrar ideal sistem ile karşılaştırma yapılır. Eğer hala bir hata var ise, hatanın miktarına göre tekrar sistemin parametreleri değiştirilir. Böylece hata gittikçe düşürülerek mevcut sistemin ideal sisteme benzemesi sağlanmış olur. Burada dikkati çeken nokta, mevcut sistemi olduğu gibi kabul edip çeşitli algoritmalar ile iç parametreleri uygun şekillerde değiştirerek sistemin kalitesini arttırmaktır.

Sistemin teorisi açısından durum böyle iken mikroişlemcilerin kontrol alanına girmesi, kontrol sistemlerine yeni bir boyut kazandırmıştır. Analog denetleyici yapıları ile kontrol yapılırken, sistemin belirli koşulları sağlama zorunluluğu, dijital yapılarla aşılmıştır. Özellikle değişken çevre ve çalışma koşullarında analog devrelerin yetersiz kalması, dijital yapıya yönelmeyi hızlandırmıştır. Kontrol algoritmalarının karmaşık bir yapıya bürünmesi, analog devreler ile gerçekleştirilmesini imkansızlaştırmıştır. Gerçek zamanda işlem yapan algoritmaların motor kontroluna uygulanmaları, sistemlerin beklentilerini arttırmış, daha az enerji ile daha büyük sistemlerin kontrolunun gerçekleştirilmesi arzulanmıştır.

Motor kontrol sistemlerinde kapalı çevrim kontrolu ile tam kontrol sağlandıktan sonra sistemden beklentiler artmıştır. Örneğin motorun sadece hız kontrolu yapmasıyla aynı zamanda akım ve moment kontrolunu da gerçekleştirmesi sağlanmış olup (Dote, 1988, 1990; Vainio vd., 1992; Bose, 1994; Lorenz vd., 1994; Kung vd., 1995; Lai vd., 1996; Tzou, 1996; Attaianese vd., 1997), klasik kontrol yöntemleriyle kontrol yerine de bu çalışma da olduğu gibi YSA ile kompleks yapıların oluşturulması sonucunda, kontrol bütün sistemin en önemli parçası haline gelmiştir.

Sistemin kontrol olayında en zayıf noktası kabul edilen belirli bir hata ve gecikme payı, belirli kabuller çerçevesinde son yıllarda üretilen sistemlerle, -özellikle 2000 'li yılların eşiğinde, elektronik alanında yaşanan gelişmeler ile yeni bir dönemin başlangıcındaki insanlık- ihmal edilebilecek değerlere çekilmiştir.

Ses ve görüntü uygulamaları ile karşımıza çıkan DSP'ler kontrol sistemlerine yeni bir bakış açısı kazandırmıştır. İşlem yapma kapasiteleri, üstün matematik işlem hacmi ve en önemlisi işleme hızı (TMS320C50 için 25-30 ns civarında), DSP'nin kullanım alanının hızla genişlemesini sağlamıştır. Klasik mikroişlemcilerden mimarisi farklı olan DSP sistemleri, matematik işlemleri donanım olarak gerçekleştirerek, üstün bir hız özelliği kazandırmıştır. Bütün bu özelliklerin kontrol uygulamalarında kullanılmaya başlanması, dijital sistemlerin vazgeçilmez bir denetleyici olmasını sağlamıştır. Analog denetleyicilerin üstün olduğu alanlarda, aynı özellikleri sağlayarak tam bir üstünlük kazanmışlar ve kontrol edilecek sistemin gerçek zamanda kontrolunu gerçekleştirmişlerdir. Dijital kontrol sistemlerinde matematiksel olarak en çok kullanılan yapı, çarpma ve toplama yapısıdır. Mikroişlemcilerin toplama işlemini donanım olarak gerçekleştirmesine rağmen, çarpma işlemi kullanıcıya gözükmeyen bir yazılım ile yapılmaktadır. Bu da, sistemin toplam işleyiş süresini artmasına yol açmaktadır. DSP sistemlerinde, işlem yapan ünitelerin birbirinden bağımsız olması ve bellek birimlerinin ayrı olması, işlemlerin donanım olarak gerçekleştirilmesini, yani bir saat çevriminde tamamlanmasını sağlamaktadır. DSP'nin üstün hız özelliği bu şekilde ortaya çıkmaktadır.

DSP sistemlerinin kontrol alanında uygulanmaya başlaması, diğer alanlara göre geç olmuştur. Bunun nedeni, tasarlanan ve uygulamaya aktarılan sistemlerde maliyet faktörünün hala en önemli etken olmasıdır. Mikroişlemcilerin yakın zamana kadar daha ucuz olmasına rağmen, yarıileken teknolojisindeki son gelişmeler, DSP'nin de aynı oranda ucuzlamasına neden olmuştur. Bu aşamadan sonra, DSP ile tasarlanan kontrol sistemlerinin daha çok uygulama alanı bulacağı ve yavaş yavaş büyük firmaların otomasyon işlemlerini DSP üzerine yönlendireceği kaçınılmaz bir gerçektir.

Elektrik makina uygulamaları genelde değişken hızlı fanları, kompresör veya pompaları içerse de son zamanlarda fabrika (üretim için gerekli teknolojik yenileme) otomasyonundaki artışa bağlı olarak servo uygulamaları da büyük önem kazanmaktadır. Ürün kalitesi ve üretim seviyesi için belirlenen ihtiyaçlar başta robotlar olmak üzere tüm diğer otomasyon sistemleri için üretimin her aşamasında kullanılmasını zorunlu kılmaktadır. Hareket veya hızdan kaynaklanan performans kontrolu da hemen hemen tüm otomasyon işlemlerinin temelini oluşturmaktadır (Lorenz vd., 1994).

Yüksek performanslı servo sistemlerinin moment, hız ve/veya konum kontrolu uygulamaları hareket kontrolu olarak da tanımlanabilir (Lorenz vd., 1994). Mevcut sistemlerde, bu iş için başta doğru akım motorları ve değişken relüktanslı adım motorları olmak üzere bir dizi sistem kullanılmaktadır. Bununla birlikte son zamanlarda en büyük ilerleme vektör kontrollu asenkron motor kullanılması konusunda olmaktadır.

Sincap kafesli asenkron makina, sağlamlığı, geniş hız aralığı, diğerlerine oranla ucuz olması ve az bakım istemesi yüzünden işletim maliyetinin düşük olması sayesinde kendine birçok uygulama alanı bulmuştur (Bose, 1986).

Güç elektroniği ve mikroişlemci alanlarındaki gelişmeye paralel olarak asenkron makina, servo uygulamalarında doğru akım ve senkron makinalarının yerine kullanılmaya başlanmıştır (Bose, 1993).

Makinanın elektriksel dinamiğiyle mekanik dinamiği arasındaki nonlineer kuplaj nedeniyle asenkron makinada yüksek performans elde etmek oldukça güçtür. Bu problemi ortadan kaldırmak üzere standart konum ve hız kontrol çevrimlerinden başka daha içte bir çevrim oluşturacak vektör kontrol teknikleri geliştirilmiştir (Leonhardt, 1985; Vas, 1990; O'Kelly, 1991; Boldea ve Nasar, 1992).

Vektör kontrolunun uygulanmasıyla asenkron makinada akı ile moment arasındaki kuplaj ortadan kalkmakta ve bu iki büyüklük ayrı ayrı kontrol edilebilmektedir. Böylece, makinanın momenti, akı sabitlendikten sonra, stator akımının momenti kontrol eden bileşeniyle lineer olarak kontrol edilebilmektedir.

Burada dekuple edici kontrolu elde edebilmek için stator, hava aralığı veya rotor akı vektörlerinden birinin büyüklüğü ve konumunun bilinmesi gerekmektedir. Asenkron makinada iki vektör kontrol tekniği bulunmaktadır: 1- Doğrudan vektör kontrolu; makinada üretilen akının algılayıcılar tarafından ölçülmesi sonucu elde edilir, 2- Dolaylı vektör kontrol; bu, akının makina uçlarından yapılan ölçümler sonucu akım, gerilim cinsinden türetildiği şeklidir.

Akının makina içinde doğrudan ölçülmesi için özel makina konstrüksiyonuna ihtiyaç duyulduğundan yapılan çalışmada 3-fazlı sincap kafesli asenkron makinanın dolaylı vektör kontrol tekniği kullanılarak kontrol edilmesi benimsenmiştir. Dolaylı vektör kontrolunun dezavantajı makina parametrelerine, örneğin rotor direnci v.b. olan bağlılığıdır (Leonhardt, 1985; Vas, 1990). Bu dezavantaj kullanılan DSP programına, sistem çalışmaya başlamadan önce motorun rotor direnci ölçülerek veya ilgili firmadan temin edilerek girilmesi ve DSP'nin kendisini buradan, yani başlangıç konumundan ayarlayarak değişimleri dikkate almasıyla giderilmektedir. Böylece kontrol yönteminin yanısıra sistem performansını arttırıcı bir diğer ara etken kendiliğinden ortaya çıkmaktadır.

Kontrol edilen sistemin parametrelerindeki değişimlere duyarlılığı bu yöntemin tamamıyla nonlineer ve çalışma parametreleri zamanla değişen bir karakteristiğe sahip asenkron makinanın kontrolunda böyle bir vektör kontrolundeki kısıtlamalar (Leonhardt, 1985; Vas, 1990; Boldea ve Nasar, 1992), DSP tabanlı bir kontrol sistemi ve nonlineer yapıya sahip olan YSA'nın belirli

kabuller çerçevesinde bu yapıdaki sistemlerin çözümüne daha iyi bir sonuçla yaklaşması sayesinde aşılmaktadır.

Çalışma esnasında makina parametrelerinde karşılaşılabilecek en büyük değişimlerden biri de hiç kuşkusuz yük momentidir. Yük momenti gözlenebilirse yüklenme ve yükteki azalma esnasında oluşabilecek anahtarlamalar minimuma indirilebilir (Leonhardt, 1985; Boldea ve Nasar, 1992; Liaw ve Lin, 1994). Sistemde yine bu işlemi üzerine alan ve sırasıyla sinyalleri çok hızlı işleyen DSP ve bu kontrol işlemi çerçevesinde elde edilen değişkenlerin sayısal sonuçların giriş değişkenleri olan rotor hızı, akım ve güce bağlı olarak çıkış değişkeni olan yük moment değerlerini YSA sağlamaktadır.

Yukarıda kısaca değindiğimiz gibi, asenkron makinalar, özellikle sincap kafesli tipleri, düşük maliyetli üretilmeleri, az bakım gerektirmeleri, ayrıca basit ve sağlam yapıda olmaları gibi birçok nedenden ötürü diğer tahrik motorlarına göre modern endüstrinin her alanında yaygın biçimde kullanılmaktadır (Çetin ve Schuisky (1987); Türkmen ve Geçtan (1991); Saçkan (1994)). Endüstride bu kadar yaygın kullanılmalarından dolayı yüksek performanslı çalışmalarının gerekliliği sistem otomasyonu açısından çok önemlidir. Ancak, yakın geçmişe kadar asenkron makinanın kontrolu zor ve pahalı idi. Asenkron makinanın yapısından kaynaklanan; momentin lineer, gecikmesiz kontrolunu engelleyen sebeplerin bazılarını şu şekilde sıralayabiliriz:

Serbest uyarmalı doğru akım makinalarında uyarma ve endüvi devreleri, doğal olarak birbirinden bağımsız biçimde denetlenebildiği halde, bu üstünlüğün asenkron makinada bulunmaması, hız-moment eğrisinin, aynı yük momentinde kararlı ve kararsız çalışma noktalarına sahip olması sebebiyle, ani hız-moment değişimlerinin asenkron makinayı kararsızlığa götürebilmesi, serbest uyarmalı doğru akım makinalarında, fırça-kollektör düzeneği ile doğal olarak, uyarma alanı akısı ve endüvi akım vektörleri birbirine göre dik ve bu konumları değişmez olmasına rağmen; asenkron motorda bunların birbirine göre faz farkları göstermesidir (Durak (1995); Şirin (1996)). O halde burada kısaca üzerinde durulan ve istenilmeyen hususlar dediğimiz durumlar; makina dışı çözümlerle giderilmelidir. Bunun için, güç elektroniği (PWM inverterle kontrol), tümdevre tekniği, sayısal işaret işleyiciler (DSP), mikroişlemciler ve yapay sinir ağları (YSA) ile kontrol alanlarında hem teoride hem de uygulamadaki gelişmeler sayesinde, asenkron motor kontrolunda önemli gelişmeler sağlanmıştır (Çetin ve Schuisky (1987); Vas (1990)). Amaç, makinayı geçici ve sürekli halde, etkileşimsiz olarak kontrol etmek olduğundan, vektörel denetimli kontrolle bunu ilk olarak Blaschke (1977) gidermiştir.

Vektörel denetimli kontrol olayına, makina dışı çözümlerde yukarıda açıklandığı şekilde ilave edilince asenkron makina, mil momenti, ani ve lineer olarak denetlenebilen, yüksek performanslı hız ve servo uygulamalarında kullanılabilen ideale yakın bir tahrik motoru olmuştur. Teknolojik gelişmelere paralel olarak her geçen gün kontrol işlemleri için gerekli birim maliyet düşmektedir.

Asenkron makinada vektör denetiminin sağlanmasında rotor hızının değerce bilinmesine de gereklilik vardır. Bu bilgi sadece veri elde edilmek istendiğinde motor miline bağlı hız algılayıcıları ile yapılmaktadır. Bu da sistemin sağlamlığını ve güvenilirliğini azaltırken maliyeti de arttırmaktadır. Bu nedenle rotor hızının tespit edilebilmesi için özel yapım DC motor üzerine yerleştirilen ölçüm yerinden yani bir takometre üzerinden, DSP özellikle program yapısında yazılı bulunan rotor hızıyla ilgili bölüm sayesinde, kayıt yapması gereken her iterasyon miktarında rotor hızını kaydederek rotor hızını kontrol etmektedir.

1.2 Gerçekleştirilen Sistemin Tanıtılması

Vektör kontrolunun asenkron motorun performansını arttırıcı bir etken olduğunu konuyu tanıtırken belirtmiştik. Buna ilave olarak sistemin performansını arttırıcı diğer etkenler ise kontrolun DSP ile yapılması ve bunun sistemden gelen verilerin işlendiği YSA ile desteklenmesidir. Böylece tüm sistem üzerinde performansı arttırıcı etken sayısı ilave sistem elemanlarıyla üç katına çıkmaktadır. Bunlardan vektör kontrolu asenkron makinanın performansını arttırırken, YSA değişken yük değerlerinde meydana gelen moment kaybını en alt sınırlara çekerek-hem değer hem de zaman olarak-, DSP'de sistem durağan haldeyken ölçümü yapılarak kendisine girilen rotor direncinin (veya ilgili firmadan alınan) değerindeki değişiklikleri takip etmekte ve de sistem kontrolunu hem sistemden verilerin alınıp gerekli yerlere iletilmesi, hem de iletilen yerlerden gelen değerlendirilmiş verileri tekrar sisteme göndererek tam kontrolu gerçekleştirmektedirler. Bu arada yükteki değişim miktarının takip edilebilmesi için YSA'nın giriş değişkenlerinden bir taneside yük değerleri ile ilişkilendirilmiştir.

Bundan önce yapılan kontrol uygulamalarında DSP ile sadece motorun kontrolu gerçekleştirilmiş ve bundan bağımsız olarak da vektör kontrolu ile performansta artış sağlanılmaya çalışılmıştır. Yapılan çalışmada ise sistemin kontrolu; 14 bit ADC (Analog Dijital Dönüştürücü) 14 bit DAC (Dijital Analog Dönüştürücü) içeren ve dinamik değişim aralığına sahip ve saniyede 19.2 kHz maksimum örnekleme frekansı (yapılan çalışmada örnekleme frekansı 15 kHz'tir) olan TLC320C40 Analog Arabirim Devresi (AAD) üzerinden bilgi alış-verişini sağlayan DSP ile gerçekleştirilmektedir. Motora yük olarak akuple edilen ve generatör

şeklinde çalıştırılan DC motorun yüklenme değerleri değiştikçe asenkron motorda meydana gelen performans kaybı ortadan kaldırmakta, mikrosaniyeler mertebesindeki bir zaman aralığından sonra normal çalışma şekline dönmektedir. Sistem kontrolu gerçekleştirildikten sonra DSP'den gelen değişkenlerin sayısal verilerini alan YSA programı içinde bir eğitme işlemine uygulanmakta, güncelleştirilen ağırlıklara bağlı olarak test fazında istenen girişler için klasik Hatanın Geriyeyayılımı Algoritması'nda 1 ms mertebesinde, Hızlı Hatanın Geriyeyayılımı Algoritması'nda 1μs mertebesinde sistemin kalkış anından sonraki genel performansına ulaşmasını sağlamaktadır. Tzou (1996) 'nun gerçekleştirdiği çalışma sadece sağlam yapıda hareket kontrol olayını sağlamaktadır. Bu bakımdan vektör kontrollü ve DSP tabanlı olarak YSA kontrolu altında sistemin oluşturulması çalışmaya ayrı bir üstünlük katmaktadır.

Ayrıca sistemde YSA algoritması olarak Hızlı Geriyeyayılım Algoritması (Fast Backpropagation Algorithm) kullanılarak test fazındaki cevap süresi milisaniyeler mertebesinden mikrosaniyeler mertebesine inmektedir (Karayiannis ve Venetsanopoulas, 1992). Sistemin kontrolunda Texas Instruments firmasının TMS320C50 DSP'si kullanılmış olup sistem kontrolundaki hızı 25-30 nanosaniye mertebelerindedir (Texas Ins. C5x User's Guide, 1997; Marven ve Ewers, 1996).

Sisteme gelen gerçek dünya verilerini işlemek üzere aynı ana kart üzerinde bulunan 14 bit ADC ve 14 bit DAC içeren ve maksimum örnekleme frekansı saniyede 19.2 kHz olan yine Texas Instruments firmasının TLC320C40 Analog Arabirim Devresi kullanılmıştır (Texas Ins. Data Acquisition Circuits Data Book, 1998).

Motorun fazlarından akan akımları algılamak için Escor firmasının Pro5 tipi 2 adet 5 A'lik Hall-Effect sensörü kullanılmıştır.

Şebekeden gelen girişlere ise ABB firmasının açık çevrim 750 Watt gücündeki inverteri bağlanmıştır.

Asenkron motora yükleme yapabilmek için generatör olarak çalıştırılan 220V, 8.5A, 1-2kW, 1500-3000d/d'lık özel yapım labaratuvar tipi bir DC motor kullanılmıştır.

Bilgi alış verişinin sağlanmasında ise bir adet Pentiun 200 MMX PC bilgisayar, YSA programı ise bu bilgisayar üzerinden C++ yazılım dili ile koşturulmaktadır.

Diğer kontrol sistem ve metodlarına göre DSP'nin sağladığı avantaj ve dezavantajları içeren bilgiler aşağıda çizelge 1.1'de verilmiştir (ti.com web sayfası, 1998).

Tezin ikinci bölümünde kontrol işleminin üzerinde yürütüldüğü makina türü olan asenkron makina, üçüncü bölümünde kontrol çeşidi olan vektör kontrol, dördüncü bölümde YSA teorisi genel bilgileri ve kontrol olayını da içerecek şekilde beşinci bölümde DSP'nin genel yapısı,özellikleri, uygulama alanları ve TMS320C50 DSP'si hakkında ayrıntılı bilgi, devre

şemaları ile sunulacaktır. Sonuç bölümünde ise yapılan çalışmanın blok diyagramları, ölçüm ve YSA sonuçları ve sistem üzerinden kontrolu yapılan değişkenlerin eğrileri ve sonuçlar hakkında yorum ve bundan sonra geliştirilecek sistemlerin yapabileceği yenilikler hakkında gelecek tasarımlarıyla ilgili düşünceler verilecektir.

Çizelge 1.1 Asenkron makina uygulamasında sisteme DSP'nin getirdiği yararlar

ASENKRON MAKİNA		MOTOR+DSP
AVANTAJ	DEZAVANTAJ	DSP'NİN SAĞLADIĞI YARARLAR
İstenilen mfg'ye (magneto motor kuvvet) ulaşmak kolay	Kontrolu zor	Alan yönelimli kontrol & Uzay vektör modülasyon
Verimli	Isınmadaki kontrol problemi	Hızlı Analog -Dijital dönüştürme ve yüksek dinamik cevap
Güvenilir	Büyük boyutlar	Faz akımı ve rotor direnç tahmini (program yapısıyla kontrol)
Düşük maliyet		

1.3 Dijital Denetleyici Yapıların Analog Denetleyici Yapılarla Karşılaştırılması

Güç elektroniği ve motor kontrolunda kullanılan sistemlerin daha çok parametrik işlem yapma ve kontrol etme isteği, kullanılmakta olan denetleyici yapıların yetersiz kalmasına neden olmaktadır. Dijital denetleyicilerin gelişimi, bu konuda analog denetleyicilerin kulanım zorunluluğundan ortaya çıkan yetersizliğin ortadan kalkmasını sağlamıştır. Analog denetleyici, pasif ve aktif devre elemanları kullanılarak sistem parametrelerinin ancak sınırlı değişim aralığına izin veren yapılardır. Lojik kapılar, işlemsel kuvvetlendiriciler ve pasif devre elemanları ile tasarlanan sistemin getirdiği sorunlar, mikroişlemci tabanlı dijital denetleyicilerin kullanımı ile ortadan kalkmış, ancak bu yapılarda kendilerine özgün sakıncaları beraberinde getirmişlerdir.

Dijital denetleyici olarak adlandırılan sistem, mikroişlemci tabanlı, kontrol edilen sistem ile gerekli olan haberleşmeyi sağlayan donanım özelliklerine sahip ve sisteme uygun bir yazılım desteği olan bir yapıdır. Dijital denetleyicilerin sağladığı en önemli faydalardan biri esneklik özelliğidir. Yani farklı kontrol problemlerine aynı donanım fakat değişik yazılımlar ile kolaylıkla

uyum sağlayabilirler. Çözüm olanaklarının bu şekilde program ile çözümlenebilmesi, kontrol edilecek sistemin parametreleri değiştiğinde, analog yapıya ait devredeki elemanları değiştirmenin zorluğu yanında çok büyük bir avantaj getirmektedir.

Kontrol edilen sistemin güvenilirliği, denetleyici yapısının güvenilirliği ile doğru orantılıdır. Analog denetleyicinin yapısındaki elemanların birbiri ile bağlantısı, sistem büyüdükçe karmaşık bir yapıya sahip olacak, böylece hata olasılığı artacaktır. Dijital denetleyiciler için diğer bir avantaj da; çevre koşullarından minimum düzeyde etkilenmesidir. Analog devre elemanlarının sıcaklığa çok bağımlı olması, sistemin hatalı davranma olasılığını arttıracaktır.

Son yıllarda geliştirilen modern kontrol algoritmalarının analog bir yapı ile gerçekleştirilmesi neredeyse imkansız hale gelmiştir. Kontrol edilen sistem karmaşıklaştıkça, sistemin parametreleri ve bağlantı sayısı artmakta böylece analog denetleyicinin bu yapısı hata ve büyüklük bakımından en önemli problemi teşkil etmekte kısacası, bu noktada iflas etmektedir. İşlemlerin sayısının artması ve karmaşık hesapların fazlalığı dijital denetleyiciden başka bir çözüme olanak sağlamamaktadır.

Fiziksel sistemler ile haberleşme yapılırken, verinin uzak mesafelere taşınması sırasında bilgi kaybı olmaktadır. Analog işaretler, akım ve gerilim seviyesi olarak değişik değerlerde taşınırken, dijital işaretlerde gerilim seviyesinin 1 veya 0 olması söz konusudur. Çok küçük değerli işaretler, magnetik etkileşim ve parazit etlilerinden dolayı anlaşılamaz hale gelerek, sistemin yanlış davranmasına neden olabilir. Dijital sistemlerde bu hata minimum düzeydedir.

Buna karşılık fiziksel sistemler analog yapıda olduklarından, dijital denetleyici yapısının kullanılabilmesi için, işaretlerin A/D (analog-dijital) ve D/A (dijital-analog) çeviriciler yardımıyla dönüştürülmesi gerekmektedir. Bu da belirli bir bilgi kaybına neden olmaktadır. Analog ve dijital denetleyiciler arasındaki karşılaştırma çizelge 1.2'de dijital denetleyici yapının blok diyagramı ise şekil 1.1'de verilmiştir (Çiprut, 1994).

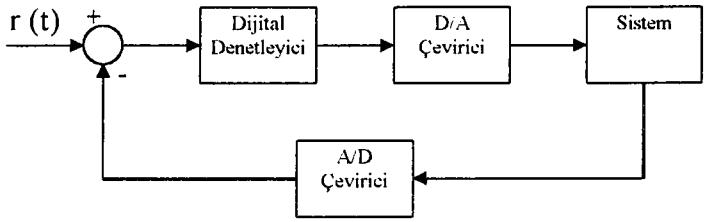
Dönüşüm işlemi sırasında oluşan anahtarlama kaybı ve kuantalama hatasından dolayı ortaya çıkan hatalar, dijital bilgi işleme sisteminin bit sayısının ve örnekleme frekansının arttırılması ile azaltılabilir.

Dijital denetleyicinin işaret işleme sırasında sayısal hatalar nedeniyle kontrol sisteminde kalıcı bir hata bulunmaktadır. Analog denetleyicide bu şekilde kalıcı bir hata, devredeki eleman değerlerinin doğru seçilmesi halinde bulunmamaktadır. Mikroişlemcili bir kontrol sisteminin cevabı, fiziksel bir kontrol sistemin cevabı yanında yavaş kalabilir. Bunun nedeni analog denetleyicinin işaretleri paralel yollardan ihmal edilebilecek bir gecikme ile yollamasına rağmen, dijital denetleyici, seri yol üzerinden haberleşerek, daha uzun bir süre harcamaktadır. Bilgi işlemedeki bu gecikme, kapalı çevrim kontrol sistemlerinde kararlılık problemlerine ve işlenen

işaretlerde sorunlara neden olmaktadır. Fakat, DSP bünyesinde hem mikroişlemci hem de denetleyici yapıyı bulundurduğundan dezavantaj olarak görünen birçok durumu avantaja çevirerek üstesinden gelmektedir (ti.com web sayfası, 1998).

Çizelge 1.2 Analog ve dijital denetleyicilerin karşılaştırılması

	AVANTAJ	DEZAVANTAJ
ANALOG DENETLEYİCİ		
	Yüksek doğruluk	Elemanların yaşlanması
	Tasarım kolaylığı	Sıcaklığa bağımlılık
	Gerçek zamanda çalışma	Kablolu bağlantı ve bağlantı sayısı çokluğu
	Sonsuz örnekleme zamanı	
DİJİTAL DENETLEYİCİ		
	Çevre koşullarından etkilenmeme	Sayısal hatalar
	Doğru kesin davranış	Örnekleme zamanın etkisi
	Karmaşık kontrol algoritmalarına uygun	Hız problemleri
	Ek fonksiyonlarına olanaklı	



Şekil 1.1 Dijital denetleyici yapısı

Dijital denetleyici kullanılarak gerçekleştirilen sistem, yazılım maliyeti nedeniyle daha pahalı olabilir. Fakat kontrol edilen sistem büyüyüp, parametreleri arttıkça, denetleyici sisteminin gerçekleştirilme maliyeti, dijital sistemler için bir avantaj haline gelmektedir. Nitekim bu da günümüzde fabrika otomasyon maliyetlerinde geniş bir şekilde kendini göstermektedir. Özellikle karmaşık kontrol algoritmalarının uygulanmasında, uygulama alanı arttıkça bu şekildeki işlemcilerin fiyatlarının düşmesi ve kontrol edilen sisteme kolayca uyum sağlamaları

büyük bir avantaj oluşturmaktadır. Son yıllarda işlemci kullanımı arttıkça firmalar rekabet amacıyla değişik özellikte işlemciler üreterek, fiyatları aşağı çekmişlerdir. Analog yapılar ise ancak belirli uygulamalara yönelik üretildiklerinde çok daha pahalı hale gelmişlerdir (Bose, 1986).

2. ASENKRON MAKİNANIN ANALİZİ

2.1 Asenkron Motor

Yapılan çalışmada prototip olarak asenkron makina ele alındığından bu bölümde motorun yapısı, performansını etkileyen temel matematiksel ifadeler ve diyagramlar, bağlantı şemaları, kontrol şekilleri ve standartlara göre test prosedürleri üzerinde durulacaktır.

2.1.1 Asenkron motorun yapısı

Asenkron motorlar endüstride en fazla kullanılan motorlardır. Asenkron motorların devir sayıları yükle çok az değişir. Doğru akım şönt motorlarının devir sayısı büyük sınırlar içinde değiştirilebildiği halde , asenkron motorun devir sayısı sınırlı olarak arttırılabilir veya azaltılabilir. Asenkron motorlar tüm yönleriyle ele alındığında,

1. Daha ucuzdur,
2. Peryodik bakıma daha az ihtiyaç gösterir,
3. Çalışması sırasında elektrik arkı meydana gelmez, (D.A. motorları çalışırken kollektör dilimleri ile fırçalar arasında kıvılcımlar çıkar).

Bu özellikler, asenkron motorların endüstride en çok kullanılan motorlar olmalarına sebep olmuştur. Asenkron motorlar genel olarak stator ve rotor olmak üzere duran ve hareketli iki kısımdan meydana gelmektedir (Saçkan, 1994).

Burada yüklenme durumundaki asenkron motorun performans kaybının YSA ile önlenmesine ve yine aynı sistem üzerinden momentinin kontrolüne yeni bir yaklaşım sunulmuştur.

2.1.2 Asenkron motorun çalışma prensibi

3 fazlı, 2 kutuplu bir asenkron motora şebekenin RST faz elektro-motor-kuvvetleri (emk)'ları uygulanır. Statordaki sargılardan geçen alternatif akımlar, dönen NS kutuplarını meydana getirirler. Stator sabit olduğu halde, dönen NS kutupları ortadaki kısa devreli rotorun çubuklarını keserek çubuklarda emk'ları indükler, kısa devreli rotor çubuklarından endüksiyon akımları geçer. Döner alan (NS) kutupları saat ibresi yönünde dönüşünü devam ettirir. Bu endüksiyon akımları rotorun NS kutuplarını meydana getirirler. Dönen stator kutupları rotorun kutuplarını etkileyerek (benzer kutuplar birbirini iter, zıt kutuplar birbirini çeker prensibinden haraket ile) rotoru saat ibresi yönünde döndürürler.

N kutbunun altındaki rotor çubukları bir yöne, S kutbunu altındaki rotor çubukları diğer yöne doğru itilirler. Bu itme kuvvetlerinin meydana getirdiği döndürme momenti rotoru saat ibresi yönünde, döner alanın yönünde döndürür.

Rotorun devir sayısı (n_r) arttıkça, döner alanın rotor çubuklarını kesmesi azalacağından, rotor çubuklarında endüklenen emk'ler ve kısa devre çubuklarından geçen endüksiyon akımları azalır. Dolayısı ile, rotoru döndüren moment azalır. Rotorun devir sayısında artış olmaz. Motor boşta çalışırken rotorun devir sayısı senkron devir sayısına (döner alanın devrine) yaklaşır. Döner alanın devir sayısı (senkron devir) n_S ile rotor devir sayısı n_r arasındaki farka "Rotorun Kayması" denir. Diğer bir ifade ile, rotor devrinin senkron devirden geri kalmasına "Kayma" denilmektedir. Kaymanın senkron devir sayısı ile ifadesi,

$$s = \frac{n_s - n_r}{n_s}.100 \qquad (2.1)$$

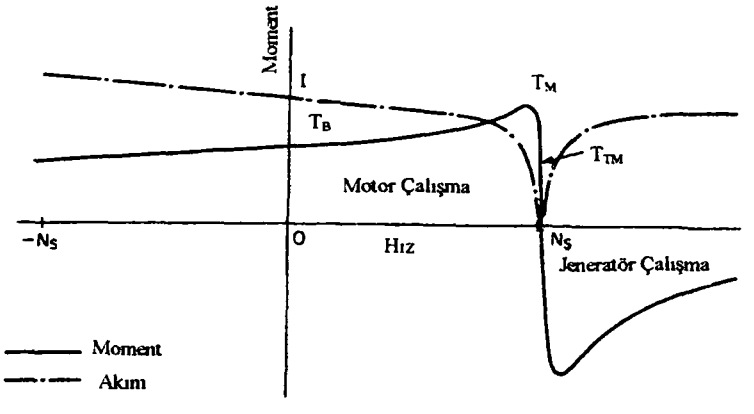
$$n_r = (1-s).n_s \quad \text{' dir.} \qquad (2.2)$$

Denklem 2.1'den de görüleceği gibi rotorun devir sayısı n_r hiçbir zaman döner alanın devir sayısına yani senkron devire (n_S) eşit olmaz. Bu da rotorun senkron devirden daha az bir devirle döndüğünün göstergesidir.

Bu arada her faz sargısında endüklenen emk'nın o sargıya uygulanan şebekenin faz emk'sına zıt yönde olduğunu, bu zıt emk'nın transformatörlerde primer sargıdan geçen alternatif akımın meydana getirdiği , çekirdekte dolaşan ve değişmekte olan manyetik akının primer sargı üzerinde endüklediği zıt emk'ya benzetilebildiğini hatırlatmakta yarar vardır.

2.1.3 Performans karakteristikleri

Şekil 2.1 de çok fazlı bir asenkron motor için hız-moment ve hız-akım eğrileri verilmiştir. Bu eğrilerde üç önemli bölge; motor çalışma, motoru frenleme ve jeneratör çalışmadır (Nasar, 1987).



Şekil 2.1 Çok fazlı bir asenkron motor için hız-moment ve hız-akım eğrileri

Bu eğriler incelendiğinde, motorun çalışma prensibinden bahsederken ifade edilmeye çalışılan senkron hız n_S, kayma s gibi sayısal değerleri içeren kısımları da anlamak daha kolay olmaktadır. Yine bu eğri ile karşımıza çıkan kritik moment ifadesi, motor çalışma bölgesinde meydana gelen maksimum momenttir. Bu şekilde T_M ile gösterilmiştir. Rotorun bloke edilmesi ile ortaya çıkan moment ise aynı şekilde T_B (kilitli-rotor momenti) ile ifade edilmektedir. Aynı zamanda aynı yükün zorlamasıyla gönderilen herhangi dağınık kuvvetlerin üzerinden ortaya çıkması gereken önemli bir moment ifadesidir. Normalde bir asenkron motor, maksimum moment T_M ve senkron hız n_S arasında çalışır. Eğride T_{TM} ile gösterilen nominal tam yük momenti ile hız ekseni arasında kalan yer, çalışma bölgesidir.

2.1.4 Asenkron motorun dinamikleri

Bu noktada, sistem kontrolunun gerçekleştirilebilmesi için daha sonraki bölümler de karşımıza çıkacak olan motor dinamiklerini biraz tanıtmak amacı ile fazla kapsamlı olmayan bilgi verilecektir.

Bir bütün olarak beygir güçlü asenkron motorları arasında, yani güce ihtiyaç duyulan çalışma koşullarındaki motorlarda, en yaygın dinamik problemler motoru harekete geçirme (başlatma), motoru hareket dışı hale getirme (durdurma) ve mevcut sistemin gürültü olarak tanımlanan sistem performansını etkileyen parazitler süresince, motorun çalışmasını aksatmadan devam ettirebilmesi için, motorun çalışma performansı ile yakından ilgilidir. Örneğin, bir endüstri kuruluşunda karşılaşılan tipik bir problem; çalıştırılmaya başlanmış bir motora aşırı ani bir akımın gelmesi ile sebep olunan gerilim azalmasından dolayı normal çalışmayı durdurmakta olup, diğer paralel bağlı motorlar da herhangi bir operasyona gerek kalmaksızın bahsetmekte olduğumuz büyük güçlü motorun çalışma kabiliyetine nasıl tesir edilebileceğidir.

Dinamik analizlerdeki asenkron motorun ifade metodları, gerçeklenebilir bir kapsam dahilinde problemin yapısının ve karmaşıklığının tam veya kesin olarak tanımlanmış veya ifade edilmiş olma niteliğine bağlıdır. Yine bu analizler dahilinde elektriksel geçici rejimler makina dinamikleri için önemli bir yer işgal etmektedir. Eğer elektriksel geçici rejimleri ihmal edilebilir olarak düşünürsek olayı bir boyutta basitleştirmiş oluruz. Çünkü elektriksel geçici rejim isminden de anlaşılacağı gibi, çok hızlı bir şekilde çalışmakta olduğumuz elektriksel olayın dışına çıkmakta yani yatışmaktadır (hareketli rejimin sürekliliği ile karşılaştırıldığında geçici rejim çok kısa bir zaman diliminde çok hızlı olarak kaybolmaktadır). Bu durum için kabulumüz 3600 devir/dakika dan büyük motorlardır.

Bu şartlar altındaki motorun ifadesi, kalıcı hal teorisine dayandırılabilir. Karşılaşılan problemin daha sonraki dinamik analizi , özellikle sistem içinde karşılaşılan lineeersizliklerden dolayı sistemi kontrolu zor bir hale getirmeyecek olan oldukça basit, fakat oldukça gerçekçi bir şekilde (matematiksel olarak) ifade etmeyi amaçlar. Nispeten basit problemler için yapılan bir yaklaşım şekli ise grafiksel yöntemdir. Hem motor tarafından üretilen hem de yükü döndürmek için

ihtiyaç duyulan moment, eğrisel formda verilen bilgi için hızın lineer olmayan fonksiyonları olarak düşünülür.

Motor dinamikleri içinde zamana karşılık düşen motor hızı bilinirse, zamana karşılık gelen akım karekteristiği de, ya hıza karşı bir akım eğrisi ya da analitik ifadelerle elde edilen bir sonuç olur. Endüvi akımının büyüklüğü, mıknatıslama reaktansının düzeninde etkin rotor empedansı olduğu noktada hız artıncaya kadar sayısal değer olarak büyük kalır. Bu noktada akım kalıcı hal çalışma değerine düşmeye başlar Yüksüz halde bu akım motorun mıknatıslama akımıdır. Tam yük altında ise, bu akım mıknatıslama akımı ve stator yük akımının toplamını içerir.

Bazı durumlarda, nominal gerilim altındaki motoru başlatma akımı çalışma şartları doğrultusunda aşılabilir.

Bu anlatılanlara belirli sınırlamalar getirmek ve motoru istenilen şekilde çalıştırmak için motorda seri empedans kullanımı, motora nominal gerilim uygulama, Y-Δ anahtarlama ve bölünmüş-sargı kullanımı (motoru harekete geçirme) gibi çalışmalar yapılarak değerce azaltılmış başlatma akımları ve momentleri elde edilir.

2.1.5 Asenkron motorlarda hız kontrolü

Yapay Sinir Ağları ile DSP tabanlı olarak performans arttırımı başlı başına bir kontrolü içerdiğinden bu noktada kısaca asenkron motorların kontrol olayından bahsedilecektir.

Basit bir asenkron motor sabit hız sürücülerinin gereksinimlerini önemli ve yeterli derecede karşılar. Ancak bir çok motor uygulamaları bir kaç hıza veya kademeli olarak ayarlanabilir hızın sürekli değişimine de ihtiyaç duyar. AC güç sistemleri tasarlandığı ilk dönemlerde mühendisler ayarlanabilir hıza sahip ac motorların gelişimi ile geniş bir şekilde uğraşmışlardır.

Yapılan çalışmalar sonucunda bir asenkron motorunun senkron hızı,

1. Kutup sayılarının değişimi ile
2. Hat frekansının değişimi ile

Kayma ise;

3. Hat gerilimini değiştirmekle
4. Rotor direncini değiştirmekle
5. Rotor devrelerine, uygun frekans gerilimlerini uygulamakla değiştirilebilir

Hız kontrol metodları yukarıda verilen bu beş özelliğe dayanır.

2.2 Asenkron Motorun Matematiksel Modelinin Teorisi

Genelde, bir sistemin matematiksel modeli, sistemin fiziksel davranışının simülasyonunu yapmak veya bir algoritmaya dayanarak gerçek zamanda kontrolü gerçekleştirmek için gereklidir. Matematiksel modeller, sistemin gerçek fiziksel davranışına oldukça uyumlu olmalı ve o davranışı iyi bir şekilde yansıtmalıdır. Model basit olmalı ve en az varsayıma dayanarak oluşturulmalıdır. Modelin karmaşıklığı, kontrol işlemlerinin süresini uzatacak ve böylece sistem

performansını düşürecektir. Modellerde kullanılan makinanın fiziksel büyüklüklerinin, skaler değil de, vektörel olarak gözönüne alınabilmesi sistem modelinin doğruluğunu arttırmaktadır. Özellikle, makinaların geçici hal davranışından oluşacak hata önemli derecede azalır.

Modellemenin en önemli aşamalarından biri de modellenecek sistemin giriş ve çıkış büyüklüklerinin belirlenmesidir. Zira bu aşamadan sonra modelleme, bu giriş ve çıkışlar arasında uygun matematiksel model, entegro-diferansiyel denklem sisteminin oluşturulmasıyla son bulur. Bu durumda giriş ve çıkış arasındaki denklemlerin fiziksel sistemi mümkün olduğunca iyi modellemesi, buna karşın mümkün olduğunca da basit olması gerekmektedir.

Asenkron motorda giriş büyüklükleri stator d, q eksen akımları (YSA için bunlardan birisinin genlik değeri yeterli olmaktadır), rotor hızı ve yük miktarı, çıkış büyüklüğü ise elektriki yük momentidir.

Bu bilgilere dayanarak, sincap kafesli asenkron makina modeli oluşturulurken aşağıdaki modelleme varsayımlarının çalışma amacına yönelik olarak birkaçı veya tamamı uygulanabilir:

1- Stator sargıları, stator çevresinde düzgün olarak yayılmıştır. Hava aralığında akı sinüsoidal biçimdedir.

2- Üç fazlı stator sargıları çevreye, 120 derecelik faz farklı olarak düzgün biçimde yayılmışlardır.

3- Doyma, diş ve oluk etkileri ihmal edilmiştir.

4- Magnetik kısımların geçirgenliği sonsuz varsayılmıştır.

5- Histeresis ve Foucault kayıpları ihmal edilmiştir.

6- Akım yığılması (deri olayı) ihmal edilmiştir.

7- Rotor çubukları, rotor eksenine göre simetrik yayılmışlardır.

8- Dirençler ve endüktansların sıcaklıktan bağımsız oldukları varsayılmıştır.

Makinayı besleyen güç kaynağı dengeli üç fazlı gerilim üretiyorsa model, iki eksende veya d, q eksenlerinde temsil edilebilir. Vektör kontrolünde kullanılan d-q modeli sayısal simülasyonlarda ve kontrol sistemlerinin tasarlanmasında kullanılmaya elverişlidir. En çok kullanılan referans eksen takımları $w=0$ (sürekli hal) ve $w= w_s$ (senkron dönme) hızlarındadır.

Buradan, kullanılacak olan üç fazlı büyüklükler üç eksenli as-bs-cs koordinat sisteminden iki eksenli durağan d-q koordinat sistemine dönüştürülür. Yukarıda teorisi verilen modelin açık matematiksel ifadeleri bölüm 3'te sunulacaktır.

3. 3-FAZLI ASENKRON MOTORUN MATEMATİKSEL MODELİ VE VEKTÖR KONTROLÜ

3.1 Giriş

Son birkaç yıl boyunca elektrikli sürme devrelerinin vektör kontrolüne uygulanması açısından, hem güç ve sinyal elektroniği hem de mikro-elektronik işlemcilerde ve DSP'ler de hızlı gelişmeler olmuştur. Bu teknolojik gelişmeler, çok düşük mertebede güç dağılım donanımı ve daha doğru kontrol yapılarıyla gerçek zamanda etkin AC sürme kontrolünün gelişimini sağlamıştır. Elektrikli sürme devre kontrolleri sadece DC akım ve gerilim kontrolünde değil, aynı zamanda üç fazlı AC akım ve gerilimleri vektör kontrol yapılarıyla birlikte hassas bir şekilde kontrol ederek daha doğru sonuçlara ulaşılmasını sağlamaktadır. Bu bölümde üç temel nokta üzerinde durulacaktır: 1- Kontrol edilen makinanın akım ve gerilim uzay vektörleri, 2- Üç faz hız ve zaman bağımlı sistemden, iki koordinatlı zamanla değişmeyen sisteme ve 3- Etkin Darbe Genişlik Modülasyonu (PWM)'in üretimidir. Bu üç faktör, AC makina kontrolünde DC makinanın kontrolünün her avantajını kazandırır ve mekanik hesaplama dezavantajlarından kendi kendini korur. Anlatılanları kısaca özetleyecek olursak, bu kontrol yapısı çok doğru kararlı ve geçici hal kontrolünü başarıyla yaparak cevap zamanları ve güç dönüşümü bakımından yüksek dinamik performansa izin verir.

3.2 Klasik AC Sürme Teknikleri

AC motor kontrol yapıları genellikle statorun üç fazındaki 120 derece faz farklı olarak dağılmış üç sinüsoidal gerilime uygulanır. Klasik AC sürme yapılarının çoğunda bu üç sinüs dalgasının üretimi, motorun elektro-mekanik karakteristiklerine ve motorun kararlı haldeki eşdeğer devre prensibine dayanır. Ayrıca, bu çeşit bir 3-fazlı sistemin kontrolü daha çok, üç ayrı tek-fazlı bir sistemin kontrolüne benzer. Bu noktada karşılaşılan bazı dezavantajlar vardır (Tex. Ins. Yayını, 1998).

1- Kullanılan makina modelleri ve karakteristikler sadece kararlı halde geçerlidir. Bu, kontrol işleminde yüksek değerli gerilim ve akım geçişlerine (geçici haller) izin verir. Bunlar sadece sürme devresinin dinamik performansına zarar vermekle kalmaz aynı zamanda güç dönüşüm verimini azaltır. Buna ilave olarak, güç bileşenleri, geçici fakat yüksek değerli elektriksel darbelere (piklere) karşı koyabilmesi için büyük güçte olmalıdır.

2- Sinüsoidal referanslarla değişkenleri kontrol etmekte karşılaşılan önemli problem: PI kontrolörler, sinüsoidal referansa hasar vermeksizin bir sinüsoidal regülasyon ve buradan histeresis kontrolörler sisteme filtre edilmesi oldukça zor olan yüksek bant genişlikli gürültü verirler.

3- 3-fazlı sistemde dengessizlik olmaz ve faz etkileşimleri düşünülmez.

4- Sonuç olarak, kontrol yapısı motor çeşidine göre (asenkron veya senkron) incelenmelidir.

Alan yönelmeli vektör kontrolü bu dezavantajların herbirinin üstesinden gelir ve böylece AC sürme tekniğinin etkinliğini oldukça arttırır. Bu konu üzerinde performans arttırıcı çalışma yapanlara böyle çözüme dayalı, hızlı ve avantajlı bir kontrol yapısı sunar.

3.3 Alan Yönelmeli Vektör Kontrol

Alan yönelmeli vektör kontrol (Tex. Ins. Yayını, 1998 ve Di Gabriele, 1997) bir vektörle temsil edilen stator akımlarının kontrolünden meydana gelir. Bu kontrol bir 3-fazlı zaman ve hız bağımlı sistemden, bir 2-koordinatlı (d ve q) zamanla değişmeyen sisteme doğru dönüşen izdüşümlere dayanır. Bu izdüşümler bir DC makina kontrolünün durumuna benzer bir yapıya izin verir. Alan yönelmeli vektör kontrollü makinalar giriş referansları olarak iki sabite ihtiyaç duyar: moment bileşeni (q koordinatıyla ayarlanmış) ve akı bileşeni (d koordinatıyla ayarlanmış). Alan yönelmeli vektör kontrol basit olarak izdüşümlere dayandığından, kontrol yapısı ani elektriksel değerler kullanır (o andaki mevcut akımın genlik değeri gibi) Bu kontrol şeklini her çalışma durumunda daha doğru (kalıcı ve geçici hal) yapar ve sınırlı bant genişliği matematiksel modeli bağımsız hale dönüştürür. Alan yönelmeli vektör kontrol böylece klasik problemleri aşağıdaki şekilde çözer:

1- Sabit referansa ulaşmak (moment ve stator akı bileşenleri),

2- d, q referans çerçevesinde momentin belirtilmesi aşağıdaki şekilde olduğundan direkt moment kontrolünü uygulamak.

$$T \propto \psi_r \, i_{sq} \qquad (3.1)$$

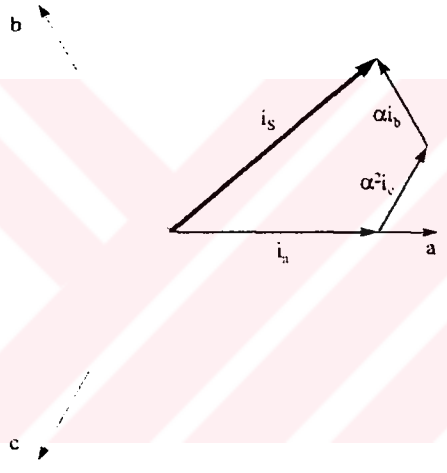
Sabit bir değerde rotor akısı ψ_r 'nin genliğini koruyarak moment T ve akımın moment bileşeni i_{sq} arasında lineer bir ilişkiye sahip oluruz. Sonra, stator akım vektörünün moment bileşenini kontrol ederek momenti kontrol edebiliriz.

3.4 Uzay Vektör Belirleme ve İzdüşümü

AC motorların 3-fazlı gerilimleri, akımları ve akıları kompleks uzay vektörleri bakımından analiz edilebilirler (Zhang vd., 1994). Akımlara ait, uzay vektörler aşağıdaki gibi belirlenebilir; burada i_a, i_b, i_c akımlarını stator fazlarındaki ani akımlar olarak varsayarak kompleks stator akım vektörü $\bar{i}_S$ şu şekilde ifade edilebilir.

$$\bar{i}_S = i_a + \alpha i_b + \alpha^2 i_c \tag{3.2}$$

burada, $\alpha = e^{j\frac{2}{3}\pi}$ ve $\alpha^2 = e^{j\frac{4}{3}\pi}$ özel operatörleri temsil eder. Şekil 3.1'deki diyagram stator akımının kompleks uzay vektörünü göstermektedir.



Şekil 3.1 Stator akım uzay vektörü ve (a, b, c)'deki bileşeni

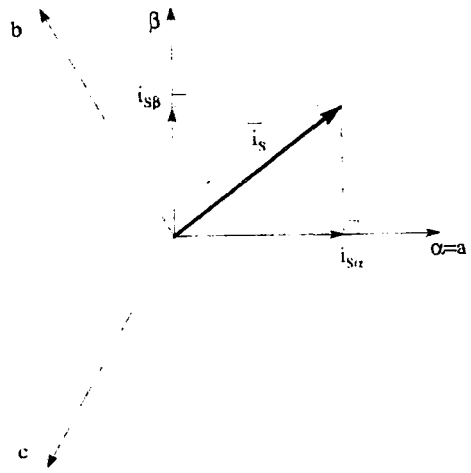
Burada (a, b, c) 3-fazlı sistem eksenleridir. Bu akım uzay vektörü 3-fazlı sinüsoidal sistemi gösterir. Bu durumda zamanla değişmeyen 2-koordinat sistemine dönüştürülmeye ihtiyaç vardır. Bunlar;

1- (a, b, c) $\Rightarrow$ (α, β) 2-koordinatlı zamanla değişen sistem (Clarke dönüşümü)

2- (α, β) $\Rightarrow$ (d, q) 2-koordinatlı zamanla değişmeyen sistem (Park dönüşümü), şeklindedir.

3.4.1 Clarke dönüşümü

Uzay vektörü (α, β) denen 2 ortogonal eksenle başka bir referans çerçevesinde yazılabilir. a ekseni ve α eksenini aşağıdaki vektör diyagramında aynı yönde varsayarak gösterebiliriz.



Şekil 3.2 Stator akım uzay vektörü ve (a, b)'deki bileşenleri

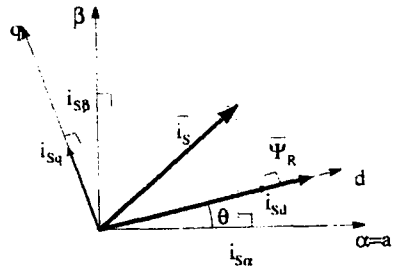
3-fazlı sistemi (α, β) 2 boyutlu ortogonal sisteme çeviren izdüşüm aşağıda verilmiştir.

$$\begin{cases} i_{S\alpha} = i_a \\ i_{S\beta} = \dfrac{1}{\sqrt{3}} i_a + \dfrac{2}{\sqrt{3}} i_b \end{cases} \tag{3.3}$$

Bu durumda zaman ve hız bağımlılığı devam eden 2-koordinatlı bir sistem $\begin{pmatrix} i_{S\alpha} \\ i_{S\beta} \end{pmatrix}$'yı elde etmiş oluruz.

3.4.2 Park dönüşümü

Bu dönüşüm vektör kontrolünün en önemli kısmıdır. Gerçekte, bu izdüşüm d, q dönen referans çerçevesinde 2-fazlı bir ortogonal sisteme dönüştürür. d ekseninin rotor akısıyla uyarlandığını düşünürsek, aşağıdaki diyagram akım vektörü için iki referans çerçevesi arasındaki ilişkiyi gösterir.



Şekil 3.3 Stator akımının uzay vektörü ve bu akımın (a, b)'de ve d, q dönen referans çerçevesindeki bileşeni

Burada θ rotor akı pozisyonudur. Akım vektörünün akı ve moment bileşenleri aşağıdaki denklemlerle elde edilir.

$$\begin{aligned} i_{Sd} &= i_{S\alpha}\cos\theta + i_{S\beta}\sin\theta \\ i_{Sq} &= -i_{S\alpha}\sin\theta + i_{S\beta}\cos\theta \end{aligned} \tag{3.4}$$

Bu bileşenler akım vektöründeki (α, β) bileşenlerine ve rotor akı pozisyonuna bağlıdır. Doğru rotor akı pozisyonu bilinirse, bu izdüşümde d, q elemanları sabit olur. Buradan aşağıdaki karakteristiklere uyan bir 2-koordinat sistemi elde edilir $\begin{pmatrix} i_{Sd} \\ i_{Sq} \end{pmatrix}$:

1- 2-koordinatlı zamandan bağımsız sistem,

2- i_{sd} (akı bileşeni) ve i_{sq} (moment bileşeni)'yle direkt moment kontrolü $\begin{pmatrix} i_{Sd} \\ i_{Sq} \end{pmatrix}$ mümkün ve kolaydır.

3.4.3 Ters Park dönüşümü

Bu gerilim dönüşümünden sadece 2-fazlı ortogonal sistemde d, q dönen referans çerçevesindeki gerilimleri değiştiren denklem verilecektir.

$$\begin{aligned} v_{S\alpha ref} &= v_{Sdref}\cos\theta - v_{Sqref}\sin\theta \\ v_{S\beta ref} &= v_{Sdref}\sin\theta + v_{Sqref}\cos\theta \end{aligned} \tag{3.5}$$

Bu bloğun çıkışları $\bar{V}_r$ referans vektörünün elemanlarıdır. V_r motor fazlarına uygulanılan gerilim uzay vektörüdür. Tüm bu dönüşüm sistemi çizelge 3.1'de gösterilmektedir.

Çizelge 3.1 Dönüşümler tablosu

Park Dönüşümü	Ters Park Dönüşümü
a, b, c $\Rightarrow$ α, β	d, q $\Rightarrow$ α, β
$i_\alpha = \frac{2}{3} i_a - \frac{1}{3}(i_b - i_c)$ $i_\beta = \frac{2}{\sqrt{3}}(i_b - i_c)$ $i_0 = \frac{2}{3}(i_a + i_b + i_c)$ $\Rightarrow$ $i_\alpha = i_a$ $i_\beta = \frac{1}{\sqrt{3}} i_a + \frac{2}{\sqrt{3}} i_b$ $i_a + i_b + i_c = 0$	$i_\alpha = i_{sd} \cos(\theta) - i_{sq} \sin(\theta)$ $i_\beta = i_{sd} \sin(\theta) + i_{sq} \cos(\theta)$
α, β $\Rightarrow$ d, q	α, β $\Rightarrow$ a, b, c
$i_{sd} = i_\alpha \cos\theta + i_\beta \sin(\theta)$ $i_{sq} = -i_\alpha \sin(\theta) + i_\beta \cos(\theta)$	$i_a = i_\alpha$ $i_b = -\frac{1}{2} i_\alpha + \frac{\sqrt{3}}{2} i_\beta$ $i_c = -\frac{1}{2} i_\alpha - \frac{\sqrt{3}}{2} i_\beta$

3.5 Alan Yönelmeli Kontrolde Kullanılan Motorun Matematiksel Modeli

AC sürme devrelerindeki model alan koordinatlarında tanımlanır (Texas Ins., 1997; Leonhard, 1985; Vainio vd., 1992; Simor, 1995; Tzou, 1996). Mekaniksel kayıplar genellikle diğer asenkron motorlarda olduğu gibi, modelin oluşturulmasında dikkate alınmayacaktır. Bu denklem takımlarında sistem lineer değildir. r indisi rotor bileşenlerini s indisi stator bileşenlerini temsil etmektedir.

$$\vec{v_s} = R_s \vec{i}_s + L_s \frac{d\vec{i}_s}{dt} + L_m \frac{d}{dt}(\vec{i_r} e^{j\theta_r}) \tag{3.6}$$

$$0 = R_r \vec{i}_r + L_r \frac{d\vec{i}_r}{dt} + L_m \frac{d}{dt}(\vec{i_s} e^{-j\theta_r}) \tag{3.7}$$

$$J \frac{dw_r}{dt} + Bw_r = T_e - T_d = \frac{P}{2} L_m I_m \left[\vec{i_s} (\vec{i_r} e^{j\theta_r})^* \right] - T_d \tag{3.8}$$

$$\frac{d\theta_r}{dt} = w_r \tag{3.9}$$

$$T_e = \frac{P}{2}\frac{L_m}{L_r}(i_{sq}\psi_{rd} - i_{sd}\psi_{rq}) \tag{3.10}$$

Bu bir direkt olmayan vektör kontrol metodudur. Fakat bu kontrolün sonunda moment YSA yaklaşımıyla direkt olarak kontrol edilmektedir. Model denklemlerini kontrol işlemini basitleştirecek şekilde vermeye devam edelim.

$$\vec{\psi}_r = L_r \vec{i}_r + L_m \vec{i}_s e^{-j\theta_r} \tag{3.11}$$

buradaki rotor akı vektörü geri besleme stator akımlarını kullanan d, q eksenlerindeki bir rotor akı modeliyle hesaplanabilir,

$$\psi_r = \frac{L_m}{s\tau_r + 1} i_{sd} \tag{3.12}$$

kayma açısal frekans ise,

$$w_{sl} = \frac{L_m}{\tau_r \psi_r} i_{sq} \tag{3.13}$$

dekuplaj kontrolde koordinasyon çevrim açısını üretmek için kullanılır.
d ekseni rotor akı vektörüne tam olarak sabitlenirse,

$$\psi_{rd} = \psi_r \quad ve \quad \psi_{rq} = 0, \quad \frac{d\psi_{rq}}{dt} = 0 \quad \text{olur,} \tag{3.14}$$

(3.10) nolu denklem şu şekle indirgenmiş olur,

$$T_e = \frac{P}{2}\frac{L_m}{L_r}\psi_r i_{sq} \tag{3.15}$$

Motor sabit moment bölgesinde çalışıyorken rotor akısından optimum verimi elde etmek için sabit tutulur. Bu durumda rotor akısının senkron açısal hızı şu şekilde olur.

$$w_e = w_r + \frac{L_m}{\tau_r \Phi_0} i_{sq} \tag{3.16}$$

Burada $i_{sd}=I_0$, $\psi_r=L_mI_0=\Phi_0$ olup I_0 mıknatıslanma akımıdır. Sistemde kullanılan diğer toplam kaçak faktörü, rotor zaman sabiti, eşdeğer direnç ve eşdeğer moment sabiti gibi parametrelerin sırasıyla eşitliklerini verecek olursak,

$$\sigma = 1 - \frac{L_m^2}{L_s L_r} \tag{3.17}$$

$$\tau_r = \frac{L_r}{R_r} \tag{3.18}$$

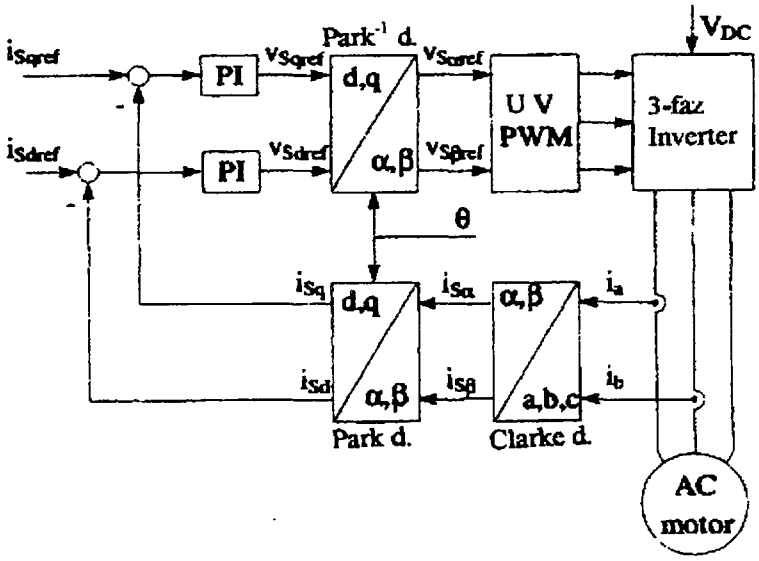
$$R_e = R_s + \frac{L_s}{\tau_r} \tag{3.19}$$

$$K_t = \frac{P}{2}\frac{L_s}{L_m}(L_m I_{sd}) \tag{3.20}$$

Bu temel denklemler bütün sayısal değerleri magnetik alandaki koordinat sistemine dönüştürür ve stator akım vektörünüde bir alan üreten bileşene (i_{sd}) bir de moment üreten bileşene (i_{sq}) ayrıştırır. Bu yapıldığında, gerçek zamanda kontrol çok basit hale gelir. Program yapısında bulunan alan zayıflatma kontrolör, gerekli alanı üretmek için alan üreten bileşeni, hız ise moment üreten bileşeni kontrol eder. Sırasıyla, alan zayıflatma kontrolörün çıkışı i_{sd} için, hız kontrolörün çıkışı i_{sq} için referanstır.

3.6 Alan Yönelmeli Vektör Kontrol İçin Temel Şema

Aşağıdaki diyagram alan yönelmeli vektör kontrolüyle moment kontrolünün temel şemasını özet bir şekilde verir (Tex. Ins. yayını, 1998; Di Gabriele vd., 1997).



Şekil 3.4 AC motor için temel alan yönelmeli vektör kontrol şeması

Şekilden de görüldüğü gibi, motor faz akımlarından iki tanesi sensörler vasıtasıyla ölçülür. Bu ölçümler Clarke dönüşüm modülüne gönderilir. Bu izdüşümün çıkışları $i_{S\alpha}$ ve $i_{S\beta}$ ile belirtilir. Bu akımın 2 bileşeni, d ve q döner referans çerçevesindeki akımı veren Park dönüşümünün girişleridir. i_{Sd} ve i_{Sq} bileşenleri i_{Sdref} (akı referansı) ve i_{Sqref} (moment referansı) referanslarıyla karşılaştırılır. Bu noktada, bu kontrol yapısı önemli bir avantaj sağlar: Basit bir şekilde akı referansını değiştirerek ve rotor akı pozisyonunu elde ederek, hem senkron hem de asenkron makinaları kontrol etmek için kullanılabilir. Senkron sürekli mıknatıslı motorlardaki gibi, rotor akısı sabittir (mıknatıs tarafından belirlenen). Bir rotor akısı üretmeğe ihtiyaç yoktur. Asenkron motorlar iş yapabilmek için bir rotor akısı oluşturulmasına ihtiyaç duyduğundan, akı referansı sıfır olmamalıdır. Bu bilindiği gibi "klasik" kontrol yapılarının büyük dezavantajlarından birine çözüm getirmeltedir. Moment komutu i_{Sqref}, bir hız alan yönelmeli vektör kontrol kullandığımızda hız kontrolörünün çıkışı olabilir. Akım kontrolörlerinin çıkışları V_{Sdref} ve V_{Sqref} 'tir; bunlar ters Park dönüşümüne uygulanır. Bunun izdüşüm çıkışları $V_{S\alpha ref}$ ve $V_{S\beta ref}$ 'tir ve bunlarda α, β sabit ortogonal referans çerçevesindeki stator vektör geriliminin bileşenleridir. Bu gerilim bileşenleri de uzay vektör PWM'in girişleridir. Bu bloğun çıkışları inverteri süren sinyallerdir. Hem Park hem de ters Park dönüşümlerinde rotor akı pozisyonuna ihtiyaç duyulur. Bu rotor akı pozisyonunun elde edilmesi, AC makina çeşidine bağlıdır (senkron veya asenkron makina). Rotor akı pozisyon kabulleri aşağıda açıklandığı gibidir.

3.6.1 Alan Yönelmeli Vektör Kontrol için giriş

Bu kontrol için temel gereksinimler 2-faz akımlarının bilgisi (motor yıldız bağlı olduğundan, üçüncü faz akımı zaten bilinir, $i_a + i_b + i_c = 0$'dan) ve rotor akı pozisyonudur.

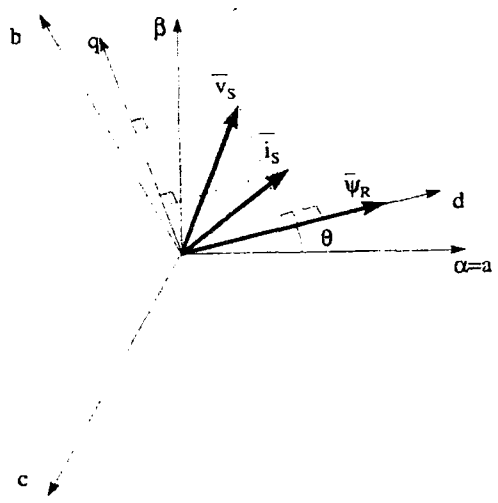
3.6.1.1 Akım Örnekleme

Ölçülen faz akımları i_a ve i_b bir A/D dönüştürücüyle örneklenir ve dönüştürülür. Alan yönelmeli vektör kontrolün doğru çalışması bu akımların doğru ölçümüne bağlıdır.

3.6.1.2 Rotor akı pozisyonu

Rotor akı pozisyon bilgisi vektör kontrolün özüdür. Gerçekte, bu değişkende bir hata varsa, rotor akısı d-ekseniyle uyarlanmıştır ve i_{Sd} ve i_{Sq} stator akımının yanlış akı ve moment bileşenleridir.

Aşağıdaki diyagram (a, b, c), (α, β) ve (d, q) referans çerçeveleri ve rotor akısının doğru pozisyonu, senkron hız da d, q referansıyla dönen stator akım ve stator gerilim uzay vektörünü gösterir.



Şekil 3.5 d, q dönen referans çerçevesinde ve bunların a, b, c ve a, b durağan referans çerçevesindeki akım, gerilim ve rotor akı uzay vektörleri

Rotor akı pozisyonunun ölçümü senkron veya asenkron motoru düşünmemiz durumunda farklıdır.

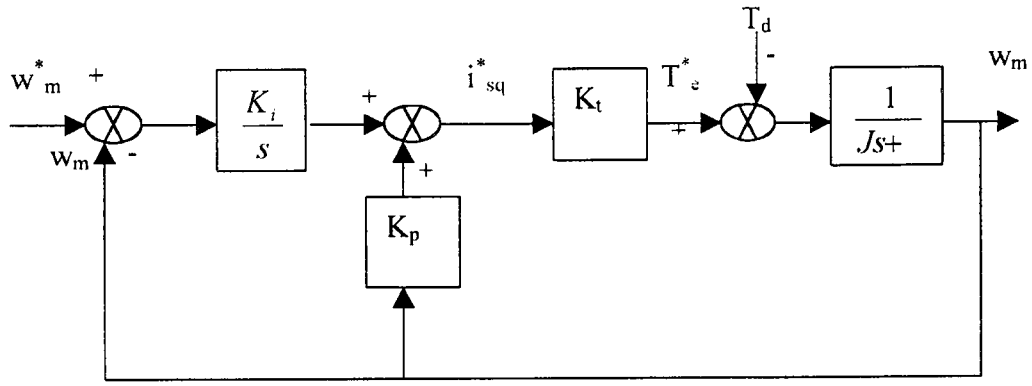
1- Senkron makinada rotor hızı rotor akı hızına eşittir. Bu durumda, θ (rotor akı pozisyonu) direkt olarak pozisyon sensörüyle veya rotor hızının integrasyonuyla ölçülür (Parasiliti, 1997).
2- Asenkron makinada rotor hızı rotor akı hızına (bir kayma hızı mevcuttur) eşit değildir. Bu durumda, θ'yı hesaplamak için özel bir metoda ihtiyaç vardır. Temel metod d, q referans çerçevesindeki motor modelinin iki denklemine ihtiyaç duyan akım modelinin kullanımıdır(Parasiliti, 1997; Di Gabriele vd., 1997, 1998).

Bu kontrol uygulaması altında AC makinaların moment ve akı bileşenlerini direkt ve ayrı olarak kontrol etmek mümkündür. Alan yönelmeli vektör kontrollü AC makinalar, bu kontrol şekliyle geçici ve kalıcı hal kontrolüne izin vererek DC makinaların her avantajını elde eder.

3.7 PI Kontrolör

Alan yönelmeli vektör kontrolünde elektriksel sürme kontrol parametreleri olarak iki sabite gereksinim duyulur: moment bileşeninin referansı I_{Sqref} ve akı bileşeninin referansı I_{Sdref}. Klasik sayısal PI kontrolör, hatayı ölçen ve kalıcı hal hatasını kontrol eden hem P terimini (K_p) hem de I terimini (K_i) doğru olarak sisteme yerleştirerek, sabit referanslara

ulaşabildiğinden, kontrolü istenen değerler moment ve akıyı geri besleme üzerinden kontrol etmek için iyi bir şekilde sisteme uygun hale getirilir. PI kontrolörün sayısal durumu aşağıdaki denklemlerde açıklanmakta olup, blok diyagramı ise şekil 3.6'da (Liaw ve Lin, 1994) verildiği gibidir.



Şekil 3.6 PI kontrolörün sürme kısmınıda içeren blok diyagramı (Liaw ve Lin, 1994)

$$T_e^* = K_t i_{sq}^* \tag{3.21}$$

transfer fonksiyonu,

$$\frac{w_m(s)}{w_m^*(s)_{T_d=0}} = \frac{K_i K_t}{Js^2 + (B + K_p K_t)s + K_i K_t} \tag{3.22}$$

bu transfer fonksiyonunu standart bir ikinci mertebeden şekille karşılaştırırsak, sönüm faktörü ve doğal frekans şu şekilde olur,

$$\xi = \frac{B + K_p K_t}{2(JK_i K_t)^{1/2}}; \quad w_d = (\frac{K_i K_t}{J})^{1/2} \tag{3.23}$$

bu transfer fonksiyonunun bant genişliği ise,

$$BW = w_d \sqrt{1 - 2\xi^2 + (2 - 4\xi^2 + 4\xi^4)^{1/2}} \tag{3.24}$$

denklem 3.24'ü (Perdikaris, 1993) cevap zamanıyla ilgili şekilde yazarsak lineer olmayan bir durum ortaya çıkar,

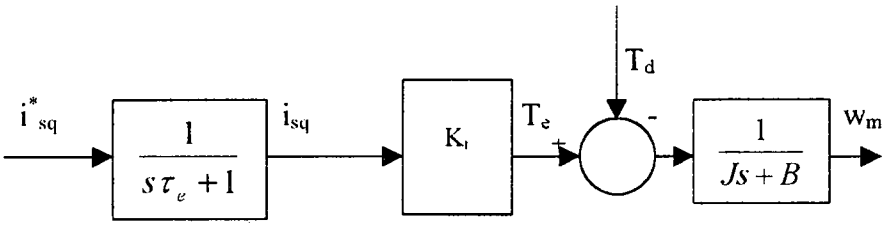
$$t_c \approx \frac{3.875}{w_d} \approx \frac{0.65 \times 3.875}{BW} \quad \xi = 1 \ için \tag{3.25}$$

böylece denklem 3.22'deki parametreler zaman bağımlı hale gelir. Buradan,

$$K_i = Jw_d^2 / K_t \quad ; \quad K_p = (2Jw_d - B) / K_t \tag{3.26}$$

olur.

PI kontrolörün asenkron motor kontrolünde gerekli olan kayma hesaplayıcı bloğunuda içeren basitleştirilmiş hali şekil 3.7'de verilmiştir.

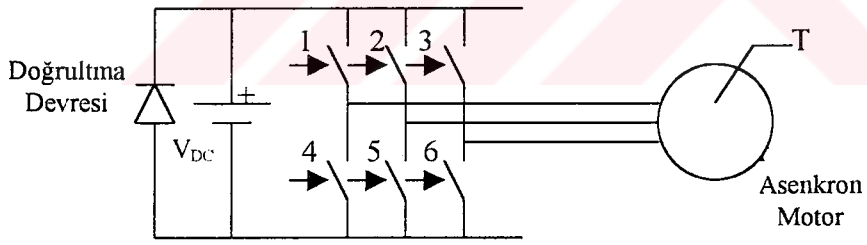


Şekil 3.7 PI kontrolörün basitleştirilmiş hali

3.8 Uzay Vektör Modülasyonu

3.8.1 3-faz İnverter

En çok bilinen bir 3-faz inverterin yapısı şekil 3.8'de verildiği gibidir (Leonhard, 1985; Bose, 1994). Burada V_A, V_B, V_C yıldız bağlı motor sargılarına uygulanan gerilimlerdir ve V_{DC}'de sürekli inverter giriş gerilimidir.



Şekil 3.8 3-faz inverter ve AC motorun temel şeması

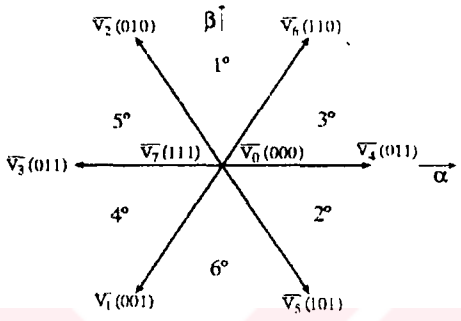
Buradaki altı anahtar BJT, GTO, IGBT gibi güç elemanları olabilir. Yapılan çalışmada IGBT modül kullanılmıştır. Bunların ON-OFF durumu aşağıdaki gibidir:

1- Anahtarların üçü daima ON ve diğer üçü de daima OFF'tur.

2- Aynı kolun üst ve alt anahtarları iki eşlenik darbeli sinyalle sürülür. Bu yönde dikey iletim mümkün değildir, yani güç anahtar geçişlerinde aynı koldaki iki eleman (IGBT) devreye giremez.

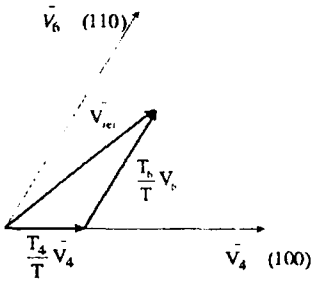
3.8.2 Uzay vektör modülasyonu

Bu modülasyon şekli hedeflenen faz gerilimlerini AC makinaya sağlar. Darbe sinyallerini üretmek için uzay vektör modülasyonda da yukarıda verilen şartlar geçerlidir ve harmonik bileşenleri minimize eder (harmonik bileşenleri makina kayıplarının büyük bir oranını teşkil eden bakır kayıpları belirler). Yukarıda belirtilen iki sınırlamayı dikkate alarak, anahtar komutları için sekiz mümkün olan kombinasyon vardır. Bu sekiz anahtar kombinasyonu sekiz adet faz-gerilim kombinasyonunu belirler. Aşağıdaki diyagram bu kombinasyonları gösterir.



Şekil 3.9 Uzay vektör modülasyonda, vektörler ve bölgeler

Vektörler, bu düzlemi altı bölgeye ayırır. Bu bölge dağılımına bağlı kalınarak gerilim referansının içinde olduğu iki komşu vektör seçilir. İki komşu temel vektörün ikili sunumları sadece 1 bit'le farklıdır, böylece üst transistörlerin sadece biri anahtarlama modeli bir vektörden komşu olana hareket ettiğinde yani bir iletim anından diğerine anahtarlama yapar. Referans vektör V_{ref}'i üçüncü bölgede varsayarak aşağıdaki duruma sahip oluruz:

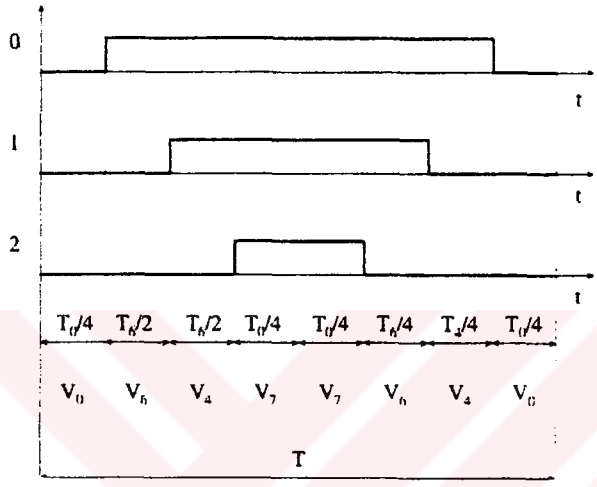


Şekil 3.10 Komşu vektörlerin birleşimi olarak referans vektör

Burada T_4 ve T_6, V_4, V_6 vektörlerinin (iletim bölgelerinin) uygulandığı zamanlar ve T_0 sıfır vektörlerinin (iletim dışı) uygulandığı zamandır. Referans gerilim (ters Park dönüşümünün çıkışı) ve örnekleme peryodu bilindiğinde, bilinmeyen T_4, T_6 ve T_0'ı aşağıdaki denklem sistemi ile ifade etmek mümkündür.

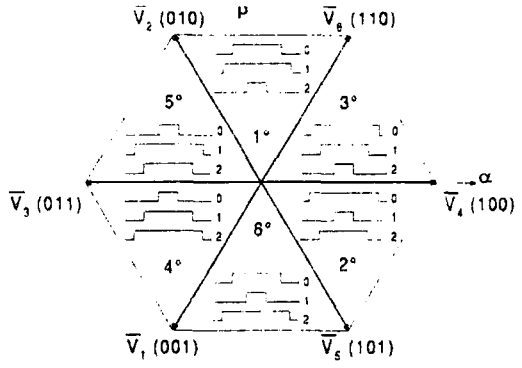
$$\begin{cases} T = T_4 + T_6 + T_0 \\ V_{ref} = \dfrac{T_4}{T} V_4 + \dfrac{T_6}{T} V_6 \end{cases} \tag{3.27}$$

Bu sınırlamalar altında referans vektörün geometrik yeri yatayları sekiz vektörün uçlarıyla oluşan bir altıgenin içidir. Üretilen uzay vektör modülasyon dalga şekilleri herbir modülasyon peryodunun ortasına yönelmiş durumda ve birbirleriyle simetriktir. Şekil 3.11 bu dalga şekillerini verir.



Şekil 3.1 3 bölgede uzay vektör modülasyonun dağılımı

Aşağıdaki diyagram ise her bir bölge için uzay vektör modülasyonun modelini gösterir.



Şekil 3.12 Uzay vektör modülasyonun altıgen modeli

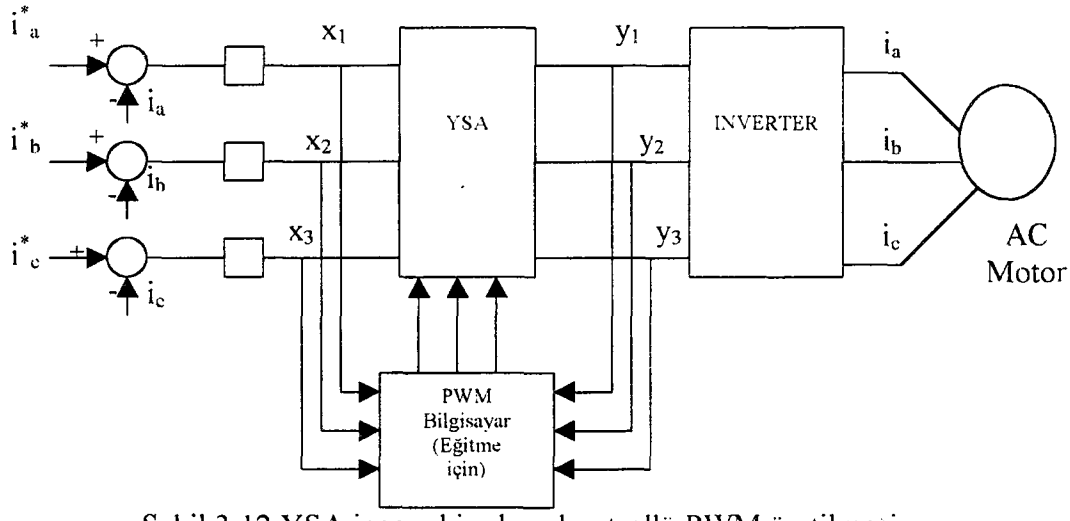
Sonuçta, uzay vektör modülasyonun girişleri referans vektör elemanları ($V_{\alpha Sr}$, $V_{\beta Sr}$)'dir ve çıkışları ise, uygun bölge sınırlama vektörlerinin her birine uygulanacak zamanlardır.

3.9 Uzay Vektör PWM Darbelerinin YSA ile Üretilmesi

Şekil 3.13 YSA yardımıyla bir akım-kontrollü PWM şemasını göstermektedir (Bose, 1994). Ağ her fazın akım hata sinyallerini K gibi bölme şeklinde kazanç ifade eden bir blok içinden alır ve inverter elemanlarını sürmek için PWM lojik sinyalleri üretir. YSA'nın çıkışı olan sigmoidal fonsiyon eşik değere ulaştığında 0 veya 1'e adapte edilir. Çıkış sinyalleri (hepsi 0 ve 1) olmak üzere inverter uzay vektör modülasyonuna dayalı sekiz mümkün çalışma durumuna sahiptir. Örneğin, bir fazdaki akımın eşik değeri +0.01'e ulaşırsa, kendi çıkışı inverterin o kolunun üst elemanını sürmek üzere 1 olmaktadır. Diğer taraftan, hata –0.01'e ulaşırsa, çıkış 0 olmakta ve aynı koldaki alt eleman sürülmektedir. Ağ bu şekilde sekiz adet giriş-çıkış örnek modelleriyle eğitilir. Ağın eğitimi için bir çalışma örneklemesi alınır ve bu örneklemeye dayalı olarak eğitme tamamlandıktan sonra artık PWM darbelerini tasarımı yapılan YSA test fazında ilgili bilgisayar üzerinden üretecektir. Çizelge 3.2'de PWM darbelerinin üretilmesi için kullanılacak olan YSA modelinin eğitilmesinde yer alan değerler verilmektedir. Çizelge 3.3 ise eğitilmesi tamamlanmış olan YSA'da test fazında sekiz mümkün çalışma durumundan birisi için elde edilen sonuçları göstermektedir. Şekil 3.14'te ise bu eğitme işlemlerinde kullanılan YSA'nın mimarisi sunulmaktadır. Bu modelde klasik Hatanın Geriyeyayılımı Algoritması kullanılmış olup 300000 iterasyon sonucunda sistem hatası olarak %0.0011 gibi çok az bir hatayla sonuca yaklaşılmıştır.

Çizelge 3.2 YSA'nın eğitilmesinde kullanılan değerler

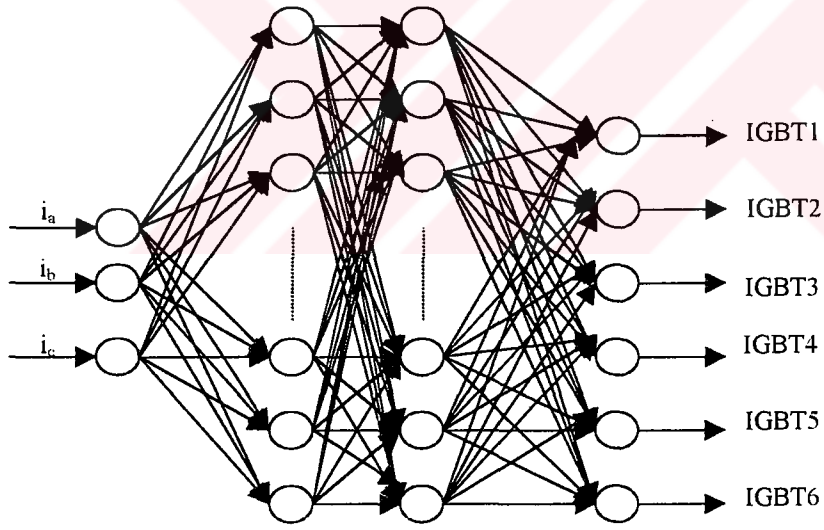
ia	ib	ic	1	2	3	4	5	6
0	0	0	0	0	0	0	0	0
0.01	-0.01	-0.01	1	0	0	0	1	1
0.01	0.01	-0.01	1	1	0	0	0	1
0.01	0.01	0.01	1	1	1	0	0	0
-0.01	0.01	0.01	0	1	1	1	0	0
-0.01	-0.01	0.01	0	0	1	1	1	0
-0.01	-0.01	-0.01	0	0	0	1	1	1
1(0)	1(0)	1(0)	0	0	0	0	0	0



Şekil 3.12 YSA içeren bir akım-kontrollü PWM üretilmesi

Çizelge 3.3 YSA test fazında sekiz durumdan birisi için elde edilen sonuçlar

ia	ib	ic	1	2	3	4	5	6
0.01	-0.01	-0.01	0.99865	0.00138	0.00009	0.00145	0.99883	1.00000



(Giriş K.) (Gizli K. 1) (Gizli K. 2) (Çıkış K.)
(10 Düğüm) (9 Düğüm)

Şekil 3.13 YSA mimarisi

4. YAPAY SİNİR AĞLARI

4.1 Giriş

Yapay sinir ağları ya da kısaca YSA; insan beyninin çalışma sisteminin yapay olarak benzetimi çabalarının bir sonucu olarak ortaya çıkmıştır. En genel anlamda bir YSA insan beynindeki birçok nöronun (sinir hücresinin), ya da yapay olarak basit işlemcilerin birbirlerine değişik etki seviyeleri ile bağlanması sonucu oluşan karmaşık bir sistem olarak düşünülebilir. Önceleri temel tıp birimlerinde insan beynindeki nöronların matematiksel modelleme çabaları ile başlayan çalışmalar, geçtiğimiz on sene içerisinde, disipline bir şekil almıştır. YSA bugün fizik, matematik, elektrik ve bilgisayar mühendisliği gibi çok farklı bilim dallarında araştırma konusu haline gelmiştir. YSA'nın pratik kullanımı genelde, çok farklı yapıda ve formlarda bulunabilen informasyon verilerini hızlı bir şekilde tanımlama ve algılama üzerinedir. Aslında mühendislik uygulamalarında YSA'nın geniş çaplı kullanımının en önemli nedeni, klasik tekniklerle çözümü zor problemler için etkin bir alternatif oluşturmasıdır. Çünkü bilgisayarlar insanın beyinsel yeteneğinin en zayıf olduğu çarpma, bölme gibi matematiksel ve algoritmik hesaplama işlemlerinde hız ve doğruluk açısından yüzlerce kat başarılı olmalarına rağmen insan beyninin öğrenme ve tanıma gibi işlevlerini hala yeteri kadar gerçekleştirememektedir. Çizelge 4.1'de bilgisayar ile insan beyni arasındaki çalışma sistem yapısı karşılaştırmalı olarak verilmiştir.

Çizelge 4.1 Bilgisayar ile insan beyni arasındaki çalışma sistem yapısının karşılaştırılması

BİLGİSAYAR	İNSAN BEYNİ
Sayısal	Analog
Seri	Paralel
Komut Kümeli	Bilgiye Adapte Olma
Yanlış Hesaplamalar Sonucu Etkiler	Birimlerin Ana İşlemlere Etkisi Azdır
Giriş Verilerindeki Hatalar Sonucu Etkiler	Giriş Verilerindeki Hatalara Her Zaman Duyarlı Değil

4.2 YSA'nın Tanımı ve Modeli

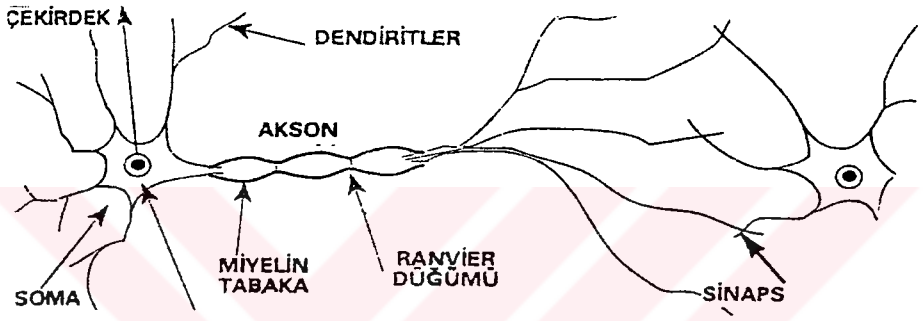
4.2.1 YSA'nın tanımı

YSA paralel dağılmış bir bilgi işleme sistemidir. Yani, YSA'nın temelinde, zeka gerektiren işlemlerden oluşan bilgi işleme işlevi vardır. Bu sistem tek yönlü işaret kanalları (bağlantılar) ile birbirine bağlanan işlem elemanlarından oluşur. Çıkış işareti bir tane olup isteğe göre çoğaltılabilir. YSA yaklaşımının temel düşüncesiyle, insan beyninin fonksiyonları arasında benzerlik vardır. Bu yüzden YSA sistemine insan beyninin modeli denilebilir. YSA çevre şartlarına göre davranışlarını şekilleyebilir. Girişler ve istenen çıkışların sisteme verilmesi ile

kendisini farklı cevaplar verebilecek şekilde ayarlayabilir. Ancak son derece karmaşık bir içyapısı vardır. Onun için bugüne kadar gerçekleştirilen YSA; biyolojik fonksiyonların temel nöronlarını örnek alarak yerine getiren kompoze elemanlar olmuştur.

4.2.2 Nöronun biyolojik yapısı ve nöron modeli

İnsanın bilgi işleme olayı beyninde gerçekleşir. Gerçektende en karmaşık sinir ağı Cerebral Cortex denilen "beyin"dir. Sinir sisteminin en basit yapısı nöronlardır. Beyinde yaklaşık olarak 0^{10} sinir hücresi vardır. Yine hücre başına bağlantı sayısı ise 0^{4} mertebesindedir. Beyin için çalışma frekansı 100 Hz'dir. Fiziksel boyutları ise 1.3 kg ve 0.15 m^2 kesitlidir. Vücudun değişik yerleri ile bilgi alışverişi yapan nöron hücresidir. Şekil 4.1 de basit bir nöron hücresi görülmektedir.

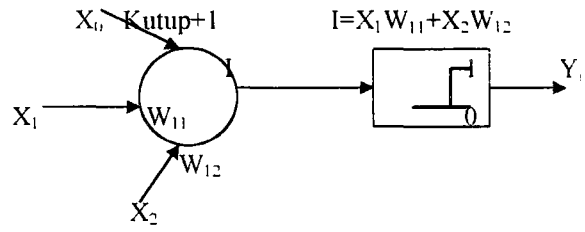


Şekil 4.1 Basit bir nöron yapısı

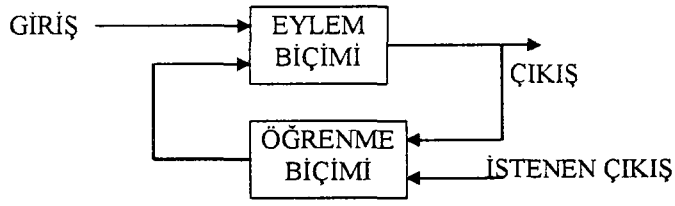
Nöron, soma adı verilen hücre gövdesi dentrit denilen kıvrımlı uzantılar ve somanın dalları sayesinde nöronu dallarına bağlayan tek sinir fiberli aksondan oluşur. Dendrit'ler hücreye gelen girişleri toplarlar. Dendrit tarafından alınan işaretler hücrede birleştirilerek bir çıkış darbesi üretilip üretilemeyeceğine karar verilir. Eğer bir iş yapılacaksa üretilen çıkış darbesi aksonlar tarafından taşınarak diğer nöronlarla olan bağlantılara veya terminal organlara iletilir. Beyindeki korteksde her nöronun bir karşılığı vardır. Bir nöronun çıkışı ona bağlı olan bütün nöronlara iletilir. Fakat korteks, işin yapılabilmesi için hangi nöron harekete geçirilecekse, sadece ona komut gönderir.

Somanın içinde ve çevresinde sodyum, kalsiyum, potasyum ve klor iyonları vardır. Potasyum yoğunluğu nöronun içinde, sodyum yoğunluğu dışındadır. Somanın zarı elektriksel olarak uyarılınca (söz konusu uyarı genellikle bir gerilim düşmesidir) zar, Na ve Ca gibi diğer iyonların içeri geçmesine izin verir ve somanın iç durumunu değiştirir. Nöronlar arasındaki bağlantılar hücre gövdesinde veya "sinaps" adı verilen dendritlerdeki geçişlerde olur. Yardımcı bir benzetme aksonlarla, dendritleri elektrik sinyallerini nörona ileten değişik empedansdaki yalıtılmış iletken olmasıdır. Sinir sistemi milyarlarca nöron ile tek bir nörondan çıkan aksonun 10000 kadar diğer nöronu bağlayan bir ağdır. Sinapslarla düzeltilen işaretleri

taşıyan aksonlar ve dendritlerle içiçe geçmiş nöronlar bir sinir ağı oluştururlar. Şekil 4.2'de en basit formda gösterilen nöron modeli, bir eşik birimi olarak algılanabilir. Şekil 4.3'de ise YSA'nın genel blok şeması gösterilmektedir.



Şekil 4.2 Nöron modeli

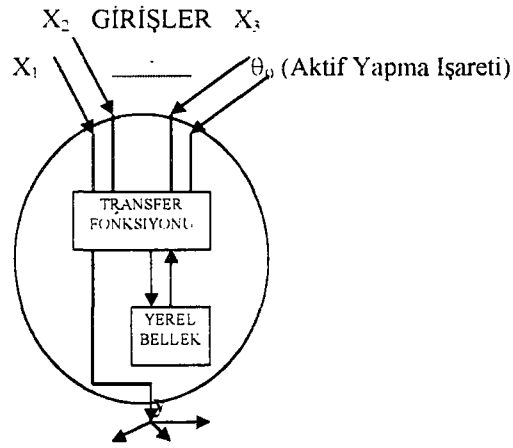


Şekil 4.3 YSA'nın genel blok şeması

Eşik birimi, çıkışları toplayan ve sadece girişin toplamı iç eşik değerini aştığında bir çıkış üreten işlem elemanıdır. Bir eşik birimi olarak nöron sinapslarındaki işaretleri alır ve hepsini toplar. Eğer toplanan işaret gücü eşiği geçecek kadar güçlü ise diğer nöronları ve dendritleri uyaran akson boyunca bir işaret gönderilir. Kesişen dendritlerden gelen sinapslarla kapılanan bütün işaretleri soma toplar. Toplam işaret daha sonra nöronun iç eşik değeri ile karşılaştırılır ve eşik değerini aşmışsa aksona bir işaret yayar. YSA, bu basit nöronların (düğümlerin ya da ünitelerin) bağlanarak bir ağ'a dönüştürülmesiyle meydana getirilir.

4.3 YSA'nın Yapısı ve İşlem Elemanı

YSA temel olarak, basit yapıda ve yönlü bir graf biçimindedir. Her bir düğüm hücre denilen n. dereceden lineer olmayan bir devredir. Düğümler işlem elemanı olarak tanımlanır. Düğümler arasında bağlantılar vardır. Her bağlantı tek yönlü işaret iletim yolu (gecikmesiz) olarak görev yapar. Her işlem elemanı istenildiği sayıda giriş bağlantısı ve tek bir çıkış bağlantısı alabilir. Fakat bu bağlantı kopya edilebilir. Yani bu tek çıkış birçok hücreyi besleyebilir. Ağ'daki tek gecikme, çıkışları ileten bağlantı yollarındaki iletim gecikmeleridir. İşlem elemanının çıkışı istenilen matematiksel tipte olabilir. Kısmen sürekli çalışma konumunda "aktif" halde eleman bir çıkış işareti üretir. Giriş işaretleri YSA'na bilgi taşır. Sonuç ise çıkış işaretlerinden alınabilir. Şekil 4.4 'de genel bir işlem elemanı (nöron, düğüm) gösterilmiştir.

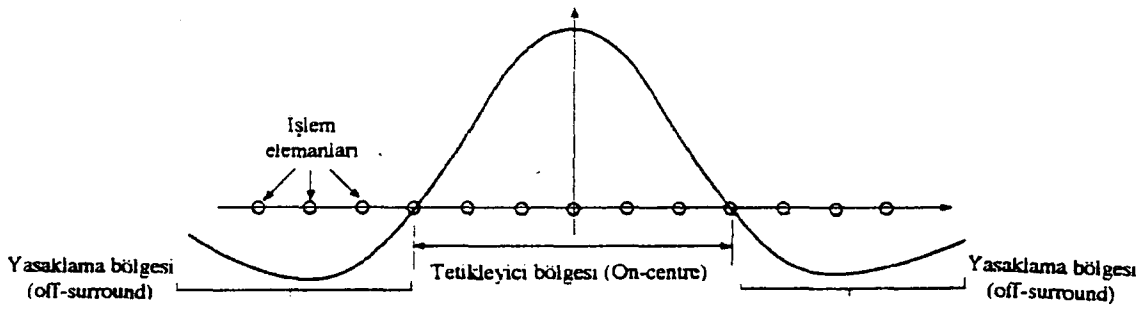


Şekil 4.4 Genel işlem elemanı yapısı

YSA birtakım alt kümelere ayrılabilir. Bu alt kümelerdeki elemanların transfer fonksiyonları aynıdır. Bu küçük gruplara "katman" layer adı verilir. Ağ katmanların birbirlerine hiyerarşik bir şekilde bağlanmasından oluşmuştur. Dış dünyadan alınan bilgi giriş katmanı ile taşınır. Bir transfer fonksiyonları yoktur. YSA transfer fonksiyonu ve yerel bellek elemanı, bir öğrenme kuralı ile giriş çıkış işareti arasındaki bağıntıya göre ayarlanır. Girişi aktif yapabilmek için yukarıda adı geçen kural ve işaretlerin bir zamanlama fonksiyonu tanımlaması gerekebilir. Kısaca bir YSA'dan beklenen görev, gerçek dünyadaki nesneler ile biyolojik sinir ağının yaptığı işlevi, benzer bir yolla yerine getirmesidir. YSA'nın giriş veri tipleri ikili (binary) 0-1 veya sürekli değerlerdir. Bu giriş durumlarından başka, işlem elemanlarına ait girişleri matematiksel olarak da sınıflamak gerekmektedir. Çünkü bir işlem elemanına gelen girişlerin bir kısmı azaltıcı uyarma girişleri olmaktadır.
Bu arttırıcı veya azaltıcı girişler "giriş sınıflarını" oluşturur.

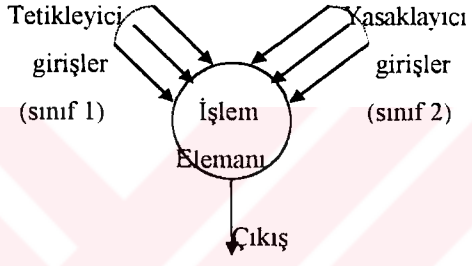
4.3.1 Giriş işareti sınıfları

İşlem elemanının transfer fonksiyonu, girişine gelen bütün işaretler için tanımlanır. Bazen değişik katman davranışlarının farklı olması tabiidir. İşaretlerin hangi bölgelerden geldiğinin bilinmesi gerekir. Değişik bölgelere göre işaretlerin sınıfları tamamlanabilir. Sıkça izlenen bir yapı ise merkezde evet/çevrede hayır (on centre/off surround) yapısıdır. Şekil 4.5'de bu yapı gösterilmektedir. Meksika şapkasına benzer bağlantı tipindedir.



Şekil 4.5 Komşu hücrelerin merkez hücreye etkisi

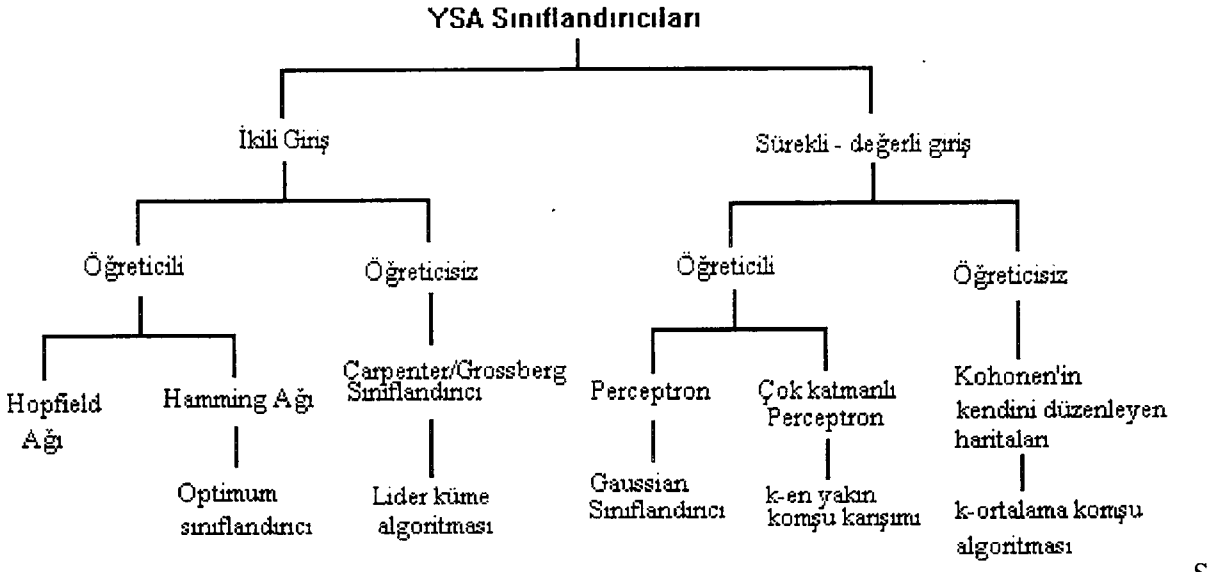
İşlem elemanı tetikleyici girişlerin kendine yakın komşu girişlerden yasaklanan girişlerini daha uzaktan alır. Böylece işlem elemanına gelen girişler sınıflarına göre değerlendirilmiş olur. Tetikleyici bölgeden gelen girişler yasaklanan sınıfı oluşturur. Şekil 4.6 böyle bir işlem elemanını gösterir.



Şekil 4.6 Tetikleyici ve yasaklanan girişlere sahip bir işlem elemanı

Bir işlem elemanına gelen girişler matematiksel tiplerine göre etiketlendirilerek sınıflandırılır. YSA, giriş veri tiplerine göre ikili giriş (0,1) ve sürekli değerli giriş olmak üzere aşağıdaki gibi sınıflandırılır (Şekil 4.7).

Burada giriş işareti olarak seçilen I , w ve yük olarak kullanılan DC motordan gelen yükün P sayısal değerleri, bu değerlerin ölçümler boyunca okunması sırasında sürekli-değer de (reel sayı) olduğundan, sınıflandırıcı olarak öğreticili öğrenmeye sahip olan çok katmanlı perseptrona bağlı olarak ileri-besleme sinir ağı ve hızlı hatanın geriye-yayılımı algoritması (Fast Backpropagation Algorithm) kullanılmıştır. Bu arada klasik hatanın geriye-yayılımı algoritmasıyla yapılan eğitme sonuçları da karşılaştırmalı olarak sunulmuştur.



Ş

Şekil 4.7 YSA Sınıflandırıcıları

4.3.2 Bağlantı geometrileri

Bağlantılarda taşınan işaret verisinin cinsi tanımlanmalıdır. Bağlantı geometrisi YSA için çok önemlidir.bağlantı işareti her cinsten olabilir. Bağlantının nerede başlayıp nerede bittiğini bilmesi gerekir. 1'den N'e kadar olan bir işlem elemanı kümesinin bağlantıları aşağıda tanımlandığı gibi NxN boyutlu matris biçiminde gösterilebilir.

$$\left[w_{ij}\right] = \begin{matrix} w_{11} & w_{12} & . & . & w_{1n} \\ w_{21} & w_{22} & . & & w_{2n} \\ . & . & & . & \\ & . & . & . & . \\ w_{n1} & w_{n2} & . & . & w_{nn} \end{matrix}$$

$w_{ij} = w_{ji} = 1 \Leftarrow i$. iþlem elemaný jiþlem elemanýna bað

$w_{ij} = w_{ji} = 0 \Leftarrow$ baðlý deðil

En fazla N^2 bağlantı olur. Bağlantılar çeşitli geometrik bölgeler arasında demetler halinde düşünülebilir. Bu bağlantı demetlerinin uyması gereken kurallar şunlardır.

1- Bağlantı demetini oluşturan işlem elemanları aynı bölgeden çıkmalıdır.

2- Bağlantı demetinin işaretleri aynı matematiksel tipten olmalıdır.

3- Bağlantı demetinin işaretleri aynı sınıftan olmalıdır.

4- Bağlantı demetinin bir seçim fonksiyonu (σ) olmalıdır.

$\sigma : T \rightarrow 2^S$ T:Hedef belgesi S:kaynak bölgesi

Hedef bölgesindeki her işlem elemanı kaynak bölgesindeki her elemana giderse "tam" (full) bağlıdır. (örn:çok katmanlı perceptron). Eğer her hedef bölgesi elemanı N kaynak bölgesi

elemanına bağlı ise " düzgün dağılmış" (uniform) olasıldır. Ayrıca her bir elemana, yine bir kaynak elemanı bağlı ise buna "bire-bir" bağlı denir

4.3.3 Ağ tipleri

Üç Çeşit Ağ Tipi Vardır

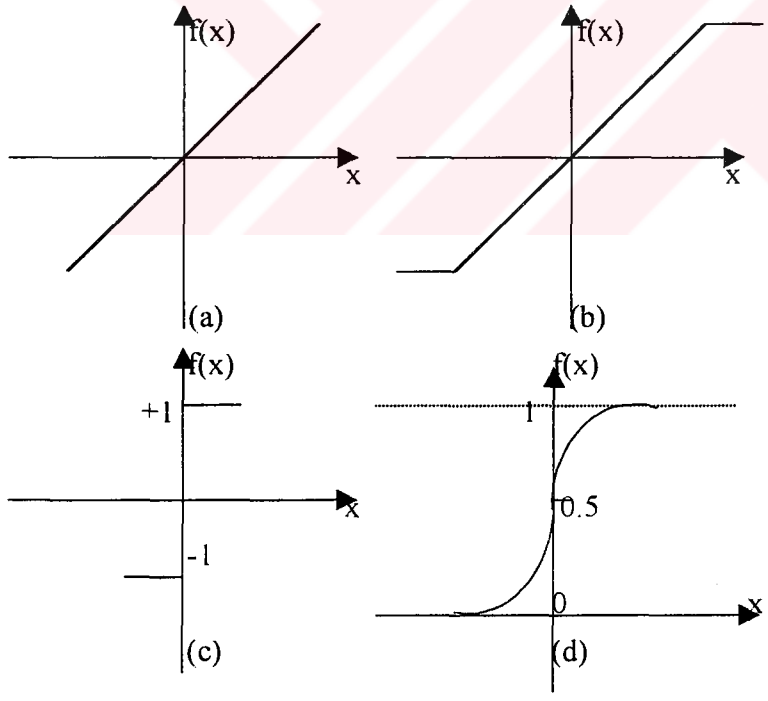
1- İleri beslemeli ağ: Her bir katmandaki hücreler sadece bir önceki katmanın hücrelerince beslenir.

2- Kaskat bağlantılı ağ: Hücreler sadece önceki katmanlardaki hücrelerce beslenir.

3- Geri beslemeli ağ: En az bir hücre sonraki katmanlardaki hücrelerce de beslenir. (Bu çalışmada hem ileri, hem de geri beslemeli ağ tipi birlikte uygulanmalıdır).

4.3.4 Eşik fonksiyonları

Transfer veya işaret fonksiyonları olarak da adlandırılan eşik fonksiyonları, muhtemel sonsuz domen girişli işlem elemanlarını önceden belirlenmiş sınırda çıkış olarak düzenler. Dört tane yaygın eşik fonksiyonu vardır. Bunlar, rampa, basamak ve sigmoid fonksiyonudur. Şekil 4.8'de bu fonksiyonlar gösterilmiştir.



Şekil 4.8 Sıkça kullanılan dört eşik fonksiyonu

Şekil 4.8 (a)'da gösterilen lineer fonksiyonun denklemi aşağıdaki gibidir.

$$f(x) = \alpha.x$$

α: işlem elemanının x aktivitesini ayarlayan reel değerli bir sabittir. Lineer fonksiyon [-τ,+τ] sınırları arasında kısıtlandığında (b)'deki rampa eşik fonksiyonu olur ve denklemi;

$$f(x) = \begin{cases} +\tau : \text{eğer } x \geq \tau \text{ ise} \\ x : \text{eğer } |x| < \tau \text{ ise} \\ -\tau : \text{eğer } x \leq -\tau \text{ ise} \end{cases}$$ şeklini alır.

+τ (-τ) işlem elemanının maksimumu (minimumu) çoğu zaman doyma seviyesi olarak adlandırılan çıkış değeridir. Eğer eşik fonksiyonu bir giriş işaretine bağlı ise yaydığı +τ giriş toplamı pozitif, bağlı değilse eşik basamak fonksiyonu [-δ] olarak adlandırılır. Şekil 4.8 (c), basamak eşik fonksiyonunu gösterir ve denklemi;

$$f(x) = \begin{cases} +\tau : \text{eğer } x > 0 \text{ ise} \\ -\delta : \text{diğer durumlar} \end{cases}$$ şeklindedir.

Son ve en önemli eşik fonksiyonu (bu çalışmada kullanılan) sigmoid fonksiyonudur. Şekil 4.8 (d) de gösterilen S biçimindeki sigmoid fonksiyonu; seviyeli, lineer olmayan çıkış veren, sınırlı, monoton artan fonksiyondur. Denklemi;

$f(x) = \frac{1}{1+e^{-x}}$ biçimindedir

Her işlem elemanı kendisine verilen yerel veriye göre, kendisini ayarlayacak bütün YSA'nın enformasyon bölgesinin öğrenmesini sağlar. (Enformasyon bölgesi olasılık-yoğunluk fonksiyonu ile de tanımlanabilir). Enformasyon bölgesi birçok uygulamada, gerçek değerin "0" ile "1" arasında normalize edilmesi gerekir. (Normalize etmek:gerçek değeri 85 olan bir girişi 0.85 şeklinde ağa uygulamaktır.) Normalizasyon aynı anda bütün girişlere uygulanabilir.

4.3.5 Ağırlık uzayı

Bir çok YSA öğrenme işlemi, işlem elemanlarının ağırlığı değiştirilerek sağlanır. Böylece tanımlanan ağırlık değiştirilerek öğrenmede iyi bir model kullanıp, ağırlıkların bu modele göre değiştirilmesi esastır. Basit bir matematiksel model olarak herbir işlem elemanının "n" adet gerçek ağırlığı olduğu düşünülerek ve N adet işlem elemanı gözönüne alınırsa;

$$w = (w_{11}, w_{12}, \ldots, w_{1n}, w_{21}, w_{22}, \ldots, w_{2n}, \ldots, w_{N1}, w_{N2}, \ldots, w_{Nn})^T$$

$$w = (w_1^T, w_2^T, w_3^T, \ldots, w_N^T)$$

$w_1, w_2, \ldots, w_N$: işlem elemanlarının ağırlık vektörleridir.

$$w_1 = \begin{matrix} w_{11} \\ w_{12} \\ . \\ . \\ w_{1n} \end{matrix} \quad \ldots\ldots \quad w_N = \begin{matrix} w_{N1} \\ w_{N2} \\ . \\ . \\ w_{Nn} \end{matrix}$$

YSA ağırlık vektörü N, n boyutlu orkid uzayında yayılır. YSA'nın enformasyon işleme performansı, ağın ağırlık vektörünün belirli bir değeri ile bulunacaktır.
Hata değişimini inceleyen iki çeşit kural vardır.

1- Hata düzeltme kuralları ,
2- Gradyen kuralları.

Hata düzeltme kuralları; Her bir giriş örüntüsünde ağırlıkları yeniden ağırlayarak çıktı hatasını en aza indirmeye çalışırlar. Gradyen kurallarında ise, ağırlıklar yeniden ayarlanarak ortalama karesel hatayı (MSE) en aza indirilmeye çalışılır.
Bu noktada gradyen kuralından kısaca bahsedecek olursak, hatayı düzeltmenin (yani minimize etmenin) geometrik bir yorumunu yapmak mümkündür. Bunu yapabilmek için ağırlıkların mümkün olan tüm değerleri, hataların kareleri toplamına karşı gelecek şekilde üç boyutlu koordinat sisteminde çizilir. Bu çizim sonunda hata yüzeyi küresel bir top şeklindedir. Bu yüzeyi bir tasa da benzetmek mümkündür. Tasın en alt kısmı hataların kareleri toplamının en küçük değerlerine karşı gelmektedir. Eğitme sırasında amaç ağırlıklar kümesinin en iyisini bulmak olan, en alt kısmına ulaşmaktır. Geriye-yayılım algoritması o andaki ağırlıklar yerine, yüzey hatasının eğimini hesaplayarak amacına ulaşır. Daha sonra da bu ağırlıkları tasın alt kısmına doğru artımsal olarak değiştirir. İşte bu artımsal olarak tasın üst kısmından alt kısmına doğru ilerleme işlemine "gradyen iniş" denir.
Ağırlık vektörü ile çalışan YSA'da önemli noktalardan birisi, bir öğrenme kuralı geliştirip, enformasyon bölgesi kullanarak (eşik fonksiyonu ile) ağırlık vektörü "w" iyi istenilen YSA performansı verecek noktaya yöneltmektir. Genellikle öğrenme kuralı için bir performans ya da maliyet fonksiyonu tanımlanır. Minimizasyon veya maksimizasyon ile "w" vektörü bulunur. Bir performasyon çeşidi olarak bilinen, MSE (karesel ortalama hata) şu şekilde tanımlanır.

$$F(w) = \oint_A |f(x) - G(x,w)|^2 \rho(x) dv(x)$$

Amaç F'i küçültmeye çalışmaktır.
y=G(w,x):sistemin giriş çıkış fonksiyonu.
y:çıkış işareti vektörü
x:giriş işareti vektörü
w:ağırlık vektörü
ρ(x):olasılık yoğunluk fonksiyonu

4.4 YSA'da Eğitme (Training)

4.4.1 Eğitme algoritmaları

Eğitme algoritmaları YSA'nın ayrılmaz bir parçasıdır. Eğitme algoritması eldeki problemin özelliğine göre öğrenme kuralını YSA'na nasıl adapte edeceğimizi belirtir. Üç çeşit eğitme algoritması yaygın olarak kullanılmaktadır.

1- Öğreticili eğitme (supervised traning).

2- Skor ile eğitme (graded training).

3- Kendini düzenleme ile eğitme (self-organization training)

Öğreticili eğitmede, elimizde doğru örnekler vardır. Yani $(X_1.X_2,.....X_n)$ şeklindeki giriş vektörünün, $(y_1,y_2,...,y_n)$ şeklindeki çıkış vektörü, tam ve doğru olarak bilinmektedir. Herbir (x_1,y_1), $(x_2,y_2),...,(x_N,y_N)$ çifti için ağ doğru sonuçları verecek şekilde seçilen bir öğrenme kuralı ile beraber eğitilir.

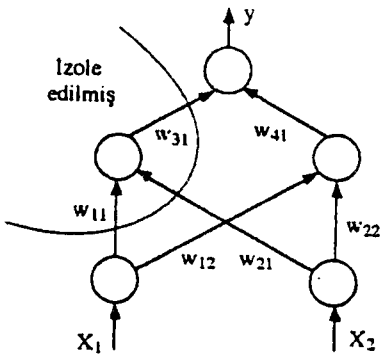
Ne tür eğitme yöntemi kullanılırsa kullanılsın, herhangi bir ağ için gerekli karakteristik özellik, ağırlıkların verilen eğitme örneğine nasıl ayarlanacağının belirtilerek öğrenme kuralının oluşturulmasıdır. Öğrenme kuralının oluşturulması için bir örneğin, ağa defalarca tanıtılması gerekebilir. Öğrenme kuralı ile ilişkili parametreler ağın zaman içinde gelişme kaydetmesiyle değişebilir.

4.4.2 Bellek

YSA'nın önemli bir özelliği bilgiyi saklama şeklidir. Bağlantı ağırlıkları YSA bellek biçimleridir. Ağırlıkların değerleri ağın o anki bilgi durumunu temsil eder. Mesela; bir giriş/istenen çıkış çiftinin belirtilen bilgi parçası ağın içinde birçok bellek biçimine dağıtılmıştır. Bellek üniteleri ile diğer saklı bilgiler, bu bilgiyi paylaşırlar. Bazı YSA bellekleri ilişkilidir. Öyleki eğitilen ağa bir kısmı uygulanırsa, ağ bu girişe belleğindeki en yakın çıkışı bu giriş için seçer ve tam girişe bağlı çıkış ortaya çıkar. Eğer YSA oto-ilişkili ise, kısmi giriş vektörlerinin ağa verilmesi bu girişlerin tamamlanması ile sonuçlanır. YSA belleğinin yapısı; eksik, gürültülü ve tam seçilemeyen bir giriş uygulandığı zaman bile mantıklı çıkış üretmeye uygundur. Bu kurala "genelleme " adı verilir. Bir genellemenin kalitesi ve anlamı, uygulama çeşidine, ağın tipine ve karmaşıklığına dayanır. Lineer olmayan çok katmanlı ağlar (özellikle geri beslemeli ağlar) gizli katmandaki özelliklerden öğrenirler ve bunları çıkışlar üretmek için birleştirirler. Gizli katmandaki bilgi, yeni giriş örüntülerine akılcı çözümler oluşturmak için kullanılabilir.

4.4.3 Hata toleransı

Klasik hesaplama sistemleri çok az bir zarardan bile etkilenir. YSA için durum farklıdır. Bu farklılık YSA'nın hata toleranslı olmasıdır. Işlem elemanlarının az da olsa zarar görmesi sistemin bütününü etkiler. YSA paralel dağılmış parametreli bir sistem olduğundan her bir işlem elemanı izole edilmiş bir ada olarak düşünülebilir. Şekil 4.9 'da çok katmanlı perseptron (MLP) için bu durum gösterilmiştir.



Şekil 4.9 MLP'nin izole edilmiş hali

Daha çok işlem elemanın zarar görmesi ile sistemin davranışı biraz daha değişir. Performans düşer ama sistem hiç bir zaman durma noktasına gelmez. YSA sistemlerinin hata toleranslı olmasının nedeni bilginin tek bir yerde saklanmayıp, sisteme dağıtılmasıdır. Bu özellik sistemin durmasının önemli bir zarara neden olacağı uygulamalarda önem kazanır.

4.4.4 YSA kullanımının sebepleri

1- YSA'lar verilerden haraketle, bilinmeyen ilişkileri akıllıca hemen ortaya çıkarabilmektedir. Bu özellikleri, uygulama açısından son derece önemlidir. Ayrıca veri toplama için bir ön sorgulama ya da açıklama gerekmemektedir.

2- YSA'lar çözüm olarak genelleştirilebilir. Bir örnekten hareketle, diğer örneklerdeki benzerlikleri doğru olarak anlayabilirler. Genelleştirme yapılabilmesi bu bakımdan çok iyi bir özelliğidir, çünkü gerçek dünya verilerinde sürekli olarak gürültü ve bozucu etkiler mevcuttur.

3- YSA'lar lineer olmayan yapıdadır. Bu özellikleri nedeni ile daha karmaşık problemleri lineer tekniklerden daha doğru çözerler. Non-lineer davranışlar hissedilir, algılanır, bilinebilir, ancak bu davranışları ya da problemleri matematiksel olarak çözmek zordur.

4- YSA'lar son derece paralelliğe sahiptir. Bağımsız işlemleri aynı anda çok hızlı yürütebilirler. Paralel donanımlar yapıları gereği YSA'lara uygun olduğundan kendisine alternatif çözüm metodlarından daha elverişlidir.

4.4.5 YSA'nın klasik yazılımlar ile karşılaştırılması

YSA'lar, ısrarla belirtildiği gibi önceden tahmin, örnek değerlendirme ve gruplama işlemlerinde etkilidir. Aynı işlemleri klasik bir bilgisayar programı ile yapmak da mümkündür. YSA'lar, açıkça kuralları bulunmayan veya anında optimizasyon kısıtlamaları koyan uygulamalar için idealdir. YSA için endüstriyel kontrol işlemleri oldukça yaygın uygulama alanlarıdır. Burada kurallar çok sık değişmez ve üstelik iyi bir tarafı da öteki çalışma koşullarına ait verilerin bol oluşudur.

Klasik programlar da belirli bir görev için yazılmış bir yazılım yıllarca aynı tip işi yapar. Örneğin, bir mühendislik programı olan Autocad ve benzerleri ile sürekli aynı hizmetler yapılabilir.

YSA'ların uygulamadaki dezavantajlarını sıralayacak olursak,

1- Bir problemin çözümünde çok uygun bir çözüm bulamayabilirler ve çözümde hata yapabilirler. Buna sebep ise eğitelecek bir fonksiyonun bulunamamasıdır. Fonksiyon bulunsa bile yeterli veri sağlanamayabilir.

2- Sonuç almak yüzlerce giriş örneğinin hesaplanmasına bağlı olabilir. Ayrıca hangi ağırlığın sonucu nasıl etkileyeceğini tahmin etmek zordur.

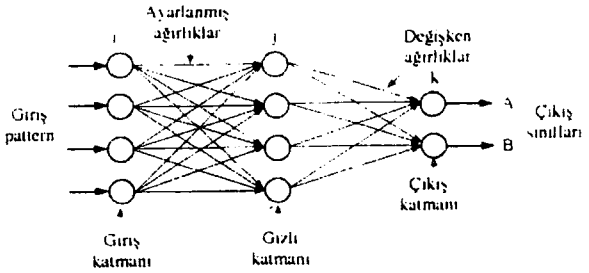
3- YSA'larla bir dizi işlem yapmak, bunları eğitmek yavaş ve pahalı olabilir. Maliyeti arttıran sebeplerden ilki eğitme verilerinin toplanması ve değerlendirilmesidir. Doğru değerleri bulmak için deneyler yapmak gerekebilir.

4- Bir YSA'nın kalitesi ve kapasitesi uygulamadaki hızı ile orantılıdır. Öyleki düğüm sayısındaki az bir artış bile yürütme zamanında çok daha fazla artışa sebep olur. Örneğin 100 düğümde 10 000 bağlantı var ise, standart bir mikroişlemci bunu 10 000 000 çarpma-saklama işlemi yaparak hesaplanır. Böylece ağdan saniyede 1000 geçiş olur. Eğer 300 düğüm var ise aynı işlemci ancak 100 kere geçiş yapmayı sağlayabilir. Kısacası düğüm sayısı 3 kat arttığında cevap süresi 10 kat azalır.

Yinede YSA'ların diğer çözümlerden daha doğru çözümler ürettikleri de bir gerçektir. Çünkü bu sakıncalı durumlar teorik olarak söz konusudur.

4.5 Çok Katmanlı Perceptron (Multi-Layer Perceptron)

Çok katmanlı perceptron giriş ve çıkış katmanları arasında birden fazla katmanın kullanıldığı YSA sistemleridir. Gizli katman (hidden layer) olarak isimlendirilen bu katmanlarda , düğümleri aracısız giriş olmayan ve aracısız çıkış veremeyen üniteler vardır. Şekil 4.10 'de çok katmanlı perceptronun genel yapısı verilmiştir. Şekil 4.11'de ise çok katmanlı perceptronda gizli katmanın etkisi gösterilmiştir.



Şekil 4.10 Çok katmanlı perceptron yapısı

YAPI	Karar Bölgeleri tipi	XOR Problemi	Bölgelere Dayalı sınıflar	En iyi ayırdığı bölge şekilleri
Tek katman (a)	Doğrusal ayrıştırılabilir. Ayrıcaklı veya işlevini gerçekleyemez.			
İki katman (b)	Konveks (iç bükey) açık veya kapalı bölgeler.			
Üç katman (c)	İç bükey olmayan hatta bağlantılı olmayan bölgeler			

Şekil 4.11 Çok katmanlı perceptronda gizli katmanın rolü

İki katmanlı ağlarda veriler giriş katmanı tarafından kabul edilirler. Ağ içinde yapılan işlemler sonucunda çıkış katmanında oluşan sonuç değer işlenen cevap ile karşılaştırılır. Bulunan cevap ile istenen cevap arasındaki herhangi bir ayrılık varsa ağırlıklar bu farkı azaltacak şekilde yeniden düzenlenir. Girişteki değer, ağırlıklar uygun noktaya ulaşana kadar değişmez. Hesaplanan çıkışlar istenilen cevaplarla karşılaştırılarak sonuçta gerekirse hata işaret belirtilir. Hata işareti gizli birimlerden çıkış birimine olan ağırlıkları değiştirmekte kullanılır. Ama bunu yaparken giriş katmanından gizli katmana gelenin değiştirilip değiştirelemediğini düşünmek gerekir. Gizli birimlerden ne tür bir çıkış istendiği bilinemeyeceği için gizli birimlerin çıkışında hata işareti verilmesi kolay bir şey değildir. Bunun yerine her bir birimin, çıkış biriminin hatalarına olan etkisi bilinmelidir. Bu hatalı birim için gizli birime bağlı olan çıkış birimlerinin hata işaretlerinin ağırlıkları toplamı alınarak yapılır. Çok gizli katmana sahip sistemlerde her sistemin hata işaretleri, bir önceki katmanın düzeltilmiş işaretlerinden çıkartılarak işlem tekrarlanır. Sonuç olarak ağırlık düzeltme işlemi çıkış seviyesine bağlı ağırlıklardan başlar ve işlem ters yönde, giriş seviyesine varana kadar devam eder. Sonuçta sistem hatalar yapar, ama bu hatalardan birşeyler öğrenip isteneni bulana kadar işleme devam

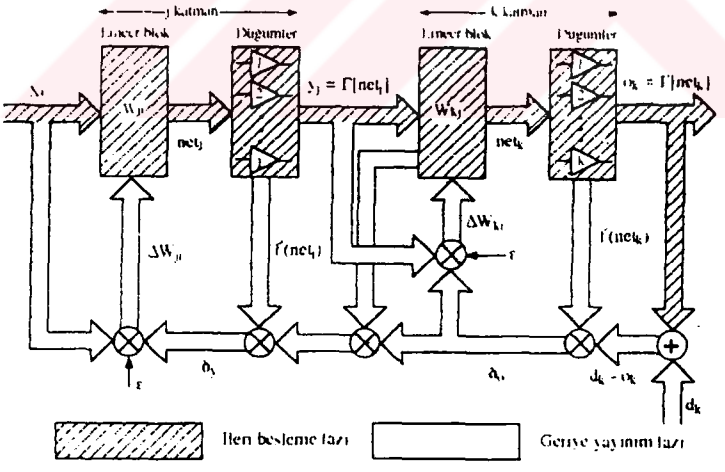
eder. Bu yönteme "hatanın geriye yayılması algoritması" (Error back-propagation algorithms) denir. Yapılan çalışma da büyük değerli sayısal verilerle işlem gerçekleştirildiğinden-bazı ufak dezavantajlarıda dikkate alınarak- sonuca daha hızlı yaklaşan "hızlı hatanın geriye yayılımı algoritması" kullanılmıştır. Kısaca klasik hatanın gerye-yayılımı algoritması hakkında bilgi verildikten sonra diğer alt bölümler de hızlı hatanın geriye-yayılımı algoritması incelenecektir.

4.6 Klasik Hatanın Geriye Yayılması Algoritması ve Genelleştirilmiş Delta Kuralı

Hatanın geriye yayılması algoritması, karesi alınmış hata fonksiyonunu minimize eden kodlu bir algoritma olup ve genelleştirilmiş delta kuralını eğitme için kullanılır. Şekil 4.12'de mimarisi gösterilen algoritma, ana hatlarıyla şöyledir:
Her bir j biriminin çıkışı o_j şu şekilde tanımlanır;

$$o_j = f(net_j) = f(x) \; ise \; net_j = \sum_j^I w_{ji} o_i + \theta_j \qquad (4.1)$$

Burada o_i; i. biriminin çıkışı w_{ji}; i biriminden j birimine bağlantının ağırlığı, θ_j; j biriminin kutbu (bias) {$\sum_i$; çıkışı j birimine akan her i biriminin toplamıdır. f(x) bir monoton artan ve türevi alınabilen fonksiyondur. Pratikte bir lojistik aktivasyon fonksiyonu olarak $f(x)=1/\ 1 + e^{-x}$ (sigmoid) daha çok kullanılır.



Şekil 4.12 Hatanın geriye yayılması algoritmasının blok diyagramı

m-boyutlu giriş örüntüleri set edildiğinde { $i_p = (i_{p1}, i_{p2} \ldots i_{pn})$; p E P}'dir. Benzer şekilde istenilen n- boyutlu çıkış örüntüleri { $t_p = (t_{p1}, t_{p2} \ldots \ldots t_{pn})$ p E P} belirtir. Burada; P:YSA uygulanan işaret şekilsel v.b. örüntüleri verir.

Bir görüntü için karesel hata (MSE) fonksiyonu E_p şu şekilde tanımlanır.

$$E_p = \frac{1}{2} \sum_{j \in \text{çıkış}} (t_{pj} - o_{pj})^2 \qquad (4.2)$$

Amaç uygun w_{ji} ve q_j seçimiyle, $E = \sum_p E_p$ toplam hatayı yeterince küçük yapmaktır. Bu amacı gerçekleştirmek için, bir $p \in P$ örüntüsü ard arda ve rasgele biçimde seçilir. Daha sonra w_{ji} ve q_j şöyle değiştirilir;

i_{pi} : Giriş işaretinin i bileşeni;

t_{pj} : Çıkış vektörünün j bileşeni;

o_{pj} : YSA uygulanan P örüntü setinin ürettiği çıkış ise;

$$\delta_{pj} = (t_{pj} - o_{pj})$$

$$\Delta_p w_{ji} = -\varepsilon\left(\frac{\partial E_p}{\partial w_{ji}}\right) \qquad (4.3)$$

$$\Delta_p \theta_j = -\varepsilon\left(\frac{\partial E_p}{\partial \theta_j}\right)$$

Burada ε : öğrenme oranı adı verilen küçük bir pozitif sabit sayılır. Şayet gizli katman yok ise; (4.3)'nin son iki denkleminin sağ tarafı hesaplanır, o zaman;

$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial o_{pj}} \frac{\partial o_{pj}}{\partial w_{ji}}$$

$$\frac{\partial E_p}{\partial o_{pji}} = -(t_{pj} - o_{pj}) = -\delta_{pj} \qquad (4.4)$$

$$o_p = \sum_i w_{ji} i_{pi} \qquad ise; \qquad \frac{\partial o_{pj}}{\partial w_{ji}} = i_{pi} \qquad (4.5)$$

elde edilir. (4.4)'ün ikinci denklemi ve (4.5) ifadelerini (4.4)'ün birinci denkleminde yerine koyarsak;

$$-\frac{\partial E_p}{\partial w_{ji}} = \delta_{pj} i_{pi} \qquad (4.6)$$

olur.

Gizli katman olduğu zaman; Hata düzeyi sadece bir minimumdan oluşmaz, aynı zamanda çeşitli minimumlar içerir. Öğrenmede en küçük minimuma ulaşılmak istenir.

Bu durumda j. düğümün lineer olmayan çıkışı;

$$o_{pj} = f_j(netp_j) \;\Rightarrow\; netp_j = \sum_i w_{ji} o_{pj} \qquad (4.7)$$

şeklindedir. Bu durumda;

$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial netp_j} \frac{\partial netp_j}{\partial w_{ji}} \;\Leftrightarrow\; \frac{\partial netp_j}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}} \sum_k w_{jk} o_{pk} = o_{pi} \qquad (4.8)$$

$$\delta_{pj} = -\frac{\partial E_p}{\partial netp_j} = -\frac{\partial E_p}{\partial o_{pj}} \frac{\partial o_{pj}}{\partial netp_j} f_j'(netp_j) \qquad (4.9)$$

İki durum var

1- o_{pj} YSA'nın çıkışı ise;

(4.4)ün ikinci denklemini (4.9)'de yerine koyarsak,

$$\delta_{pj} = (t_{pj} - o_{pj}) f_j'(netp_j) \tag{4.10}$$

bulunur.

2- Eğer gizli katmanların çıkış işaretinden bahsediliyorsa yani eleman çıkış elemanı değilse;

$$\sum_k \frac{\partial E_p}{\partial netp_k} \frac{\partial netp_k}{\partial p_j} \tag{4.11}$$

şeklinde ise,

$$\sum_k \frac{\partial E_p}{\partial netp_k} \frac{\partial}{\partial o_{pj}} \sum_i w_{ki} o_{pj} = -\sum_k \delta_{pk} w_{kj} \tag{4.12}$$

olur. Bulduğumuz son işlemi (4.9)'de yerine koyarsak;

$$\delta_{pj} = f'(netp_j) \sum_k \delta_{pk} w_{kj} \tag{4.13}$$

elde edilir. (4.12) denklemindeki (-) işareti, ağırlıkların ters yönde değiştiğini belirtir. Bütün yaptığımız işlemleri kısaca özetleyecek olursak;

1. Genelleştirilmiş Δ (delta) kuralı:

$\Delta_p w_{ji} = \varepsilon \delta_{pj} i_{pi}$

2. Çıkış katmanı elemanları için;

$\delta_{pj} = (t_{pj} - o_{pj}) f_j'(netp_j)$

3. Gizli katman elemanları için;

$\delta_{pj} = f_j'(netp_j) \sum_k \delta_{pk} w_{kj}$

olur. İşlem elemanında, transfer (eşik) fonksiyonu olarak "sigmoid" fonksiyonu kullanılırsa;

$$o_{pj} = \frac{1}{\sum_i 1 + e^{-w_{ji} o_{pi} + \theta_j}} \tag{4.14}$$

$(netp_j) = \sum_i w_{ji} o_{pi} + \theta_j$ ifadesinin türevi alınır ve gerekli kısaltmalar yapılırsa;

$$\frac{\partial o_{pj}}{\partial netp_j} = o_{pj}(1 - o_{pj}) \tag{4.15}$$

bulunur. Bunu (4.14) de yerine koyarsak, çıkış elemanı için;

$$\delta_{pj} = (t_{pj} - o_{pj}) o_{pj}(1 - o_{pj}) \tag{4.16}$$

elde edilir. (4.15)'i (4.13) de yerine koyarsak, gizli katman elemanı için;

$$\delta_{pj} = o_{pj}(1 - o_{pj}) \sum_k \delta_{pk} w_{kj} \tag{4.17}$$

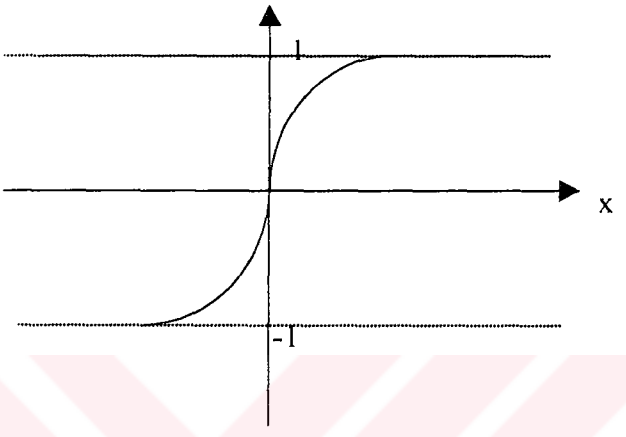
bulunur. Yukarıda toplam içerisinde gösterilen k'nın, j çıkış birimine akan herbirim k olduğuna dikkat edilmelidir. Hesaplamayı hızlandırmak için momentum terimleri (α) eklenirse, en genel halde çıkış ve gizli katman ifadeleri şu şekilde olur:

$$\begin{aligned} \Delta_p w_{ji}(t+1) &= \varepsilon \delta_{pj} o_{pi} + \alpha \Delta_p w_{ji}(t) \\ \Delta_p \theta_j(t+1) &= \varepsilon \delta_{pj} + \alpha \Delta_p \theta_j(t) \end{aligned} \tag{4.18}$$

Burada; t: öğrenme saykıllarının sayısını gösterir. (α) küçük pozitif bir sayıdır.

4.7 Hızlı Hatanın Geriye Yayılımı Algoritması (Fast Backpropagation Algorithm) ve Hızlı Delta Kuralı (Fast Delta Rule)

Şekil 4.12'de mimarisi gösterilen algoritma, tanh'e dayalı hata fonksiyonunu minimize eden bir algoritma olup, hızlı delta kuralını eğitme için kullanılır. Şekil 4.13'de ise fonksiyonun kendisi verilmektedir.



Şekil 4.13 Algoritmada kullanılan tanh(x) eşik fonksiyonu

4.7.1 Hızlı Delta Kuralı

Tek katmanlı sinir ağlarının eğitimi genellikle aşağıdaki denklemde tanımlanan nesnel fonksiyon G(λ)'nın, $G(\lambda)=\sum_{k=1}^{m} G_k(\lambda)$ şeklinde ifade edilmesine bağlıdır.

$$G(\lambda)=\lambda E+(1-\lambda)E'=\lambda\sum_{k=1}^{m}\sum_{i=1}^{n_0}\phi_2(e_{i,k})+(1-\lambda)\sum_{k=1}^{m}\sum_{i=1}^{n_0}\phi_1(e_{i,k}) \tag{4.19}$$

burada $\phi_2(x)=\frac{1}{2}x^2$ ve $\phi_1(.)$ pozitif tanımlı, konveks ve heryerde diferansiyeli alınabilir (türevi tanımlı) fonksiyon ve $\lambda \in [0,1]$. $G_k(\lambda)$ ise,

$$G_k(\lambda)=\lambda\sum_{i=1}^{n_0}\phi_2(e_{i,k})+(1-\lambda)\sum_{i=1}^{n_0}\phi_1(e_{i,k}) \quad \forall k = 1,2,.....,m \tag{4.20}$$

şeklinde tanımlıdır. Böylece, denklem 4.19, denklem 4.20 (k=1,2,....,m) de yerine konup, düzenli olarak küçültülerek minimize edilir. Gradyen iniş metodu geniş bir şekilde denklem 4.20'nin minimizasyonunu gerçekleştirmek için kullanılır. Ağın çıkışı analog ise, gradyen

inişin ağdaki snaptik ağırlıklar için güncelleştirilmiş denklemi olan 4.21'deki şekli Ek1 de açıklanmıştır.

$$w_{p,k} = w_{p,k-1} + \alpha \varepsilon^{0}_{p,k}(\lambda) x_k \tag{4.21}$$

$$\varepsilon^{0}_{p,k}(\lambda) = \lambda(y_{p,k} - \hat{y}_{p,k}) + (1-\lambda)\tanh\left[\beta(y_{p,k} - y_{p,k})\right] \tag{4.22}$$

Yine Ek1'de belirtildiği gibi, ağın çıkışı binary (ikili) snaptik ağırlıklar ise, bu da denklem 4.21'deki gibi güncelleştirilmiştir.

$$\varepsilon^{0}_{p,k}(\lambda) = (1 - \hat{y}^{2}_{p,k})(y_{p,k} - \lambda y_{p,k}) \tag{4.23}$$

Hızlı Delta Kuralı algoritması çizelge 4.2'de sunulan akış diyagramıyla özetlenebilir.

Denklem 4.24'ün minimizasyonunun alternatif bir formülasyonu binary çıkışa sahip ileri-besleme sinir ağları için ikinci-mertebeden algoritmaların gelişimine temel sağladı (Karayiannnis, 1991; Karayiannis ve Venetsanopoulas, 1991). Bu formülasyon da G(λ)'nın $G(\lambda) = \sum_{i=1}^{n_0} G_{i,m}(\lambda)$ şeklinde gözlemlenmesine dayanır.

$$G(\lambda) = \sum_{k=1}^{m} \sum_{i=1}^{n_0} y_{i,k}(y_{i,k} - \hat{y}) + \frac{1}{2}\lambda \sum_{k=1}^{m} \sum_{i=1}^{n_0} (\hat{y}^{2}_{i,k} - y^{2}_{i,k}) \tag{4.24}$$

$$G_{i,m}(\lambda) = \sum_{k=1}^{m} y_{i,k}(y_{i,k} - y_{i,k}) + \frac{1}{2}\lambda \sum_{k=1}^{m} (\hat{y}^{2}_{i,k} - y^{2}_{i,k}) \quad \forall i = 1,2,\ldots\ldots,n_0 \tag{4.25}$$

Denklem 4.24, her bir eleman $y_{i,k}$, k=1,2,.....,m olmak üzere sadece snaptik ağırlıklar matrisinin i'nci sırasına bağlı olduğundan, her bir i=1,2,.....,n_0 için 4.25'de belirtilen nesnel fonksiyonu minimize ederek minimize edilebilir. Bu sonuç snaptik ağırlıklar matrisinin her bir sırasının w_i^* , i=1,2,.....,n_0 olmak üzere nesnel fonksiyon 4.25'i minimize ederek tahmin edilebilmesini sağlar. Açık bir şekilde, $G_{i,m-1}(\lambda)$ sadece (y_k,x_k), k=1,2,.....,m-1 şeklinde birleşmelere bağlıdır. Bu yüzden, ağ, $G_{i,m-1}(\lambda)$'yi minimize ederek (y_k,x_k), k=1,2,.....,m-1 şeklinde birleşmelere bağlı olarak eğitilebilir. Ağın snaptik ağırlıkları $G_{i,m}(\lambda)$ 'yı minimize ederek yeni tanımlı indis (y_m,x_m) ile güncelleştirilebilir. w_i'nin $G_{i,m-1}(\lambda)$'yi minimize edilerek (y_k,x_k), k=1,2,.....,m-1 çağrışımlarına bağlı olarak güncelleştirilmiş olduğu varsayılarak w_i

$=w_{i,m-1}$ şeklinde sonuçlanır. Snaptik ağırlıklar matrisinin (w'nın) i'nci sırası denklem 4.26'yı kullanarak, $G_{i,m-1}(\lambda)$'yi minimize ederek (y_m, x_m) çağrışımıyla güncelleştirilir.

$$w_{i,m} = w_{i,m-1} - \alpha H_{i,m}(\lambda)^{-1} \Big| w_i = w_{i,m-1} \frac{\partial G_{i,m}(\lambda)}{\partial w_i} \Big| w_i = w_{i,m-1} \tag{4.26}$$

burada $\partial G_{i,m}(\lambda)/\partial w_i$ $G_{i,m}(\lambda)$'nın wi'ye bağlı gradyeni, Hi,m(λ) Hessian matrisi, α öğrenme oranı, pozitif bir gerçek sayıdır.

Çizelge 4.2 Hızlı Delta Kuralı'nın akış diyagramı

Start
Initialize w with random values
Select μ
    λ=1
1   k=0 (çağrışım sayısı)
        E=0
    2   k← k+1
        x=x_k
        y=y_k

$$\hat{y}_i = \sigma\left(\sum_{j=1}^{n_i} w_{ij} x_j\right)$$

$$e_i = y_i - \hat{y}_i$$

$$\varepsilon_i^0(\lambda) = \lambda e_i + (1-\lambda)\tanh[\beta e_i] \qquad \text{(analog çıkış)}$$

$$\varepsilon_i^0(\lambda) = (1-\hat{y}_i^2)(y_i - \lambda \hat{y}_i) \qquad \text{(binary çıkış)}$$

$$w_i \leftarrow w_i + \alpha \varepsilon_i^0(\lambda) x$$

$$\hat{y}_i = \sigma\left(\sum_{j=1} w_{ij} x_j\right)$$

$$E \leftarrow E + \frac{1}{2}\sum_{i=1}^{n_0} (y_i - \hat{y}_i)^2$$

        if: k< m; then: go to 2
    λ= exp(-μ/E^2)
    if: E> E_o; then: go to 1
Stop

4.7.2 Hızlı Geriye Yayılım Algoritması

Bu algoritma k=1,2,....,m için 4.20'de belirlenen $G_k(\lambda)$ nesnel fonksiyonunu sıralı olarak minimize etmesi Ek2 de ayrıntılı bir şekilde gösterilir. w_{pq} snaptik ağırlıkları için güncel denklem aşağıdaki gibi elde edilir.

$$w_{p,k} = w_{p,k-1} + \alpha \varepsilon^{0}_{p,k}(\lambda) h_k \tag{4.27}$$

ağın çıkışı analog ise,

$$\varepsilon^{0}_{p,k}(\lambda) = \lambda (y_{p,k} - y_{p,k}) + (1-\lambda) \tanh\left[\beta (y_{p,k} - y_{p,k}) \right] \tag{4.28}$$

diğer taraftan ağ binary çıkışlara sahip ise,

$$\varepsilon^{0}_{p,k}(\lambda) = (1 - \hat{y}^{2}_{p,k})(y_{p,k} - \lambda \hat{y}_{p,k}) \tag{4.29}$$

v_{pq} snaptik ağırlıklarının aşağıda verilen denklem aracılığıyla güncelleştirilebildiği Ek2 de ayrıntılı olarak verilmektedir.

$$v_{p,k} = v_{p,k-1} + \alpha \varepsilon^{h}_{p,k}(\lambda) x_k \tag{4.30}$$

$$\varepsilon^{h}_{p,k}(\lambda) = (1 - \hat{h}^{2}_{p,k}) \sum_{i=1}^{n_0} \varepsilon^{0}_{i,k}(\lambda) w_{ip} \tag{4.31}$$

Çıkış hatası $\varepsilon^{0}_{i,k}(\lambda)$ağın çıkışı analogsa denklem 4.28'de verildiği gibi, binary elemanlarla oluşturuluyorsa 4.29'daki gibidir. Hızlı Geriye Yayılım Algoritması çizelge 4.3'te verilen akış diyagramıyla özetlenebilir.

Basitliğinden dolayı, bu algoritma eğitme süresince λ'nın etkinliğini araştırmak için ideal bir temel sağlar. Şimdi analog çıkışlı bir ağ düşünelim. Giriş adaptasyon çevrimleri boyunca λ≈1'dir. Bu durumda, denklem 4.28'in birinci terimi etkin haldedir. $e_{p,k} = y_{p,k} - \hat{y}_{p,k}$ eğitme süresince azaldığından λ'da azalır ve 4.28'in ikinci terimi etkin hale gelir. Böylece, tanh($\beta e_{p,k}$) lineersizliğinin hareketi hem β'ya hem de $e_{p,k} = y_{p,k} - y_{p,k}$'ya bağlı olur. Eğer, ($\beta e_{p,k}$) yeterince küçükse tanh($\beta e_{p,k}$)≈($\beta e_{p,k}$) eşit olur ve bu yüzden $\varepsilon^{0}_{p,k}(\lambda) \approx \left[\lambda + (1-\lambda)\beta\right] e_{p,k}$ olduğu görülür. β=1 olduğu özel durumda ise, $\varepsilon^{0}_{p,k}(\lambda) \approx e_{p,k} = y_{p,k} - \hat{y}_{p,k}$ olur. $\beta e_{p,k}$ yeterince büyükse tanh($\beta e_{p,k}$) +1 veya -1'e yaklaşır, bu da de $e_{p,k} = y_{p,k} - \hat{y}_{p,k}$'nın durumuna bağlıdır. Sonuç algoritmasının yaklaşımı tanh($\beta e_{p,k}$) lineersizliğinin hareketini belirleyen β değerinden etkilendiği kolayca anlaşılır. β ≈1 ise, tanh($\beta e_{p,k}$) lineer bir fonksiyon gibi hareket eder ve sonuç algoritmasının bu durumdaki yaklaşımının Hatanın Geriye Yayılımı

Algoritmasına benzediği kabul edilir. β'nın değeri bileşkeden yeterince büyükse algoritmanın yaklaşımı sonucu geliştirdiği kabul edilir. β sonsuza yaaklaştığından $\tanh(\beta e_{p,k})$ asimtotik olarak $\mathrm{sgn}(e_{p,k})$'ya yaklaşır. Bu şart altında, sonuçlandırma algoritmasının yaklaşım yapacağı garanti değildir yani uzaklaşabilir.

λ'nın etkinliğinin araştırılması durumunda, ağ binary çıkışlı ise bize sonuç açısından daha fazla bilgi veren bir duruma sahiptir. λ=1 olduğunda denklem 4.29, aşağıdaki denklemi verir.

$$\varepsilon^{0}_{p,k}(1)=(1-\hat{y}^{2}_{p,k})(y_{p,k}-\hat{y}_{p,k}) \tag{4.32}$$

Açık bir şekilde, λ=1 ise bu algoritma Hatanın Geriye Yayılımı Algoritmasıyla çakışır (aynı duruma gelir). Bu şart altında, w_{pq} snaptik ağırlıklarının adaptasyonu hata $e_{p,k}=y_{p,k}-\hat{y}_{p,k}$'la belirlenir. Bu durum rahatlıkla denklem 4.33'te görülmektedir.

$$\varepsilon^{0}_{p,k}(1)=\begin{cases}(1+\hat{y}_{p,k})(y_{p,k}-\hat{y}_{p,k})^{2} & if\ y_{p,k}=+1\\ -(1-\hat{y}_{p,k})(y_{p,k}-\hat{y}_{p,k})^{2} & if\ y_{p,k}=-1\end{cases} \tag{4.33}$$

$\hat{y}_{p,k}$ tahmini hedef $y_{p,k}$ 'ya yaklaştığında, snaptik ağırlıklar $e^{2}_{p,k}=(y_{p,k}-\hat{y}_{p,k})^{2}$ terimiyle etkinleşir. Bu terim, algoritmanın yaklaşımını giriş adaptasyon saykıllarından sonra belirgin bir şekilde yavaşlatır. λ=0 ise, denklem 4.34'teki duruma ulaşılır.

$$\varepsilon^{0}_{p,k}(0)=y_{p,k}(1-\hat{y}^{2}_{p,k})=\begin{cases}y_{p,k}(1+\hat{y}_{p,k})(y_{p,k}-y_{p,k}) & if\ y_{p,k}=+1\\ -y_{p,k}(1-\hat{y}_{p,k})(y_{p,k}-y_{p,k}) & if\ y_{p,k}=-1\end{cases} \tag{4.34}$$

Bundan sonraki denklem, λ'nın değerine bakmaksızın, snaptik ağırlıkların adaptasyonu $e_{p,k}=y_{p,k}-y_{p,k}$ hata terimiyle belirlenir. Ancak, snaptik ağırlıkların adaptasyonunda bu hata teriminin etkisi ağın eğitim süreci boyunca değişen bir eğim teriminden etkilenir.

Yukarıdaki analiz, klasik ikinci dereceden hata terimi $E=\frac{1}{2}\sum_{k=1}^{m}\sum_{i=1}^{n_0}e^{2}_{i,k}=\frac{1}{2}\sum_{k=1}^{m}\sum_{i=1}^{n_0}(y_{i,k}-\hat{y}_{i,k})^{2}$ 'in temeline dayanır. Bir gizli katmanlı, ileri-besleme sinir ağının eğitimi rölatif entropi kriterine (Karayiannis ve Venetsanopoulos,1992) dayanıyorsa, ağın snaptik ağırlıkları çıkış hatasını geriye yayarak güncelleştirilir.

$$\eta^{0}_{p.k} = y_{p.k} - y_{p.k} \qquad (4.35)$$

Giriş adaptasyon saykılları boyunca, her bir.....hedef $y_{p.k}$ 'dan çok uzaktır. Bu durumda, çıkış lineersizliği $\sigma(x) = \tanh(x)$ istenilen seviyeye yaklaşmaz. Bir sonuç olarak, çıkış lineersizliği lineer bir fonksiyon gibi hareket eder, bu da $\sigma(x) = \tanh(x) \approx x$ demektir. Bu tahmin altında, ağın binary çıkışı analog çıkış $y_{i.k} \approx y_{i.k}$ 'ya çok yakındır. Giriş adaptasyon saykılları boyunca, rölatif entropi kriteriyle sağlanan güncel denklemler $E = G(1) = \frac{1}{2}\sum_{k=1}^{m}\sum_{i=1}^{n_0}(y_{i.k} - y_{i.k})^2$ 'nin $\hat{y}_{i.k} = \bar{y}_{i.k}$ olduğu yerde minimizasyonuyla elde edilmeğe benzerlik teşkil eder. Diğer bir ifadeyle, giriş adaptasyon saykılları boyunca, rölatif entropi kriterinin maksimizasyonu, nesnel fonksiyon E'yi minimize ederek kendisinin analog çıkışındaki gibi, binary çıkışla bir ağı eğitmeye hemen hemen eşdeğerdir. Bu sonuçlar altında, binary çıkışlı ağların eğitiminin kendilerinin güncelleştirme kabiliyetini önemli derecede azalttığı deneysel olarak ispatlanmıştır (Karayiannis,1991). Rölatif entropi kriterine dayanan algoritmayla eğitilen sinir ağlarının, bu şekilde eğitilen klasik Hatanın Geriye Yayılımı Algoritmasından daha düşük genelleme kabiliyetine sahip olduğu şaşırtıcı birşey değildir (Solla vd., 1988).

Bu bölümde sunulan Hızlı Geriye Yayılım Algoritmasını ve giriş adaptasyon saykıllarından sonra yani, herbir $\hat{y}_{i.k}$ hedef $y_{p.k}$ 'ya yaklaştığında rölatif entropinin maksimizasyonuna dayanan algortimayı göz önüne alalım. $y_{p.k}=1$ ve $\hat{y}_{i.k} \in [0,1)$ ise, $1 \le y_{p.k}(1+\hat{y}_{i.k}) < 2$ 'dir. Denklem 4.6'da $\varepsilon^{0}_{p.k}(0)$ 'nın belirlenmesi $\eta^{0}_{p.k} \le \varepsilon^{0}_{p.k}(0) < 2\eta^{0}_{p.k}$ vurgular. $y_{p.k}=-1$ ve $\hat{y}_{i.k} \in (-1,0]$ ise, benzer şekilde $\eta^{0}_{p.k} \le \varepsilon^{0}_{p.k}(0) < 2\eta^{0}_{p.k}$ gösterilebilir. Bu ifade Hızlı Geriye Yayılım Algoritması ve giriş adaptasyon saykıllarından sonra rölatif entropi kriterine dayanan algoritma arasında bir benzerlik gösterir. Bu her iki algoritmanın hızlı yaklaşımını açıklar. Bu iki algoritma arasındaki çok önemli fark giriş adaptasyon saykılı boyunca Hızlı Geriye Yayılım ve Hatanın Geriye Yayılım algoritmalarına hemen hemen tanımlı genelleme kabiliyeti sağlamayı garanti eden $E = \frac{1}{2}\sum_{k=1}^{m}\sum_{i=1}^{n_0} e_{i.k}^2 = \frac{1}{2}\sum_{k=1}^{m}\sum_{i=1}^{n_0}(y_{i.k} - \hat{y}_{i.k})^2$ ikinci dereceden fonksiyonunun minimizasyonu ilkesine dayanır.

Yukarıdaki açıklama rölatif entropinin maksimizasyonuyla sağlanan güncel denklemler ve $\lambda=0$'a yaklaştığındaki giriş adaptasyon saykıllarından sonra Hızlı Geriye Yayılım algoritması arasında önemli bir benzerlik göstermiştir. Bu rölatif entropi kriteri denklem 4.19'da

genelleştirilmiş nesnel fonksiyonun formasyonuna dayandırılabilir. Gerçekte, bir ileri-besleme ağı nesnel fonksiyon $G(\lambda)=\lambda E+(1-\lambda)E'$, $E=\frac{1}{2}\sum_{k=1}^{m}\sum_{i=1}^{n_0}e_{i,k}^2=\frac{1}{2}\sum_{k=1}^{m}\sum_{i=1}^{n_0}(y_{i,k}-y_{i,k})^2$ 'deki ikinci dereceden hata fonksiyonu ve E'=-H olduğu durumda minimize ederek eğitilebilir. Bu durum, denklem 4.28'de snaptik ağırlıklar w_{pq}'nun durumunda rahatlıkla görülebilir.

$$\varepsilon_{p,k}^{0}(\lambda)=\left[\lambda(1-\hat{y}_{p,k}^{2})+(1-\lambda)\right](y_{p,k}-\hat{y}_{p,k})=(1-\lambda\hat{y}_{p,k}^{2})(y_{p,k}-y_{p,k}) \tag{4.36}$$

Benzer şekilde, snaptik ağırlıklar v_{pq} denklem 4.30'la güncelleştirilebilir. Buradaki $\varepsilon_{p,k}^{h}(\lambda)$ 4.18'de verilen $\varepsilon_{p,k}^{0}(\lambda)$ hata çıkışıyla, 4.31'den elde edilebilir.

Biraz önceki analizin direkt genellemesi, birden fazla gizli katmana sahip çok katmanlı sinir ağları için Hızlı Geriye Yayılım algoritmasına izin verir. $v_{pq}^{(r)}$; r=0,1,....,L snaptik ağırlıkları için güncel denklemler Ek2 de denklem 4.37 olarak elde edilir.

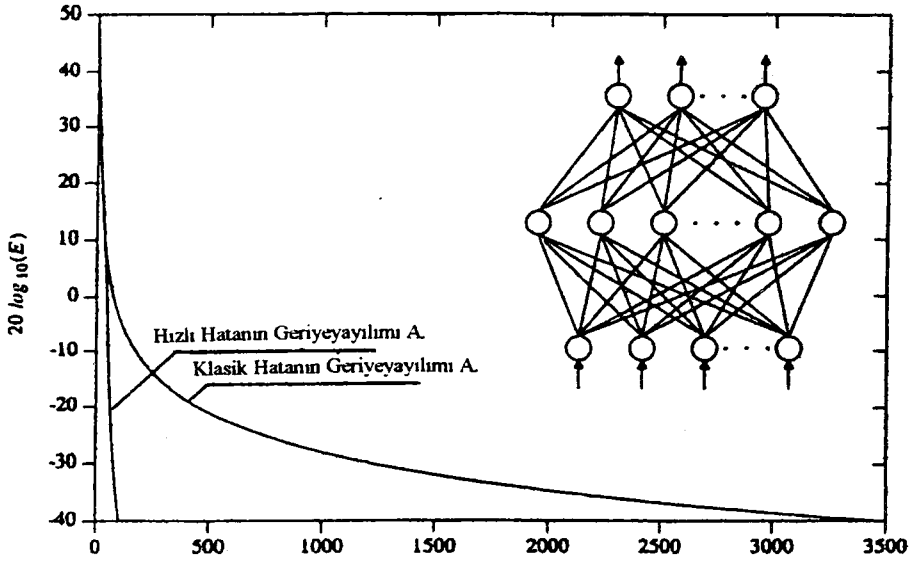
$$v_{p,k}^{(r)}=v_{p,k-1}^{(r)}+\alpha\varepsilon_{p,k}^{(r)}(\lambda)\hat{h}_{k}^{(r+1)} \quad \forall r=0,1,....,L \tag{4.37}$$

burada $\hat{h}_{k}^{(L+1)}=x_k$, çıkış hatası $\varepsilon_{p,k}^{0}(\lambda)=\varepsilon_{p,k}^{0}(\lambda)$ ağ çıkışı analog ise 4.28'e göre, ağ çıkışı binary ise 4.29'a göredir ve,

$$\varepsilon_{p,k}^{(r)}(\lambda)=(1-\hat{h}_{p,k}^{(r)2})\sum_{l=1}^{n_{r-1}}\varepsilon_{l,k}^{(r-1)}(\lambda)v_{lp}^{(r-1)} \quad \forall r=1,2,.....,L \tag{4.38}$$

şeklindedir.

Sonuca yaklaşma açısından Hatanın Geriye Yayılımı algoritmasının Hızlı Geriye Yayılım algoritmasıyla karşılaştırmalı diyagramı şekil 4.14'te gösterilmiştir.



Şekil 4.14 Hatanın Geriye Yayılımı algoritmasının Hızlı Geriye Yayılım algoritmasıyla karşılaştırmalı diyagramı

Çizelge 4.3 Hızlı Geriye Yayılımın akış diyagramı

Start

Initialize W ve V with random values

Select μ

λ=1

1 k=0 (çağrışım sayısı)

E=0

2 k← k+1

x=x_k

y=y_k

$$y_i = \sigma(\sum_{j=1}^{n_h} w_{ij} h_j)$$

$$e_i = y_i - y_i$$

$$\varepsilon_i^0(\lambda) = \lambda e_i + (1-\lambda)\tanh[\beta e_i] \qquad \text{(analog çıkış)}$$

$$\varepsilon_i^0(\lambda) = (1-y_i^2)(y_i - \lambda y_i) \qquad \text{(binary çıkış)}$$

$$w_i \leftarrow w_i + \alpha \varepsilon_i^0(\lambda) h$$

$$\varepsilon_j^h(\lambda) = (1-h_j^2)\sum_{i=1}^{n_0} \varepsilon_i^0(\lambda) w_{ij}$$

$$v_j \leftarrow v_j + \alpha \varepsilon_j^h(\lambda) w_{ij}$$

$$h_j = \rho(\sum_{l=1}^{n_i} v_{jl} x_l)$$

$$y_i = \sigma(\sum_{j=1}^{n_h} w_{ij} h_j)$$

$$E \leftarrow E + \frac{1}{2}\sum_{i=1}^{n_0}(y_i - y_i)^2$$

if: k< m; then: go to 2

$\lambda = \exp(-\mu/E^2)$

if: E> E_0; then: go to 1

Stop

4.7.3 Öğrenme ve Momentum katsayıları

YSA ile ilgili bir başka sorunda, düzgün bir öğrenme katsayısının (α) ayarlanmasıdır. Ağırlıkları çok yüksek tutmak davranışın bozulmasına neden olabilir. O nedenle öğrenme katsayısını böyle bir davranışı önlemek için küçük tutmak gereklidir. Öğrenme katsayısı, $0.01 < \alpha < 10$ aralığında seçilen sabit bir sayıdır. Öte yandan çok küçük bir öğrenme oranıda, öğrenme işleminin yavaşlamasına yol açar.

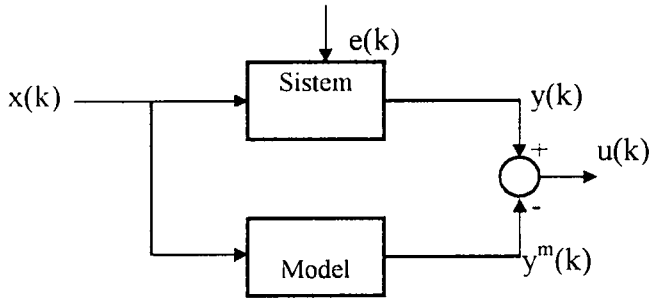
Momentum (ε) fikirsel olarak bu noktadan hareketle ortaya atılmıştır. Momentum mevcut delta ağırlığı üzerinden önceki delta ağırlığının belli bir kısmını besler. Böylece daha düşük öğrenme katsayısı ile daha hızlı öğrenme elde edilir. Momentum katsayısı genellikle $0 < \varepsilon < 1$ aralığında değişen sabit bir sayıdır.

4.8 Yapay Sinir Ağları ile Sistem Tanıma

4.8.1 Sistem tanıma

Sistem tanıma, sistem teorideki temel konulardan biridir ve bir sistemin deneysel olarak elde edilen giriş-çıkış verilerinden yararlanarak, sistemin matematiksel modelini oluşturmaya çalışır.

Şekil 4.15'te temel bir sistem tanıma blok diyaramı gösterilmektedir. (Narendra ve Parthasarathy, 1990).



Şekil 4.15 Sistem tanıma

4.8.2 Sistem tanıma aşamaları

Sistem tanıma aşağıdaki 4 aşamadan oluşur (Söderström ve Stoica,1989), (Landau, 1990).

1. Sistemin giriş-çıkış bilgilerinin elde edilmesi.
2. Model yapısının seçimi,
3. Model parametrelerinin kestirimi,
4. Modelin yapısının ve parametrelerinin uygunluğunun testi.

Tanınacak sisteme uygulanacak işaretler, sistem tanımada çok büyük öneme sahiptir. Girişler, sistemin tüm modlarının uyarabilmelidir. Bu yüzden girişler, frekans yönünden çok zengin olmalıdır. Tüm frekansları içerdiği için rastgele işaretler kullanılır. Lineer sistemler için girişin genlik yönünden iki seviyeli olması yeterli olabilirken, lineer olmayan sistemlerde, girişin genliği de rastgele olarak seçilmelidir.

4.8.3 YSA ile Off-Line sistem tanıma

Off-line sistem tanımada, önce sisteme rastgele girişler uygulanır ve sistemin bu girişlere ilişkin çıkışları elde edilir. Bu girişler ve çıkışlardan yararlanarak eğitim kümesi belirlenir (x:giriş, y:çıkış olmak üzere). Bu eğitim kümesi kullanılarak, ağ, kullanılan algoritma ile grup uyarmalı ve veri uyarmalı olarak eğitilir.

4.8.4 YSA ile ON-Line sistem tanıma

On-line sistem tanımada, sabit bir eğitim kümesi yoktur. Sisteme, rastgele girişler uygulanır ve sistemin bu girişlere ilişkin çıkışları elde edilir. Ağ giriş çiftine göre veri uyarmalı olarak eğitilir. Momentum metodu kullanılabilmesine rağmen adaptif öğrenme oranı kullanılamaz. On-line sistem tanımada, örnekler sürekli olarak değiştiği ve bu hataya göre eğitimin sonlandırılması gerekir Bu noktada her çıkış için bağıl hata şu şekilde tanımlanır (Parlos vd., 1994),

Off-line sistem tanımada öğrenme zamanı, fazla önem taşımamaktadır. On-line sistem tanıma, sistemin sürekli çalıştırılmasıyla yapıldığı için performansının yüksek olması istenir. Yapılan çalışmada vektör kontrol, DSP ile işleme ve YSA kullanılarak performans artışına dikkat edilmiştir.

5. DSP'NİN GENEL YAPISI, DİĞER İŞLEMCİLERLE KARŞILAŞTIRMALI ANALİZİ VE UYGULAMA ALANLARI

5.1 DSP'nin Genel Özellikleri

Kontrol algoritmalarının gerçek zamanda işlem yapabilmesi, dış dünyadan; yani üzerinde çalışılan sistemden elde edilen fiziksel büyüklüklerin değerlendirilip aynı anda gerekli olan kontrol parametrelerinin hesaplanması anlamına gelmektedir. Sistemler karmaşık bir yapıya sahip olup, algoritmalar daha kapsamlı hale geldikçe mikroişlemciler yapıları itibariyle çözüm için yeterli olmamaya başlamışlardır. Tüketiciden gelen talepler doğrultusunda üretici firmalar uygulamalara özel bazı işlemciler üretmişlerdir. Bu işlemciler genel amaçlı işlemcilerden daha hızlı, fakat uygulamalara özel olduklarından kullanım alanı açısından genel anlamda sınırlı işlem kapasitesinde kalmışlardır. Bu tür teknolojiye yönelik uygulamalarda ulaşılmak istenen hedef, mikroişlemcilerin yararlarını yapısında bulunduran ve yetersiz kaldığı durumlarda yeni olanaklar sunan bir işlemci geliştirmek olmuştur.

Mikroişlemcilerin üretilmesinde kullanılan yarı iletken teknolojisi, gün geçtikçe yeni gelişmelere tanık olmaktadır. VLSI (Very Large Scale Integration) denilen üretim tekniği yardımıyla, çok daha hızlı ve güçlü yapılar, çok daha küçük alanlara sıkıştırılarak, üretilmeye başlanmıştır. İşlemcilerin güç kaybının düşük değerde olmasını sağlayan bu sistem, tasarımcıya karmaşık problemlerin kolayca çözümlenebilmesi kolaylığını getirmiştir. Mikroişlemcilerin mimari yapıları iki çeşittir. Bu mimariler kendilerini geliştiren kişilerin adları ile anılmaktadır. Bunlardan bir tanesi Von Nuemann olarak adlandırılmıştır. Veri ve komutların saklanması için aynı bellek biriminin kullanılması, aynı bilgi transfer yolu üzerinde veri daralması olarak da tanımlayabileceğimiz veri sıkışması problemlerine neden olmaktadır. Komut ile veri aynı yol üzerinden taşınmaktadır. Bu da işlemlerin çok daha uzun sürede tamamlanmasına ve saat çevrim süresinin uzamasına sebep olmaktadır. Örneğin 16 bitlik sayılarla işlem yaparken, 8 bitlik bir işlemci, sayıları iki parça olarak alıp, işlemektedir. Bu da 16 bitlik bir işlemcinin yarı hızında çalışması anlamına gelmektedir. Yeni nesil işlemcilerde, üniteler bağımsız olarak çalıştırılarak, hızı arttırma yoluna gidilmiş ve kısıtlı bir paralel çalışma sağlanmıştır. Veri taşınması, eklenen yeni komut ve yapı değişikliği ile tek saat çevriminde gerçekleştirilebilmektedir.

Harvard mimarisi olarak adlandırılan ikinci tip mimari DSP lerde ve yeni nesil işlemcilerde kullanılmaya başlanmıştır. Bu mimari tipinde veri ve adres yollarının birbirinden ayrı olması nedeniyle, işlemci çok daha hızlı işlem yapabilmektedir. Bunun yanısıra DSP'lerde bulunan paralel işlem yapabilme özelliği, aynı anda birkaç komut işleyerek hızının katlanmasını

sağlayan bir diğer etken olmaktadır. Bir mikroişlemcinin işlem yapabilme kapasitesini gösteren komut sayısı ve adresleme modları, DSP sistemlerinde diğer işlemcilere göre daha kısıtlı gibi görünse de, sistemin özelliklerini tam olarak kullanmayı sağlayan özel komutlar sayesinde, birçok işlem tek bir saat çevrimi süresinde yapılabilmektedir. Özellikle kontrol sistemlerinde çok kullanılan çarpma ve toplama işlemlerinin birkaç makine çevriminde gerçekleştirilebilmesi, hız yönünden büyük bir avantaj getirmektedir. Bir algoritma içinde tekrarlanan döngü işlemleri; DSP'lerin özel komutları sayesinde çok daha hızlı ve verimli olarak gerçekleştirilebilmektedir. Donanım olarak yapıların birbirinden bağımsız olması, ayrı yollar üzerinden veri ve adres iletebilmesi; merkezi işlem birimi, merkezi mantık birimi ve yardımcı kaydedici aritmetik ünitelerinin yoğun paralellik içersinde çalışması sistemin üstün özellikleri olarak tanımlanabilir.

Uygulamaya yönelik olarak, işlemlerde kullanılan sayıların yapısal özellikleri doğrudan doğruya yolların genişliğine bağlıdır. Genel olarak sabit nokta aritmetiği ile çalışan sistemlerde 16 bit, daha büyük sayılar kullanan kayan nokta aritmetiği ile çalışan sistemlerde 32 bit yol genişliği bulunmaktadır. Çarpma ve toplama işlemleri sonrasında oluşan yuvarlama ve kesme hataları, kontrol edilen sistem üzerinde büyük hatalara neden olabilir. Bunu önleyebilmek için, işlemcinin işlem yapabilme kapasitesinin yeterince büyük olması gerekmektedir. DSP'lerde donanım olarak bulunan çarpma ünitesinin çıkışı, çarpma işleminden sonra bit sayısının iki katı bir sonuç oluşturmaktadır. Veri yolunun iki katı büyüklüğünde olan bu sayı ile, belirli bir yuvarlatma hatası sonucu tek bir kelime boyutunda veya doğrudan çift kelime boyutunda işlem yapılabilir.

DSP'nin program çevrimleri sırasında hangi adresteki işlemi gerçekleştireceğini otomatik olarak belirlemesi, istenen bir özelliktir. Bu işlemin Aritmatik-lojik Ünite (ALU) tarafından gerçekleştirilmesi mümkündür. Fakat belirli bir gecikmeye neden olacağından, DSP içinde ayrı bir veri adres generatörü bulunmaktadır. Adresleme işleminin önem kazandığı, sıralı adreslemenin yapılmadığı uygulamalarda bu ünite, bağımsız çalışan bir işlemci gibi adresleri üretmektedir.

Algoritmanın işlemesi sırasında komutların sıralı ve doğru olarak işlemesi, komut sıralama ünitesi sayesinde olmaktadır. Bu ünite sayesinde, komutların işlenmesi sırasında, işlemcinin mikrokod olarak çevrim ve döngüleri kullanmasına gerek kalmamaktadır. Bu da komutların daha az çevrim ile daha kısa sürede tamamlanmasını sağlamaktadır.

Ünitelerde yapılan işlemler sonunda elde edilen verilerin genişliğinin, veri yolunun genişliğinden fazla olmasından dolayı oluşan hatalar, kaydırma ünitesi yardımı ile ana işlemciye gerek kalmadan, tek bir çevrim süresinde çözümlenmektedir. Kaydırma ünitesi aynı

zamanda, çok bit'li tek çevrim kaydırmalar, normalize/denormalize işlemleri, kayan nokta aritmetiği gibi işlemlerde büyük kolaylık sağlamaktadır.

Yukarıdaki kısımlarda sözü edilen, sayısal işaret işleme sırasında karşılaşılan sorunları çözmek için, DSP'ler çeşitli yöntemlerle -bu bir yazılım olabilir- en az hata ile işlem yapabilmelidir. Komut işlemesi sırasında komutların aynı anda aynı veri yollarını kullanma isteğinden ortaya çıkan durum veri sıkışması olarak adlandırılmaktadır. Aynı durumla işlemcilerin veri işlemesinde de karşılaşılmaktadır. Bu sorunu ortadan kaldırmak için işlemcinin paralele çalışma denilen parçalara ayırarak çalışma yapısının kurulması gerekmektedir. Bu sayede komut çevrim süresi en az düzeyde olacak ve aynı anda birden fazla komut işlenebilecektir. İşlemi paralel yollara ayırarak yapılan çalışma, her bir parçanın bağımsız olarak işlenmesi prensibine dayanmaktadır. Bu çalışma şekli, parça sayısı ile ters orantılı olacak oranda, işlemin kısa sürede tamamlanmasını sağlamaktadır. Bunu yanında, işlemlerin optimal olarak dağıtılması her zaman söz konusu olmayabilir. Fazladan eklenen kütükler işlemin hızını arttırmaya yarayacaktır. Örneğin çarpma işleminde, çarpanların ayrı yollar üzerinden taşınması, çarpma işleminin süresini iki katı oranda azaltmaktadır.

Yukarıdaki kısımda bahsettiğimiz paralel çalışma özetle, aynı anda birden fazla komutun işlenmesine izin veren yapıya sahip olmasının yanı sıra, komut saat çevrimini en düşük seviyesine indirirken, işlemcinin çıkışından en yüksek performansın alınmasını sağlamaktadır. Bu da performansı arttırıcı bir diğer etkendir. Birinci nesil DSP'lerde (TMS1x) bir paralel çalışma hattı varken; yani bir fazladan komutun aynı anda işlenmesine izin verirken, ikinci nesil işlemcide (TMS2x) üç seviyeli, üçüncü nesil işlemcide (TMS3x) dört seviyeli ve beşinci nesil işlemcide altı seviyeli çalışma söz konusudur.

Paralel çalışma işleminin tam olarak yeterli olmadığı durumlarda paralel çalışma alternatif bir çözüm olmaktadır. Bir işlemin alt parçalara ayrılıp işlenmesi yerine, paralel işlemciler ile aynı işlemin yapılması çok daha iyi sonuç vermektedir. Bu tür çalışma ise, donanımın karmaşıklığına ve fiyatının biraz artmasına sebep olsa da yapılacak işlemlerin birbirinin yerine geçebilir yapıda ayarlanması çalışmadaki verimi arttıracaktır.

Bütün bu anlatılanlar dikkate alındığında bu ünitelerin tümünün birbiri ile uyumlu çalışabilme özelliklerinden dolayı, karmaşık algoritmalar hızlı bir şekilde işlenebilmektedir. Mimari yapıyı oluşturan ünitelerin birbirinden bağımsız olarak çalışabilmesi, sistemin hızını katlamaktadır. Komut çevrim süresinin kısa olması, belirtilen bu özelliklerden çok daha verimli olarak yararlanılabilmeyi sağlamaktadır. DSP'ler de arzu edilen özellik, tüm komutların tek bir saat çevriminde işletilmesidir. Bunun mümkün olmaması durumunda, gelişmiş mikroişlemcilerdeki gibi, komutların mikrokod olarak çözümlenmesi yerine, mimari yapı, komutların gerektirdiği işlemleri doğrudan yapabilen lojik bir yapı olarak kurulmuştur.

Sistem, mümkün olduğu kadar çok işlemi mümkün olan en az saat çevriminde gerçekleştirdiğinde, en verimli mimari yapıya sahip demektir.

Sistem tasarımı sırasında, DSP sisteminin seçiminde belirli faktörlerin önemi bulunmaktadır. Bunlar; mimarisini oluşturan ünitelerin iç yapısı, işlem yapabilme kapasiteleri, komut giriş grupları, ve geliştirme üniteleridir. Genel amaçlı işlemcilerde benzer yapılar bulunmasına rağmen, DSP'lerin kendilerine özgü bir sınıflandırması bulunmaktadır. Sistem mimarisi, kontrol edilecek sisteme çok bağımlıdır. Gerekli olan yapı ancak kontrol edilecek sistemin çok iyi tanınması ile mümkündür.

Seçilecek mimari yapı, kontrol algoritmasını en az hata ile, en hızlı biçimde en yüksek performansı sağlayacak şekilde gerçekleştirmelidir. Bunun yanında, başka bir sisteme ne kadar kolay uyum sağladığı etkili bir faktördür. Kontrol sisteminin isteğine göre, yardımcı birimler, bellek birimleri ve giriş - çıkış birimlerini bünyesinde bulundurmalı, veya bu birimler ile kolaylıkla iletişim kuracak yapıda olmalıdır. DSP sistemlerinde kolay programlanabilme de önemli bir özelliktir. Bunların yanında performans-fiyat ve performans-verim arasındaki ilişkilerinde göz önüne alınması gereklidir.

5.2 Yapılan Çalışmada Kullanılan TMS320C50 Mimarisi

5.2.1 Bus (Bilgi iletim hat) yapısı

Ayrı program ve veri yolları, yoğun paralel çalışma sayesinde program komutlarının ve verinin aynı anda işlenmesini sağlamaktadır. Örneğin, gelen iki verinin çarpma işlemi yapılırken, diğer bir taraftan da, bir önceki çarpım akümülatördeki değerden çıkartılabilir, veya toplanabilir; ayrıca yeni bir adres üretilebilir. Bu paralel çalışma sayesinde tek bir makine çevriminde, aritmetik, lojik ve bit-değiştirme işlemlerinin yapılması sağlanır. C5x mimarisinde 4 ana bus bulunur

1- Program Bus (PB)
2- Program Adres Bus (PAB)
3- Veri (Data) Okuma Bus'ı (DB)
4- Veri (Data) Okuma Adres Bus'ı (DAB)

Burada, PAB okuma ve yazma işlemleri için program hafıza alanına adres sağlamakta; PB, program hafıza alanından Merkezi İşlem Birimi (CPU)'ya komut kodları ile ani operasyonel komutları taşırmakta; DB, veri hafıza alanı ile CPU'yu birleştirmekte kullanılmaktadır.

Program ve veri busları birlikte çalışarak tek bir makine çevrimi ile yapılan çarpma/aküde saklama işlemlerini gerçekleştirmektedirler. (Texas Instruments C5x User's Guide, (1997)).

5.2.2 Çip üzeri hafıza

C5x'in mimarisi sistem performansını arttırmak amacı ile değişik hafıza birimleri ile donatılmıştır.

1- Sadece okunabilir program hafıza (ROM)

2- Veri/program iki çeşit işlem yapabilen RAM (DARAM)

3- Veri/program tek işlem yapabilen RAM (SARAM)

C5x 16 bitlik 224 K-word'lük kelimeyi adresleyebilen bir işlemcidir. Hafıza alanı 4 adet tek tek seçilebilen hafıza parçalarına ayrılmıştır. Bunun parçaları 64 K-word (kelime) program hafıza alanı, 64 K-word'lük lokal veri hafıza alanı, 64 K-word'lük giriş/çıkış kapıları, ve 32 K-word'lük global veri hafıza alanıdır.

5.2.2.1 Sadece okunabilir program hafızası

Bütün C5x DSP'lerinin üzerinde 16 bitlik maskelenebilir ve programlanabilir ROM vardır. Bu DSP'ler üzerlerindeki ROM'lar da boot yükleyici kod taşırlar. Bu hafıza sayesinde program kodu, yavaş olan dış ROM veya EPROM'dan, daha hızlı olan iç veya dış RAM'a aktarılır. Eğer iç ROM seçili değil ise, yani MP/MC ucu yüksek sinyalde ise C5x yürüteceği işleme dış hafızadan başlar.

5.2.2.2 Veri/Program ikili işlem yapan RAM

Bütün C5x DSP'ler üzerlerinde 16 bitlik 1056 kelimeden oluşan ikili işlem yapan RAM (DARAM) bulundurmaktadır. Bu, DARAM 3 adet ayrı ayrı seçilebilen hafıza blokları içermektedir.

B0: 512 kelimelik veri veya program DARAM bloğu

B1: 512 kelimelik veri DARAM bloğu

B2: 32 kelimelik veri DARAM bloğu

DARAM bloklarının asıl amacı veri kümelerinin saklanması olduğu halde; gereken durumlarda program saklanması amacı ile de kullanılabilirler. Yukarıda da belirtildiği gibi, B0 bloğu veri ve program bilgilerinin saklanması için kullanılabilirken; B1 ve B2 blokları

sadece veri bilgilerinin saklanması amacı ile kullanılmaktadır. DARAM aynı zaman da C5x'in işlem yapma hızını da arttırmaktadır.

5.2.2.3 Veri/Program tek işlem yapan RAM

C52 dışındaki tüm C5x DSP'ler üzerlerinde 16 bitlik tek işlem yapabilen RAM da bulundurmaktadır. Herhangi bir kodun işlenmesi harici ROM'dan başlayıp, yapılacak işlem DSP nin tam hızda çalıştırılmasından sonra dahili SARAM'a gönderilir. Böylece SARAM yazılım ile aşağıdaki durumlardan biri ile biçimlendirilebilir.

1- Bütün SARAM veri hafıza olarak biçimlendirilebilir.

2- Bütün SARAM program hafıza olarak biçimlendirilebilir.

3- SARAM hem program hem de veri hafıza olarak biçimlendirilebilir.

SARAM adres hafıza alanında 1K ve/veya 2K bitişik kelime bloklarına ayrılmıştır. Bütün C5x DSP'ler bu hafıza alanları ile paralel çalışabilirler. Fakat bir SARAM bloğu her makine çevriminde sadece bir defa işlenebilir. Diğer bir değişle, CPU bir SARAM bloğu üzerinde çalışırken, diğer bir SARAM bloğunda da okuma veya yazma işlemi yapabilir. CPU çoklu işlem yapılmasına ihtiyaç duyduğunda; SARAM yapacağı işlemleri planlar, sıraya sokar ve bu işlemleri gerçekleştirirken CPUya da işlem sonucunun hazır olmadığını belirtmek için hazır-değil (not-ready) şeklinde bir sinyal yollar. Bu sinyal süresince işlemlerin her birini SARAM birer saat çevriminde gerçekleştirir.

SARAM, DARAM'a göre adres haritalandırılmasında daha esnek davranır. Çünkü SARAM aynı anda hem program hem de veri hafıza olarak, yapılmakta olan işlem için şartlandırılabilir. Fakat komutu bulup getirme ve veriyi bulup getirme işlemleri DARAM da tek bir makine çevriminde yapılabilirken; SARAM'da 2 makine çevrimi sürmektedir.

5.2.2.4 Dahili hafıza koruması

C5x DSP'ler hafızanın içeriğini koruyan maskelenebilir çip seçme özelliğine sahiptirler. İlgili bit aktif duruma getirildiği zaman dış dünya verilerine dayalı hiçbir komut, çip üzerindeki hafızalarda işlem yapamaz.

5.2.3 Diğer dahili çevre birimleri

Bütün C5x DSP'ler aynı CPU yapısına sahip olsalar da, farklı dahili çevre birimleri içerirler. C5x'in dahili diğer donanımları şunlardır.

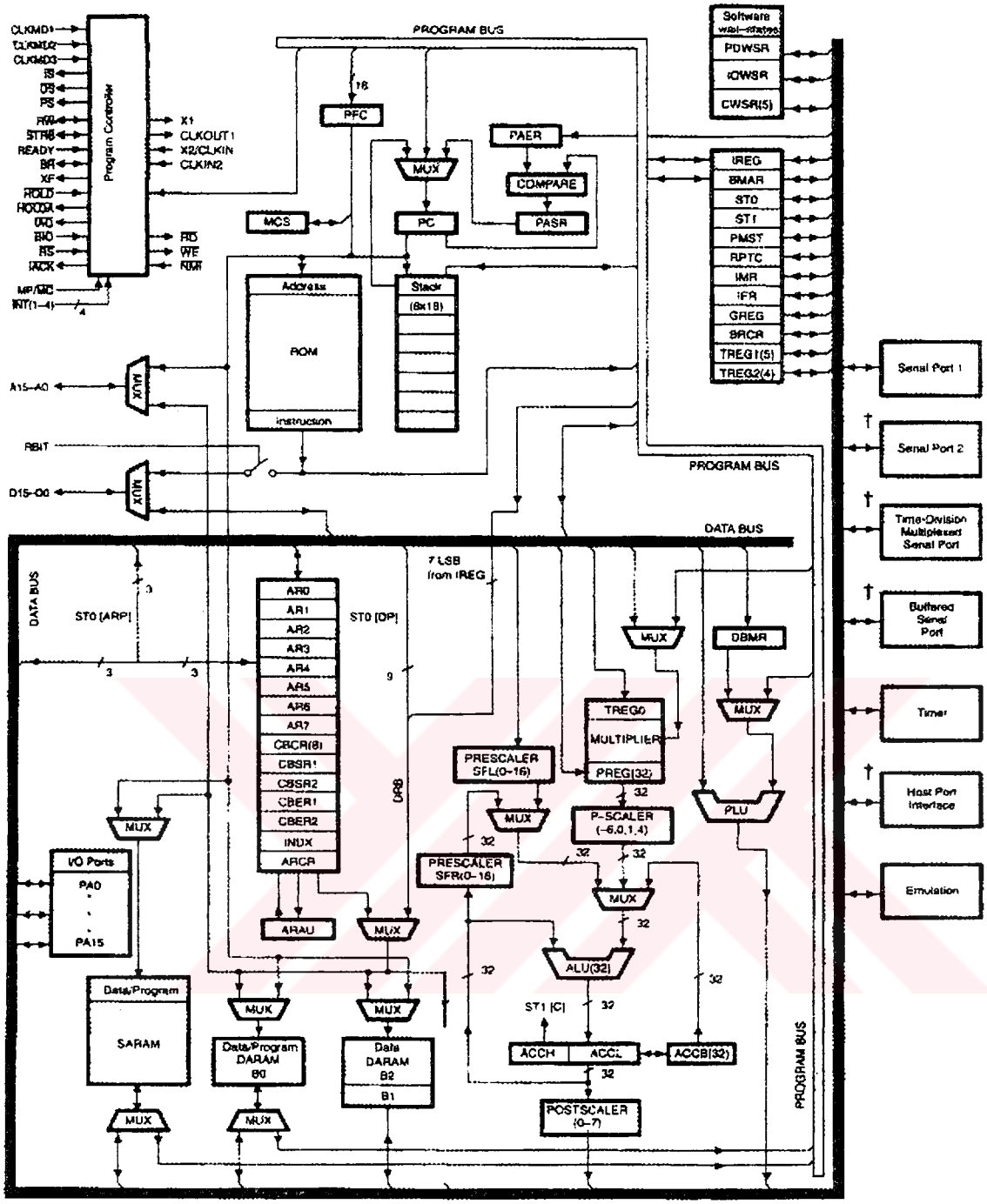
1- Saat darbe üretici
2- Donanımsal zamanlayıcı
3- Yazılımla programlanabilen durum beklemeli jeneratörler
4- Paralel giriş-çıkış kapıları
5- Yardımcı kapı arayüzeyi
6- Tampon bağlantı görevi yapan seri kapı
7- Seri kapı
8- Zaman paylaşımlı seri kapı
9- Kullanıcı tarafından maskelenebilir kesmeler

5.2.4 Merkezi işlem birimi (CPU)

Merkezi işlem biriminin içerisinde şu bloklar bulunur.
Merkezi Aritmetik Lojik Ünite (CALU)
Paralel Lojik Ünite (PLU)
Yardımcı Kütük veya Kaydedici Aritmetik Ünitesi (ARAU)
Hafızada Haritalanmış Kütükler
Program Kontrolörü
Merkezi İşlem Birimi'nin blok diyagramı şekil 5.1'de verildiği gibidir.

5.2.4.1 Merkezi aritmetik lojik ünite (CALU)

Merkezi aritmetik lojik ünitesinin alt bölümleri şunlardır.
16 bit x 16 bit paralel çarpıcı,
32 bit 2'ye tümlemeli aritmetik mantık ünitesi,
32 bitlik akümülatör,
32 bitlik akümülatör biçimlendirici
0-, 1-, veya 4-bit sola ve 6-bit sağa kaydırıcı
0—16 bit'e kadar sola kaydırıcı,
0—16 bit'e kadar sağa kaydırıcı,
0—7 bit'e kadar sola kaydırıcı,



Şekil 5.1 C5x DSP'sinin blok diyagramı – Merkezi İşlem Birimi

5.2.4.1.1 Çarpıcı, çarpım kütüğü (PREG) ve geçici kütük 0 (TREG0)

16 bit x 16 bitlik, dahili çarpıcı yardımı ile işaretli veya işaretsiz 32 bitlik çarpım, tek bir makine çevrimi ile elde edilir. İşaretsiz çarpım komutu (MPYU) (multiply unsigned) dışındaki tüm çarpma komutları işaretli çarpma yaparlar. Yani çarpılan sayılar 2'ye tümlemeli sayılardır ve sonuç karşımıza 32 bitlik 2'ye tümlemeli bir sayı olarak çıkar.

Çarpanlardan birisi hafızada adreslenmiş geçici kütüklerden veya kaydediciden TREG0'a girilmekte, diğer giriş ise, veri yolundan veya program yolundan sağlanmaktadır. Elde edilen 32 bitlik sonuç PREG adlı kütük de Aritmatik Lojik Birimi'nin (ALU) işlemleri için hazır olarak bekletilir. ALU, üzerine düşen işlemleri gerçekleştirirken gerekli bilgileri ya 16 bitlik veri hafızadan ya da ani komutlardan veya 32 bitlik PREG kütüğünden alır. ALU aynı zamanda Bool (2'li sistem) işlemlerini de gerçekleştirebilir. 32 bitlik ALU çıkışı Akümülatör (ACC)'de depolanır. ACC aynı zamanda ALU'ya ikinci bir veri girişi sağlar. ACC'deki 32 bitlik bilgi ACCH ve ACCL olarak 16'şar bitlik bilgiler halinde tutulur.

Kaydırıcılar (p-scaler, prescaler, post scaler) merkezi aritmetik lojik ünitesinin ölçekleme (nümerik ölçekleme, bit işlemleri, taşma koruması v.b.) yapmasına yardımcı olurlar. Bu kaydırıcılar PREG ve ACC çıkışlarına bağlanmışlardır.

4 adet kaydırma modu (PM) sayesinde PREG çıkışı çarpma/depolama işlemleri ile kesirli aritmetik,kesirli çarpma işlemlerini yapar. Bu 4 adet kaydırma durumu ST1 durum kütüğünün PM bitleri yardımı ile belirlenir.

1- PM=00_2 32 bitlik PREG çıkışı kaydırılmaz ve sonuç kaydırılmadan ALU'ya verilir.

2- PM=01_2 PREG çıkışı 1 bit sola kaydırılarak ALU'ya aktarılır. LSB=0 konulur. Bu mod yardımı ile 16 bitlik iki sayı çarpıldığında elde edilen 1 bitlik fazladan işaret bit'i kompanze edilmiş olur.

3- PM=10_2 PREG çıkışı 4 bit sola kaydırılarak ALU'ya aktarılır. 4 adet LSB = 0 konulur Bu kaydırma işlemi ile MPY çarpma komutunun kısa ani değerler ile (13 bit veya daha az) kullanıldığında ve diğer çarpanın 16 bit uzunluğunda olması durumunda elde edilen 4 bitlik fazlalık işaret bitlerinin yok edilmesinde kullanılır.

4- PM=11_2 PREG çıkışı 6 bit sağa kaydırılarak ALU'ya aktarılır. 6 LSB bit'i yok olur. Bu komut yardımı ile 128'e kadar arka arkaya çarpma/yükleme işlemi taşma problemi olmadan gerçekleştirilebilir.

Bu sırada PREG deki bilgi, kaydırma esnasında değişmez.

LT komutu yardımı ile TREG0'a veri yolundan gelen bilgi ile yüklenir; MPY komutu sayesinde çarpma işlemleri için gerekli ikinci çarpan sağlanmış olur. Kısa veya uzun ani değer operatörüyle çarpım yapıldığında MPY komutu bir ani değer operatörü ile kullanılır.

Mevcut 4 çeşit çarpma/yükleme komutu (MAC,MACD,MADD,MADS) yardımı ile her türlü çarpma işlemi gerçekleştirilebilir. Bu komutlar sayesinde her iki operatör üzerinde de aynı anda işlemler yapılabilir. Bu operatörler için gerekli olan veri her çevrimde veri ve program yollarından çarpıcılara sağlanır. Bu komutlar, tekrar komutları olan RPT ve RPTZ komutları ile birlikte kullanıldıklarında, bir tek çevrimlik çarpma/yükleme komutu olarak işlem görürler.

Bu tip tekrarlamalı komutlarda katsayı adresleri PC (Program Sayıcı) ile sağlanırken, veri adresleri ARAU (Yardımcı Kütük veya Kaydedici Aritmetik Ünitesi) tarafından sağlanır. RPTZ komutu yapacağı işlemlere başlamadan önce ACC'yi ve PREG'in içeriğini temizler. Örneğin, 10 x 10 boyutlu iki matristen, birinin satırı ile diğerinin sütununu çarpalım. MTRX1 birinci matrisin başlangıç adresini, INDX=10 matris boyutunu göstermek üzere, o andaki AR (Yardımcı Kaydedici) ikinci matrisin başlangıcını ifade etsin. Bu durumda program ifadesi aşağıdaki gibi olur. Şekil 5.2 CALU'nun blok diyagramını göstermektedir.

```
*****************************************************************
RPTZ  #9                ; for I=0 , I<10 , I++
MAC  MTRX1,*0+          ; Preg=DATA(MTRX1+I) x DATA[MTRX2+(I+INDX)]
APAC                    ; ACC ⇐ PREG
*****************************************************************
```

5.2.4.1.2 Aritmetik lojik ünite (ALU) ve akümülatörler (ACC)

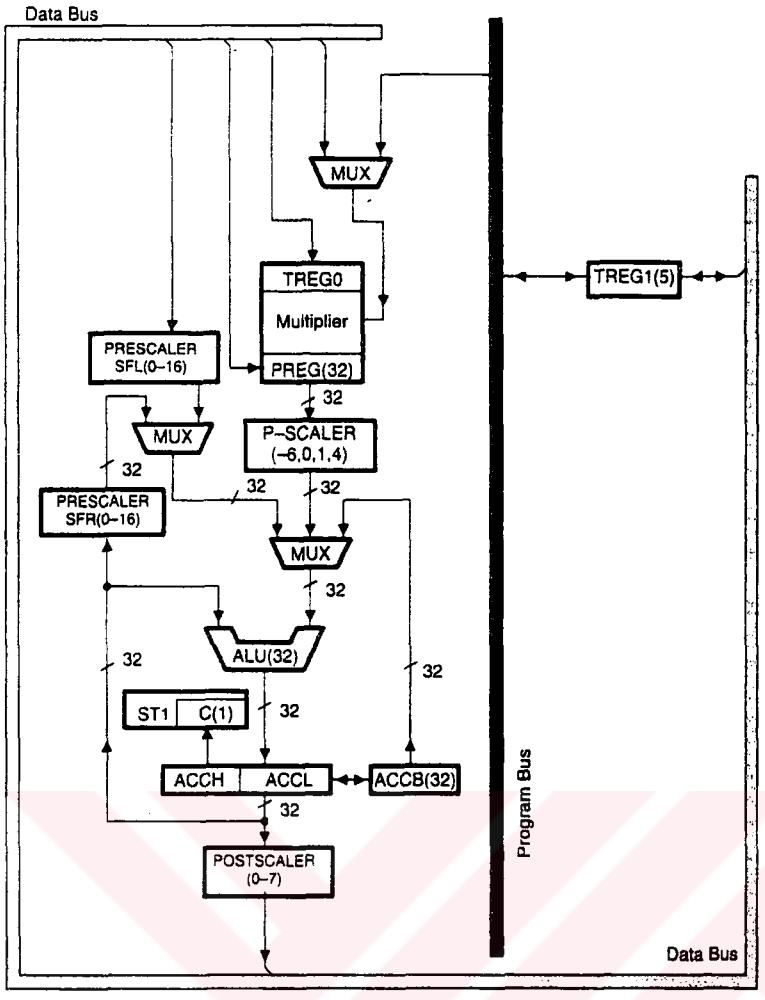
32 bitlik genel amaçlı ALU ve ACC bir çok aritmetik ve lojik işlemlerin gerçekleşmesine izin verir. Bu işlemlerin büyük bir kısmı tek bir saat çevriminde yapılmaktadır. İşlem gerçekleştirildiğinde, ortaya çıkan sonuç ACC'ye aktarılır. ALU'ya aktarılma işleminde ise veri ölçeklendirilebilir.

Tipik bir ALU işleminin gerçekleştirilmesi esnasında aşağıdaki adımlar sırasıyla takip edilmektedir.

1. Veri, hafızadan veri hattına (bus'ına) aktarılır,
2. Veri, kaydırıcı (prescaler)'dan ALU'ya geçirilir ve orada işlenir,
3. Sonuç ACC'ye aktarılır.

ALU; işlemlerini, veri hafızasından gelen veya ani komutlardan türetilen 16 bitlik kelimeler ile gerçekleştirmektedir. ALU aritmetik işlemlerin yanında lojik işlemlerini de yapar. ALU'nun bir girişi her zaman ACC'den sağlanmakta olup, diğer giriş ise, çarpıcının PREG kütüğünden, ACCB veya kaydırıcı (pre-scaler) ölçekleyicisinin çıkışından yapılır. ALU'nun içeriği bu işlemler tamamlandıktan sonra ACC de depolanır. Akümülatördeki bilgiler ile Akümülatör Tampon (Buffer) Geçiş'teki bilgiler arasında karşılaştırma işlemleri de bu birimde yapılabilir.

ACCB; ACC bilgilerinin geçici olarak saklandığı tampon geçiş kütüğüdür.



Şekil 5.2 Merkezi Aritmetik Lojik Birimi (CALU)'nun blok diyagramı

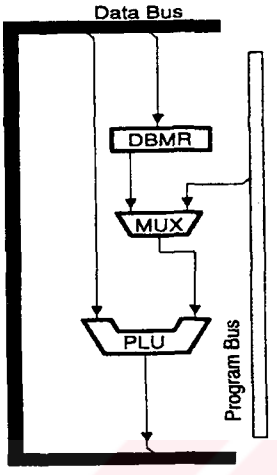
5.2.4.1.3 Ölçekleme kaydırıcıları ve geçici kütük 1 (TREG1)

Kaydırıcı (pre-scaler), veri bus'ına bağlı 16-bitlik bir girişe ve ALU'ya bağlı 32-bitlik bir çıkışa sahiptir. Bu kaydırıcı giriş verisinde 0'dan 16'ncı bite kadar bir sola kaydırma üretmektedir. Bu kaydırma sayısı komut kelimesinde yerleştirilmiş bir sabitle veya TREG1'deki bir değerle belirlenmektedir.

5.2.4.2 Paralel lojik ünitesi (PLU)

Paralel lojik ünitesi (PLU) yardımı ile kontrol ve durum kütüklerinin bitleri birer, birer, veya gruplar halinde değiştirilebilir ve kontrol edilebilir. ACC veya PREG'in içeriği değiştirilmeden, PLU yardımı ile veri hafıza değerleri üzerinde doğrudan işlemler yapılabilir.

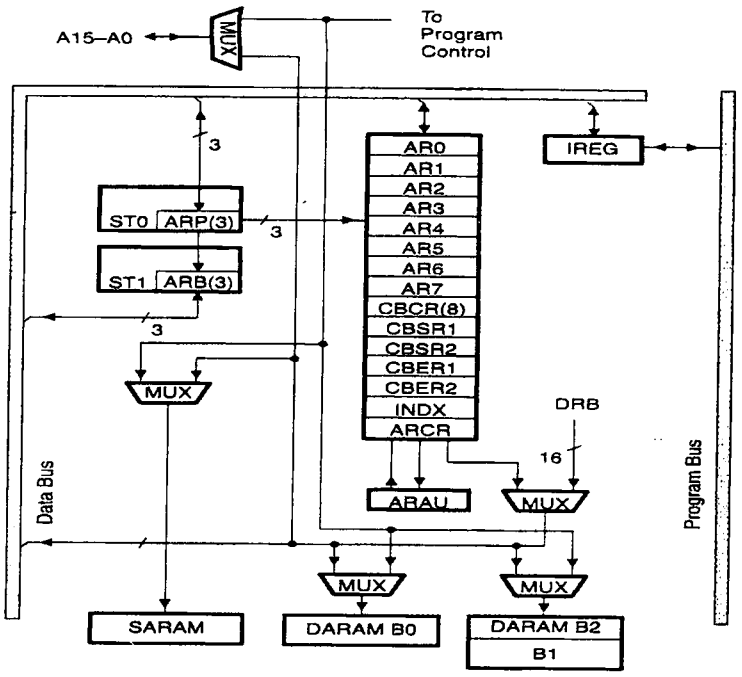
PLU veri alanların da kayıtlı bulunan veriler üzerinde okuma, değiştirme ve yazma işlemi yapabilir. PLU'ya bir giriş veri yolundan, diğeri bilgi program yolundan ani program komutları ile veya DBMR (Dinamik Bit Değiştirme Kaydedici) üzerinden yapılabilir. Daha sonra PLU ilgili komutla belirlenen iki operatör üzerinde bir lojik işlemini uygular. Elde edilen sonuç ilk operatörün getirildiği veri alanına yazılır. Şekil 5.3 PLU'nun blok diyagramını göstermektedir.



Şekil 5.3 Paralel Lojik Ünitesi (PLU)'nun blok diyagramı

5.2.4.3 Yardımcı kütük aritmetik ünitesi (ARAU)

Yardımcı kütük dosyası 8 adet AR0-AR7 (Auxiliary Register) şeklinde adlandırılan kaydedici içermektedir. Bu kütük kümesi yardımı ile dolaylı adresleme yapılabilmekte, gerektiği taktirde bu kütüklerde geçici olarak veri de saklanabilmektedir. Dolaylı adresleme de bu kütüklere işlem yapılacak verinin adresi girilir. İlgili kütük, ARP (Auxiliary Register Pointer) adlı 3 bitlik yardımcı kütük işaretleyicisi yardımı ile seçilir. AR'ler ve ARP'ler veya komut satırından girilen ani bir operatörle, bilgiler; ACC veya PREG'den, veya veri hafızadan yüklenebilirler. AR0-AR7'ye kadar yardımcı kütükler şekil 5.4 ten de görüldüğü gibi, yardımcı kütük aritmetik ünitesine bağlıdırlar. CPU veri hafıza yerleşimi ile adresleniyor iken, ARAU, ilgili AR'yi istediği gibi değitirebilir. Yani ya, +1, -1'le veya indeks kaydedicinin içeriğiyle (INDX) değiştirme işlemini yapar. Sonuç olarak tablo işlemleri yapılırken adres güncelleştirmenin yapılabilmesi için CALU'ya ihtiyaç yoktur.



Şekil 5.4 Yardımcı Kütük Aritmetik Ünitesi blok diyagramı

Yardımcı Kütük Aritmetik Ünitesinin yapmış olduğu işlemleri şu şekilde listeleyebiliriz:

1- İlgili AR + INDX → ilgili AR

2- İlgili AR – INDX → ilgili AR

3- İlgili AR + 1 → ilgili AR

4- İlgili AR - 1 →ilgili AR

5- İlgili AR →ilgili AR

6- İlgili AR+ 8 bitlik değer → ilgili AR

7- İlgili AR - 8 bitlik değer → ilgili AR

8- İlgili AR + rc(INDX) → ilgili AR ;ilgili AR'ye elde bayrağı terslenerek INDX
;ilave edilmesi

9- İlgili AR + rc(INDX) → İlgili AR ;ilgili AR'ye elde bayrağı terslenerek INDX
;çıkartılması

ve ilgili AR değerinin ARCR (Auxillary Register Compare Register) ile karşılaştırılması ile TC (Test/Kontrol Bit'i) değerinin değiştirilmesidir.

BANZ ve BANZD gibi dallanma ve gecikmeli dallanma komutları ile AR'ler kapalı çevrim sayıcılar olarak da kullanılabilirler. Bu kütüklerin yanı sıra dairesel kütükler de mevcuttur. Bu kütükler CBCR (Circular Buffer Control Register) yardımı ile kontrol edilir ve Reset (Yeniden girişe hazır) işleminin yükselen kenarını takiben bütün dairesel kütük içerikleri kaldırılır. Bir dairesel kütüğü işleme sokabilmek için CBSR1 ve CBSR2 kütükleri (Circular

Buffer Start Register) başlangıç adresleri ile; CBER1 ve CBER2 kütükleri (Circular Buffer End Register) bitiş adresleri ile yüklenirler. Daha sonra dairesel kütüklerle birlikte kullanılacak olan ilgili AR başlangıç ve bitiş adreslerinin arasındaki bir adres ile yüklenir. Son olarak da, CBCR'ye ilgili AR değeri yüklenerek kullanılan dairesel kütüklerin çalışması için gerekli izin bit'lerinin devreye sokulması sağlanır. İlgili adres, dairesel kütüklerin arasında yol alırken mevcut durumdaki AR değeri CBER ile karşılaştırılır. Eğer o andaki AR değeri CBER değerine eşit ise ve AR'nin güncelleşmesi söz konusu ise, CBSR değeri otomatik olarak AR'ye yüklenir. Eğer CBER değeri AR'ye eşit değil ise, AR'nin güncelleşmesi belirtilen komuttaki şekilde yapılır. Dairesel kütükler hem azaltma hem de arttırma işlemlerinde kullanılabilir. Azaltma işlemlerinde kullanılacak ise CBSR değeri CBER'den büyük; arttırma işlemlerinde kullanılacak ise küçük olmalıdır.

5.2.4.4 Program Sayacı (PC)

C5x çağırma komutlarıyla kullanılan iç ve dış program hafızasının adresini içeren bir 16-bit Program Sayıcı (PC)'ye sahiptir. PC, PAB vasıtasıyla çipin ya aktif ya da aktif olmayan durumunda program hafızasını adresler. PAB'ın yolu üzerinden önce bir komut Komut Kaydedici (IREG)'e yüklenir ve bu işlemi takiben PC bir sonraki komut çağırma çevrimini başlatmaya hazırdır. Şekil 5.5 PC'nin fonksiyonel blok diyagramını göstermektedir.

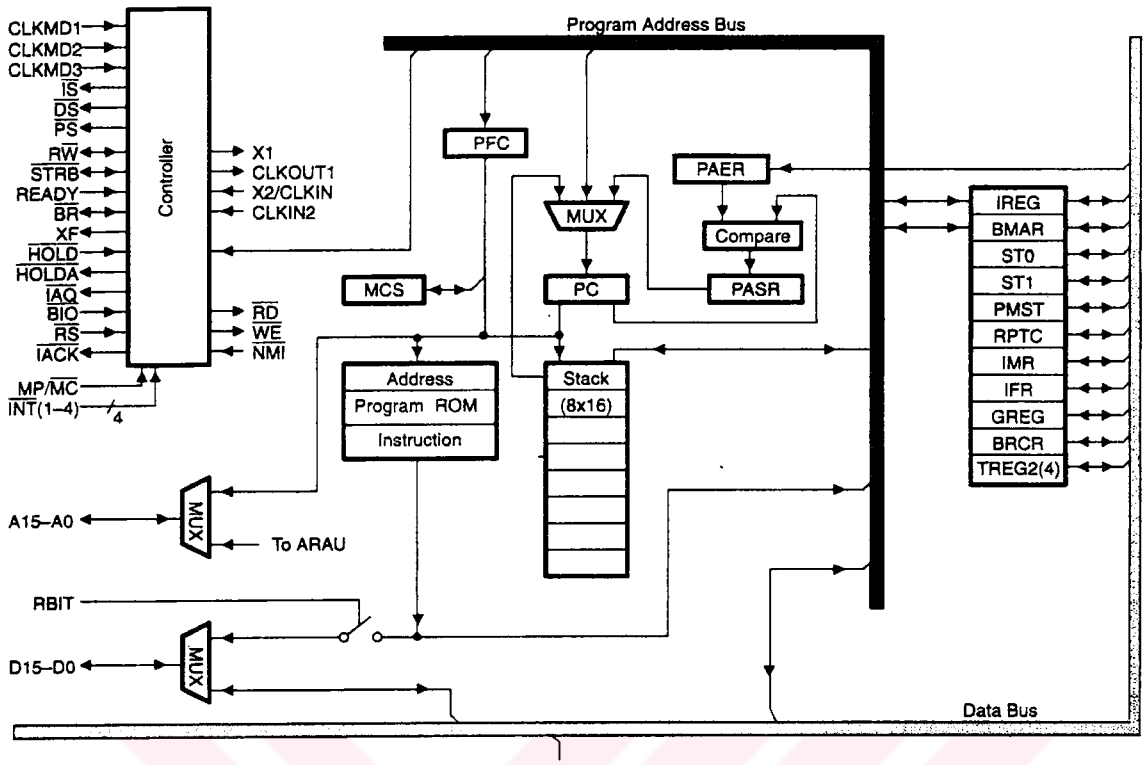
5.2.4.5 Durum ve kontrol kütükleri

5.2.4.5.1 Dairesel Tampon Kontrol Kütüğü (CBCR)

CBCR veri hafıza alanının kütükler bölümünde yer alır ve diğer hafıza alanlarına nasıl bilgi yazılıyorsa burada da aynı işlemler uygulanır. CBCR kütüğü, CALU ve PLU tarafından kontrol edilebilmektedir.

5.2.4.5.2 İşlemci Mod Durum Kütüğü (PMST)

PMST veri hafıza alanının kütüklere ayrılmış kısmında bulunur. Kesme işlemlerinde kullanılmak üzere derinliği 1 olan bir yığın kaydedicisi ile ilişkilendirilmiştir



Şekil 5.5 Program kontrolunun fonksiyonel blok diyagramı

5.2.4.5.3 Durum Kaydedicileri (ST0 ve ST1)

Durum kütükleri veri hafıza alanında bulunurlar. Bu nedenle veri hafıza bölgesinden okunup yazılabilirler. LST (Load Status Register) komutu ST0 ve ST1'e bilgi yazmada; SST (Store Status Register) komutu ise, bu kaydedicilerden bilgi okumada kullanılır. ARP bitleri ile kesme modu, INTM bitleri bu komutlardan etkilenmez. Kesme durumunda, yapılan işlemin son halini kaydetmek üzere bu kaydedicilerin derinliği 1 olan bir yığın kaydedicisi vardır. ST0'ın kesme modu, INTM ve taşma, OVM bitleri ile, ST0'ın C,CNF,HM,SXM,TC ve XF bitleri birbirlerinden bağımsız olarak SETC komutu ile set (girişe hazır) yapılabilir. CLRC komutu ile reset (yeniden girişe hazır) edilebilirler. PM bit'leri ile belirlenen ürün kaydırma modu özellikleri çizelge 5.1'de verilmiştir.

Çizelge 5.1 PM bit'leriyle belirlenen ürün kaydırma modu (Product Shifter Mode)

PM Bit Değerleri		PREG Çıkışı için P-Scaler Kaydırma Modu
BİT 1	BİT 0	
0	0	Herhangi bir kaydırma yok.
0	1	Sola 1 bit kaydırma. LSB sıfır değeri ile doldurulur
1	0	Sola 4 bit kaydırma. 4 LSB sıfır ile doldurulur.
1	1	Sağa 6 bit kaydırma.6 LSB kaybolur. Çarpım işaretlidir.

5.2.4.6 Koşullu işlemler

Koşulsuz dallanmaların yanı sıra C5x işlemcilerde birçok koşullu dallanma, koşullu alt program çağırma ve koşullu alt programlardan dönme komutları bulunur. Bu komutların çalışmasını sağlayan koşulların listesi çizelge 5.2'de verilmiştir.

Çizelge 5.2 Dallanma, Çağırma ve Tekrar komutları için Koşullar

Sembol	Durum	Açıklama
EQ	ACC=0	Akümülatör 0'a eşit
NEQ	ACC≠0	Akümülatör 0'a eşit değil
LT	ACC<0	Akümülatör 0'dan küçük
LEQ	ACC≤0	Akünün içeriği 0'dan küçük veya eşit.
GT	ACC>0	Akümülatör 0'dan büyük
GEQ	ACC≥0	Akümülatör 0'dan büyük veya eşit
NC	C=0	Elde biti 0
C	C=1	Elde biti 1
NOV	OV=0	Akümülatör taşması yok.
OV	OV=1	Akümülatör taşması var.
BIO	BIO' lojik 0	BIO' lojik 0
NTC	TC=0	Test/kontrol bayrağı lojik 0
TC	TC=1	Test/kontrol bayrağı lojik 1
UNC	Yok	Koşulsuz işlem.

5.2.4.6.1 Koşullu dallanma

C5x ile koşullu alt programlara dallanma ve bu alt programlardan koşullu olarak ana programa geri dönme işlemleri yapılabilir. Aşağıdaki program parçası buna örnek olarak verilebilir.

```
*****************************************************************
CC     OVERFLOW,OV          ; taşma oluşursa taşma alt rutinine atlama yap .

OVERFLOW                    ; taşma alt programı

RETC         GEQ            ; ACC>=0 ise ana programa geri dön

RET                         ; geri dön
*****************************************************************
```

5.2.4.6.2 Çok koşullu dallanma

C5x işlemleri ile bir çok koşul yerine getirildiğinde, dallanma yapan komutları da içerirler. Aşağıda verilen çizelge 5.3'de her gruptan bir tane koşul seçilerek bu dallanma komutları işleme sokulabilmektedir.

Çizelge 5.3 Çok Koşullu İşlem Yapan Komut Grupları

Grup 1	Grup 2	Grup 3	Grup 4
EQ	OV	C	TC
NEQ	NOV	NC	NTC
GT			BIO
LT			
GEQ			
LEQ			

5.2.4.6.3 Koşullu çalışma

1 veya 2-kelimelik kod üzerinden geçerek koşullu dallanma yapmak istediğimiz durumlarda, ilgili dallanma XC (Execute Conditionally) komutuyla gerçekleştirilebilir. Bu komutun iki şekli vardır. Bunlardan biri, 1-kelimelik komut XC1, diğeri ise, ya bir 2-kelimelik komut ya da iki 1-kelimelik komut XC2'dir.

5.2.4.7 Tek komutlu işlem yapan tekrar fonksiyonları

Tek bir komut (N+1) kere tekrarlanabilir. Buradaki N sayısı 16 bitlik tekrarlamalı sayıcı kaydedicisine (RPTC), RPT veya RPTZ tekrar komutları ile atılır. Bir komut en fazla 2^{16} kere yani 65536 kere işlenebilir. RPTC programlanamaz, içeriği reset ile temizlenir, RPT veya RPTZ ile yüklenir. Tekrar fonksiyonu kullanıldığı zaman her komut işleminden sonra RPTC değeri 0 oluncaya kadar 1 azaltılır. Yeniden fonksiyon işleme sokulduğu andan itibaren, işlemci döngü bitene kadar tüm kesmelere (maskelenemeyen kesmeler de dahil) kapalı kalır.

5.2.4.8 Blok tekrar fonksiyonları

Bir blokluk komutlar (N+1) kere tekrarlanabilir. Burada da N sayısı benzer bir işlem dahilinde 16 bitlik blok tekrar sayıcı kütüğüne (BRCR)'ye atılır. Bir blok da en fazla 65536 kere işlenebilir. Blok tekrar fonksiyonu, PASR, PAER, BRCR kaydedicileri ile ve PMST özel kaydedicisindeki BRAF bit'i ile kontrol edilir. BRAF bit'i PMST üzerinden set/reset edilebilir.

Blok tekrarlama komutu RPTB işleme başladıktan sonra, otomatik olarak BRAF (Block Repeat Active Flag) biti set edilir. PASR (Program Address Start Register)'a RPTB komutunu takip eden adres yüklenir ve ani uzun adres bilgisi de PAER (Program Address End Register) kaydedicisine yüklenir. Program adres bitiş kütüğüne kaydedilen veri, döngüdeki en son komutu takip eden komutun adres değerinin 1 eksiğidir. Tekrar bloğu en az 3 komut kelimesi içermek zorundadır. Aşağıdaki örnek blok tekrar komutu RPTB'nin kullanımını göstermektedir.

```
*****************************************************************
        SPLK    #0Fh,BRCR       ;tekrar sayısı 16 BRCR'ye atılıyor
        RPTB    END_LOOP-1      ;for I= BRCR;   I>=0; I—
*
        ZAP                     ; ACC=PREG=0
        SQRA    *,AR2           ; PREG=x2
        SPL     SQRX            ; X2 yi kaydet
        MPY     *               ; PREG=b x X
        LTA     SQRX            ; ACC=bX TREG=X2
        MPY     *               ; PREG=aX2
        APAC                    ; ACC=aX2+bX
        ADD     *,0,AR3         ; ACC=aX2+bX+c=Y
        SACL    *,0,AR1         ; Y yi kaydet
        CRGT                    ; maksimum değeri kaydet
END_LOOP
*****************************************************************
```

5.2.4.9 TMS320C50'deki kesmeler

C50 işlemcileri toplam 9 adet iç ve dış kesmeye izin verir. Dış kesmeler dış donanımlar tarafından gerçekleştirilir. Bunların sayısı 4 dür (INT1'—INT4'). İç kesmeler ise şunlardır:
Zamanlayıcı kesmesi (TINT),
Seri kapı kesmeleri (RINT,XINT,TRNT,TXNT,BRNT ve BXNT),
Bunların yanı sıra C50 işlemcisi INTR,NMI ve TRAP gibi yazılım kesmeleri ile RS' ve NMI' gibi maskelenemeyen kesmeler de içerir. Bir dış kesmenin C5x işlemcisi tarafından tanımlanabilmesi için ilgili kesme en az 2 makine çevrimi lojik 1 ve en az 3 makine çevrimi lojik 0 olmalıdır. Bu yöntemle kesme girişlerinin gürültülerden etkilenmesi önlenir. Çizelge 5.4 kesme vektör adreslerini ve önceliklerini göstermektedir.

Çizelge 5.4 Kesme vektör adresleri ve öncelikleri

İSİM	ADRES	ÖNCELİK	FONKSİYONU
RS'	0h	1	Harici maskelenemeyen reset işareti
INT1'	2h	3	1 nolu harici kullanıcı kesmesi
INT2'	4h	4	2 nolu harici kullanıcı kesmesi
INT3'	6h	5	3 nolu harici kullanıcı kesmesi
TINT	8h	6	Zamanlayıcı kesmesi
RINT	Ah	7	Seri port alıcı kesmesi
XINT	Ch	8	Seri port yollayıcı kesmesi
INT4'	12h	11	4 nolu harici kullanıcı kesmesi
TRAP	22h	Yok	Yazılım komutu
NMI'	24h	2	Maskelenemeyen kesme

Kesme vektörleri 2K-kelimelik program sayfalarının başına yazılım yardımı ile yeniden haritalanabilir. Bu işlem PMST saklayıcısındaki IPTR bitleri ile gerçeklenmektedir. Reset işlemini takiben bütün IPTR bitleri sıfırlanır. Yani tüm vektörler program hafızanın 0'ıncı sayfasına atılır. Fakat yazılım sayesinde bunlar istenilen yere taşınabilir.

5.2.4.9.1 Kesme işlemi

Bir kesme oluştuğu zaman 16 bitlik kesme bayrak kütüğündeki ilgili bayrak aktif hale gelir. Kesme işlemini yapan program işleme sokulur sokulmaz ilgili bayrak içeriği temizlenir. Bir kesme işlemi algılandığı zaman şunlar olur:

1- CPU o an yaptığı işlemi tamamlar,
2- Tüm kesmeler kapatılır. (INTM=1),
3- Program sayacının içeriği yığına atılır,
4- Program sayacı kesme hizmet alt programının adresi ile yüklenir,
5- Anahtar kaydediciler gölgelenmiş kaydedicilere yüklenir,
6- IACK' (kesme kabul sinyali) aktif olur,
7- IFR kütüğündeki ilgili kesme bayrağı temizlenir.

5.2.4.9.2 Kesme Bayrak Kütüğü (IFR)

Veri hafızada 06H'de yeri bulunan bir CPU kütüğüdür. CPU tarafından ilgili kesme algılanacağı zamana kadar, bununla ilgili kesme bayrağını set eder. Aşağıdaki olaylardan herhangi biri kesme bayrağının temizlenmesi için yeterlidir.

1- C50 reset edilince,
2- İlgili kesme algılanınca,
3- IFR'deki ilgili bit'e 1 değeri yazılınca.

5.2.4.9.3 Kesme Maskeleme Kütüğü (IMR)

Kesme maskeleme kütüğü veri hafızada yerleşik (04h adresinde) bir kaydedicidir. IMR iç ve dış kesmelerin maskelenmesi için kullanılır. Maskelenemeyen kesmelere herhangi bir etkisi yoktur.

5.2.4.9.4 Kesme modu bit'i (INTM)

ST0 durum kütüğündeki INTM bit'i yardımı ile tüm maskelenebilir kesmeler global olarak devreye sokulur veya çıkartılır.

INTM=0 $\Rightarrow$ Tüm maskelenmemiş kesmeler çalıştırılır.

INTM=1 $\Rightarrow$ Tüm maskelenmemiş kesmeler devre dışı bırakılır.

Aşağıdaki işlemlerden her biri çalışmaya başlayınca INTM biti set olur.

C5x'in reset edilmesi (RS' ayağının aktif olması),

Kesme alt işlemine dallanılması,

Maskelenemeyen kesme komutunun işlenmeye başlanması,

SETC INTM komutunun işlenmesi.

Aşağıdaki işlemlerden herbiri çalışmaya başlayınca INTM biti reset olur.

CLRC INTM komutunun işlenmesi,

RETE komutunun işlenmesi,

5.2.4.9.5 Maskelenemeyen kesmeler

RS' VE NMI' maskelenemeyen kesmeler INTM biti ve IMR kütüğünün içeriğinden etkilenmez. NMI' kesmesi NMI komutu yardımı ile yazılım tarafından da sağlanabilir.

NMI' ucunun aktif olması veya NMI komutunun çalışmaya başlamasıyla aşağıda sıralaması verilen işlemler devreye girer.

1- Merkezi işlem birimi o andaki komutunun işlenmesini tamamlar,
2- Kesmeler global olarak etkisiz kılınır,
3- PC, NMI' vektörünü işaret eder.

NMI' ucunun kesme alt programının çalıştırılması esnasında aktif olabileceği düşünülerek ilgili anahtar kaydedicilerin içerikleri otomatik olarak kaydedilmez. NMI' RPT gibi çok çevrimli komutlarda geciktirilir.

5.2.4.9.6 Yazılım kesmeleri

Tüm 16 CPU kesmeleri belirtilmediği sürece kullanılabilir durumda değildir. İç donanıma veya dış uçlara bağlanmamış kesme vektörleri yazılım kesmeleri olarak kullanılabilir. INTR (Yazılım kesmesi), NMI (Maskelenemeyen kesme), veya TRAP (Yazılım kesmesi) yazılım kesmesi komutları, INTM (Kesme modu) bit'inden veya IMR'(Kesme maskeleme kaydedicisi)'nin içeriğinden etkilenmez.

INTR komutu herhangi bir kesme işleminin, işleme alınmasını sağlayabilir. NMI komutunun çalışma şekli NMI' donanım kesmesi ile aynıdır. Bu komut aynı zamanda program kontrolünü program hafizasının 24h alanına taşır.

5.2.5 Adresleme çeşitleri

5.2.5.1 Direkt adresleme

Direkt adresleme modunda, komut, veri hafiza adresinin en az ağırlıklı alt 7 bitini içerir. Bu 7-bit veri hafıza sayfa işaretleyicisinin (DP) 9-biti ile birleşerek kullanım için 16-bitlik adres sağlamış olur. Veri hafıza sayfa işaretleyicisi, ST0 durum kaydedicisinin içeriğinde bulunur. 16-bitlik veri hafıza adresi iç veri hafıza adres yoluna gönderilir. DP, 512 adet kullanılabilir hafıza sayfalarından birisini ve komut noktalarındaki en az ağırlıklı 7-bit ise, bu veri hafıza sayfalarındaki 128 kelimelik olanlardan birisini işaret ederek seçer. Kullanıcı DP bitlerini LDP komutu ile veya LST #0 komutu ile yükleyebilir.

Makina yeniden giriş yapılabilecek hale getirildiği zaman (reset), DP belirli bir değer de değildir. Yani belirsizdir. Bu nedenle DP'nin de bu işlemlerin problemsiz yürütülebilmesi için belirlenmesi gerekir.

5.2.5.2 Dolaylı adresleme

8 adet yardımcı kaydediciler (AR0-AR7) dolaylı adreslemenin yapılmasında kullanılır. 64K-kelimelik veri hafıza alanından herhangi bir adres AR'lere yüklenen 16-bitlik adres bilgisi ile işlenebilir.

Yukarıda bahsedilen 8 adet yardımcı kaydediciden hangisinin seçileceği ARP (Yardımcı Kaydedici İşaretleyici)'ne yüklenen 0-7 arası sayı ile belirlenir. Program çalışırken ARP'nin gösterdiği değer o an kullanılan yardımcı kaydediciyi gösterir. İlgili AR'ye adres yüklemek için LAR komutu kullanılır. Mevcut durumda kullanılan AR'nin içeriği ise,

1- ADRK komutu ,
2- MAR komutu,
3- SBRK komutu,

ile değiştirilebilir. Herhangi bir komut AR'nin içeriğinin gösterdiği adresteki veriyi işledikten sonra Yardımcı Kaydedici Aritmetik Ünitesi (ARAU) tarafından ilgili AR'nin içeriği arttırılabilir veya azaltılabilir. Bu aritmetik ünite 16-bitlik işaretsiz aritmetik işlemler yapabilir. AR'lere bilgi, veri yolu üzerinden aşağıdaki komutlarla yazılmaktadır.

APL, BLDD, LMMR, OPL, SACH, SACL, SAMM, SMMR, SPLK, XPL.

AR'ler dolaylı adreslemenin yanı sıra aşağıdaki işlemlerde de kullanılabilir.

1- Bu kaydediciler koşullu dallanmaları desteklemede CMPR komutu ile birlikte kullanılırlar. Bu komut ilgili AR'nin içeriğini yardımcı kaydedici karşılaştırma kütüğü

(ARCR) ile karşılaştırır ve sonucu ST1 durum kütüğündeki test/kontrol (TC) bayrağına yazar.

2- Bu kaydediciler bilgilerin geçici olarak saklandığı bölgeler olarak da kullanılabilirler. Bu tip kullanımda veri ilgili kaydediciye LAR (AR'yi yükle) ve SAR (AR'ye depola) komutları ile kaydedilebilir.

5.2.5.2.1 Dolaylı adresleme seçenekleri

1- Herhangi bir artma veya azaltma yok: Komut AR'nin içeriğini, veri hafızayı adresleme de kullanır. Fakat komutun işlenmesini takiben AR'nin içeriğinde herhangi bir değişim olmaz. Assembler dilinde komutlara eklenen takı, *'dır,

2- 1 arttırma, 1 azaltma : Komut AR'nin içeriğini, veri hafızayı adreslemede kullanır. Fakat komutun uygulanmasını takiben AR'nin içeriği 1 arttırılır veya 1 azaltılır. Assembler dilinde komutlara eklenen takı, *+ veya *- 'dır,

3- İndeks oranında artma veya azaltma : Komut AR'nin içeriğini, veri hafızayı adreslemede kullanır. Fakat komutun uygulanmasını takiben AR'nin içeriği INDX oranında azaltılır veya arttırılır. Assembler dilinde komutlara eklenen takı, *0+ veya *0- 'dır,

4- Ters elde kullanarak indeks oranında artma veya azaltma :
Komut AR'nin içeriğini, veri hafızayı adreslemede kullanır. Fakat komutun uygulanmasını takiben AR'nin içeriği INDX oranında azaltılır veya arttırılır. İlgili hafıza adresinin uygulanmasını takiben INDX'in içeriği elde bayrağını ters işaretle yükleyerek, AR'nin içeriğine eklenir veya çıkartılır. Assembler dilinde komutlara eklenen takı, *BR0+ veya *BR0-'dır.

5.2.5.3 Hızlı adresleme

Bu adresleme şeklinde, komut kelimesi hızlı operatörün değerinin içerir. C5x 1-kelimelik (8-bit, 9-bit ve 13-bit sabit) kısa hızlı komutlar ve 2-kelimelik (16-bit sabit) uzun hızlı komutlara sahiptir. Bu komutların dağılımı aşağıda verildiği gibidir.

8-bit sabit: ADD, ADRK, LACL, LAR, RPT, SBRK, SUB,

9-bit sabit: LDP,

13-bit sabit: MPY,

16-bit sabit: ADD, AND, APL, CPL, LACC, LAR, MPY, OPL, OR, RPT, RPTZ, SPLK, SUB, XOR, XPL.

5.2.5.4 Sadece bir işlem gerçekleştiren kaydedicili adresleme

Bu adresleme şekli uzun hızlı adresleme moduna benzemekle birlikte, ilgili adres CPU'daki iki adet özel amaçlı hafıza-haritalanmış kaydedicilerin birisinden gelir. Bu adreslemenin en önemli özelliği, çalışmakta olan program süresince aktif olarak hafıza bloğunun adresini değiştirebilmesidir. Bu iki özel kaydedici ise, BMAR ve DBMR'dir.

5.2.5.5 Hafıza-haritalanmış kaydedicili adresleme

Bu adresleme de, ilgili veri sayfa işaretleyici değerini etkilemeksizin mevcut hafıza-haritalanmış kaydediciler değiştirilebilir. Buna ilave olarak, herhangi bir RAM yerleşimi veya veri sayfa numarasında gereken değişiklik yapılabilir. Aşağıda bu çeşit adresleme için DP'nin içeriğini etkilemeyen komutları verilmektedir.
LAMM, LMMR, SAMM ve SMMR.

5.2.5.6 Tampon geçiş kaydedicileri üzerinden dolaylı adresleme

Sonlu darbe cevabı (FIR) filtresi gibi birçok algoritma, işlenen en son veriyi içeren düzgün bir pencere oluşturmak için hafıza da bu çeşit bir adresleme yapabilir. C5x AR'ler aracılığıyla aynı zamanlı olarak oluşan iki dolaylı tampon çalışmayı yürütebilmektedir. Bu çalışma şeklini gerçekleştiren hafıza-haritalanmış 5 adet kaydedici şunlardır.
CBSR1, CBSR2, CBER1, CBER2 ve CBCR.

5.3 DSP Mimarisini Oluşturan Ana Üniteler

DSP'nin genel yapısı hakkında yukarıdaki bilgileri verdikten sonra sırasıyla DSP mimarisini oluşturan ana üniteler ve yapılan çalışmada kullanılan TMS320C50 mimarisi ile ilgili bilgilerde bu alt bölümlerde verilecektir.

5.3.1 Çarpım ünitesi

Diğer mikroişlemcilerden farklı olarak, DSP kendi donanımına yerleştirilmiş olan çarpım ünitesi sayesinde, çarpma işlemleri hızlı bir şekilde yapabilmektedir. Çarpım ünitesi, tek çevrim süresinde, girişine uygulanan sayıların paralel dizi çarpma işlemini gerçekleyen sistem olarak tanımlanabilir. Girişine uygulanan sayıların kontrolunu yaparak, işaretsiz, işaretli ve

karışık sayılar üzerinde çarpma işlemini gerçekleştirebilmektedir. Sonuç üzerinde yuvarlatma yaparak da veri kaybının minimum düzeyde kalmasını sağlanmaktadır. Mikroişlemcilerden farklı bir donanıma sahip olduğu yapısı ile işlevini gerçekleştiren çarpım ünitesi, DSP'ye tartışılmaz bir üstünlük kazandırmaktadır. Mikroişlemciler de, çarpma işleminin yazılım olarak gerçekleştirilmesi sırasında, Merkezi İşlem Birimi (CPU) ile haberleşme anında meydana gelen zaman kaybı, adres ve verilerin aynı yol üzerinden yollanmak zorunda kalınmasından dolayı daha da artmaktadır. Çarpım işleminin tek saat çevriminde tamamlanması; diğer üniteler ile paralel çalışma ve veri transferini mümkün kılmaktadır.

Çarpım ünitesinin hızı, RAM belleğe erişim hızı ile karşılaştırma yapmaya uygun ve paralel işlem yapabilmeye yatkın olmalıdır. Çarpım ünitesinden istenen bir diğer özellikte girişindeki sayıların kontrolunu yapabilmesidir. Bilgi işlem sistemlerinde kullanılan sayılar, işaretsiz veya ikinin tümleyeni, tamsayı veya kesirli yapıda olmaktadır. Bu tür sayıların birbirine dönüşümünü en az hata ile sağlamalıdır. Sonuçta elde edilen değerin yine istenen formatta, tamsayı veya kesirli olması gerekir. Ayrıca sonucun genişletilmiş veya yuvarlatılmış seçeneği ile hata kontrolunu elinde bulundurmalıdır. Çarpım işleminin tek saat çevriminde sonuçlanması, paralel çalışmaya olanak sağlamakta; kullanılan kütükler, sistemin minimum gizlilik içinde kullanıcıya görünür bir yapıda çalışmasını kolaylaştırmaktadır.

5.3.2 Çarpım-Akümülatör ünitesi (MAC)

İlk üretilen DSP'ler de tek başına bulunan çarpım ünitesi, daha sonraki nesil DSP'ler de akümülatör ile birleştirilmiştir. Dijital filtreler, FFT (hızlı Fourier dönüşümü) algoritmaları ve vektörel işlemler gibi birçok uygulamada karşılaşılan karmaşık işlemin kolaylıkla gerçekleştirilmesi için bu ünite kullanılmaktadır. DSP'ler, donanım içinde yer alan çarpım üniteleri ile birleşerek, çarpma-depolama işleminin tek saat çevriminde gerçekleşmesini sağlamaktadır. Böylece çok daha hızlı işlem yapabilme yeteneğine kavuşulmuştur. MAC'ın yapısında bulunan geri besleme yolları ve paralel çalışmanın gerçekleşmesini sağlayan kütükler yardımı ile, tekrarlanan yapıdaki işlemler kolayca yerine getirilmektedir. MAC ünitesinin en önemli özelliği, çarpma-biriktirme işlemini tek saat çevriminde gerçekleştirmesidir. Biriktirme işlemi sırasında, sonucun doğruluğunu arttırmak için akümülatörün çıkış kütüklerinde, büyütme bitleri adı verilen koruma bitleri bulunmaktadır. Çarpım ünitesinin çıkışının genişliği ile uyumlu bit sayısı bulunmaktadır.

MAC ünitesinde yapılan işlemlerde, istenen bir diğer özellikte taşma ve yuvarlama kontrolu yapabilmesidir. Taşmanın daha önceden algılanabilmesi, sisteme eklenecek bir lojik birim ile sağlanabilir.

5.3.3 Aritmetik-Lojik işlem ünitesi

DSP sistemlerinde Aritmetik-Lojik İşlem Ünitesi (ALU), mikroişlemcilerde olduğu gibi, aritmetik ve lojik işlemleri gerçekleştiren birimdir. Bilgi işlemede önemli bir yeri olan lojik işlemler bu birim tarafından gerçekleştirilmektedir. DSP'lerdeki ALU ünitesinin farkı ise, tek saat çevriminde işlem yapabilmesidir. Bunun yanında, MAC ünitesinde olduğu gibi, tekrarlanan işlemlerde paralel çalışma denilen yapıya izin vererek, verilerin sıkışmasını önlemektedir. DSP, ölçeklendirme birimleri ile desteklenerek, verinin maksimum dinamik aralığını korumasını sağlamaktadır.

ALU ünitesinden istenen en önemli özellik, bit işleme ve karakter tutma işlemlerini gerçekleştirirken, hızlı işlem yapmasıdır. Bu birim, mikroişlemci ile karşılaştırıldığında, daha kısıtlı komut seti ve adresleme modları olmasına rağmen, tek saat çevriminde işlem yapma ve paralel çalışma sayesinde, temel işlemleri mümkün olan en kısa sürede tamamlayabilmektedir.ALU'dan istenen bir diğer özellik ise, işlem yapılmadan önce, genel sistem durumu hakkında, durum bitleri yardımı ile, bilgi vermesidir. Veri yollanması ve işlenmesi sırasında verimli çalışabilmesi, işlem yaparken girişteki bayrakların durumuna göre sonucu kaydırabilmesi de DSP'nin hızını arttıran diğer özellikler olarak sıralanabilir.

5.3.4 Yığın kaydırma ünitesi

Kaydırma ünitesi (BS), işlem sırasında taşma ve veri kaybı sorunlarını önlemek amacı ile, sayıları ölçeklendirmek için ve sabit nokta ile kayan nokta arasında dönüşüm yapabilmek için kullanılan bir ünitedir. Mikroişlemcilerdeki kaydırma ünites, her bir çevrimde bir bit kaydırma yaparken, DSP'deki ünite, tek saat çevrimi içinde birçok bitin kaydırılmasına olanak sağlayarak, hızı arttırmada etkili bir rol oynamaktadır. Kaydırma ünitesi, aritmatik ve lojik kaydırma işlemleri yapabilir. Aritmatik kaydırma işlemi, sağa ve sola kaydırma ile sayıların ölçeklenmesini sağlar. Lojik kaydırma ise, veriyi işaretsiz olarak kabul eder ve dönüş yönüne göre, sayıyı sıfır ile doldurur. Bu ünite, mimari yapısından dolayı, uzun kelimelerin iki çevrim süresi içinde döndürme işlemini tamamlayabilir. Bu ünitenin gerçekleştirdiği en önemli işlemler, normalize, denormalize, kayan nokta bloklama olarak sıralanabilir.

5.3.5 Veri adres jeneratörü ünitesi

Veri adres jeneratörü ünitesi (DAG), bellek üzerindeki adreslere belirli sayı değerleri atayan kütükler içeren bir yapı görünümündedir.Bu sayede veri okunup yazılması çok daha hızlı

olmaktadır. DSP'ler de adres jeneratörünün ayrı bir yapı olması, algoritmanın işlenmesi sırasında, aynı anda birkaç verinin işlenebilir olması ve ünitelerin birbirlerinden bağımsız paralel bir yapıda çalışabilmesini sağlamaktadır. Bu özellik aynı zaman da sistemin hızını katlamakta, hem de karmaşık algoritmalar ile işlemler kolayca yürütülebilmektedir. Yaptığı işlemler arasında, veri belleğine adres gönderebilmesi, hesaplanmış bir başlangıç değerinden adres üretebilmesi, lojik işlemler ve kaydırma yapabilmesidir. Maskeleme işlemlerinin gerçeklenmesi, FFT gibi karmaşık adresleme gerektiren uygulamalarda da bu ünite birçok kolaylık sağlamaktadır.

5.3.6 Komut sıralama ünitesi

Komut sıralama ünitesi (SEQ), program akışını belirleyen komut adreslerini üretir. Bu ünite, Program Sayacı (PC)'nin arttırılması, alt program çağrılması ve dış kesmelerin yürütülmesi görevlerini üstlenmiştir. Bu ünitenin asıl fonksiyonları; her komuttan sonra PC'yi bir arttırarak, program akışını sağlamak, alt program adreslemesini düzenlemek ve geri dönüldüğünde adreslerin işlemesini sağlamak, Veri taşmasının düzeltilmesi gerektiğinde gerekli olan alt program parçalarını çağırmak, dış I/O elemanları ile haberleşebilmek için gereken kesmeleri sağlamak, kesme servis programlarına dallanmak şeklinde sıralanabilir.

5.4 DSP Sistemlerin Seçimi ve Karşılaştırılması

Kontrol edilecek sisteme uygun bir denetleyici seçimi, bir tasarımcının bütün sistemin kurulması aşamasında göz önüne alması gereken en önemli bölümdür. Kontrol edilen sistemin karakteristiğine göre, DSP sisteminin yapısal özellikleri seçilmelidir. İsteğe cevap vermeyen bir kontrol yapısı, yanlış kontrola neden olacaktır. Örneğin sabit nokta ile çalışabilecek yapıdaki bir sistemde kayan nokta kullanılması, sistemin maliyetini arttıracaktır. Kontrol edilen sistemin dinamik sayı aralığının büyük olması veya sayısal hataların getireceği sorunlara karşı aşırı duyarlı olması durumunda, kayan nokta işlemcinin kullanılması, sistemin daha yüksek bir performansta çalışmasını sağlayacak, maliyet fonsiyonunu dengeleyecektir.
DSP sistemleri, mikroişlemci ve mikrodenetleyici olmak üzere iki ana yapıda toplanabilir. Mikroişlemci yapısı; sadece DSP yapısını içermekte, yapının içinde herhangi bir bellek birimi veya seri kapı, A/D birimi gibi giriş çıkış birimi bulunmamaktadır. Yapısında herhangi bir yan birim bulunmaması, hızlı işlem yapma açısından büyük yarar sağlamaktadır. DSP ne kadar hızlı veri işlerse işlesin, veriyi aktarırken erişebildiği hız ancak bellek biriminin hızı kadardır. Bu yüzden kullanılan yardımcı birimler genel kontrol yapısına uygun seçilmelidir.

Bu yardımcı birimlere erişim süresi, sistemin hızını olumsuz olarak etkilemektedir. Bu yüzden mikroişlemci yapıdaki DSP'lerin, kütük dosyaları ve komut setleri yeterince büyük yapılarak, bellek birimlerinin verimli kullanılması yoluna gidilmektedir. Böylece sistemin hızını azaltacak yapısal değişiklikler mümkün olduğu kadar aza indirgenecektir.

Mikrodenetleyici yapıdaki DSP sistemleri ise, bellek ve yardımcı birimleri bünyesinde bulundurmaktadır. Bu sayede belirli boyuttaki programlar dışarıdan bir bellek birimine gerek duyulmadan saklanabilmektedir. Yapı içine konulan bellekler, çok kısıtlı boyutta olmasına rağmen, basit uygulamalar için ideal bir çözüm sunmaktadır. Daha geniş bellek ihtiyacı, dış bellek birimleri ile karşılanabilir. Kontrol edilen sistemin parametrelerinin gözlemlenebilmesi için yardımcı birimlere ihtiyaç duyulmaktadır. Yapısında seri bir kapı bulunduran DSP sistemleri, kontrola daha uygun bir alternatif oluşturmaktadır. Analog bir büyüklük, dijital bir veriye dönüştürülerek, sistemin üzerinde işlem yapabileceği bir hale getirilmelidir. A/D çevirici, özellikle motor kontrolunda gerek duyulan bir yapıdır. Yapısında A/D çevirici bulunduran sistemlerde, kontrol edilen motorun akımı belirli bir dönüşüm ile algılanarak kontrol altında tutulabilir. Bu gibi alt birimleri yapısında bulunduran bir DSP, dış birimlere minimum gereksinim duyacak şekilde çalışmaktadır. Mikrodenetleyici yapısındaki DSP, mikroişlemci türüne göre daha yavaş işlem yapsa da, mikroişlemcinin dış birimlerle haberleşirken harcadığı zaman bu açığı kapatacaktır. Yalnız içinde bellek birimi bulundurmayan yapı, ek mimari yapılar için gerekli alana sahip olacağından, daha kuvvetli bir işlem kapasitesi sağlanabilir.

DSP sistemlerin kontrol edilen sistemin gereksinimini karşılayabilmesi için mimari yapısının belirli özellikleri taşıması gerekmektedir. Kontrol edilen sistemin ihtiyacının karşılanması sistemin tanınması ile başlamaktadır. DSP'den yeterli verimin alınması, mimarisinde belirli yapıların bulunması ile sağlanabilir. Sistemin maksimum verim ile çalışabilmesi için aritmatik işlemleri yürüten ALU, MAC ve kaydırma üniteleri, veri akışının kontrolunu yapan DAG ve SEQ üniteleri bulunmaktadır. Bu ünitelerden birinin eksik olması veya yeterli performansta çalışmaması sistemin performansını büyük oranda düşürecektir. Örneğin; kaydırma ünitesindeki etkisiz çalışma durumu, kayan nokta işlemlerinin düşük verimle çözümlenmesi anlamına gelmektedir. Veri aktarımı ve akış kontrolunu sağlayan üniteler, sistemin hızını doğrudan etkilemektedirler. Bu ünitelerin yapısındaki bazı eksiklikler, hızın birkaç kat azalmasına neden olmaktadır. Bu ünitelerle beraber sistemin çalışmasına etki eden bir diğer etken de birimleri bağlayan yollardır. Yolların yeterince geniş olmaması, hızı azaltacağı gibi, aritmatik işlemler sonunda verinin tam olarak iletilememesi hatayı arttıracaktır.

Genel kontrol uygulamalarında mikroişlemci yerine DSP'lerin kullanılması uygulamanın ihtiyacının değişmesiyle belirginleşmiştir. DSP'lerin mikroişlemcilere göre tercih edildiği

uygulamalar, matematik işlem yönünden çok kapsamlı olanlarıdır. Dijital işaret işleme sistemlerinin bir diğer özelliği de gerçek zamanda işlem ihtiyacı göstermeleridir. Kullanıcıya belirli bir gecikme ile ulaşan bir işaret, sistemin hatalı çalışmasına neden olacaktır. Örneğin, motor kontrol sisteminde yük üzerindeki ani bir değişikliğin kontrol sistemi tarafından zamanında algılanamaması sistemin yanliş cevap vermesine neden olacak, motorun tahrip olmasına kadar gidebilecek problemler ortaya çıkaracaktır. Bu yüzden, DSP sistemleri gerçek zamanda veri işleyecek yapıda olmalıdır.

Dijital işaret işlemenin temel problemi işaretin örneklenmesidir. İşaretin doğru olarak algılanması, örnekleme frekensının yeterince büyük olmasına bağlıdır. Kontrol uygulamalarında kullanılması gereken işlemci, sistemin bant genişliğinin en az on katı bir hızla örnekleme yapabilmelidir. Ancak bu şekilde, işaretin anlamını kaybetmemesi sağlanabilir. DSP sistemleri işareti yeterince yüksek hızda örnekleyebilmeli ve elde edilen verileri saklayabilmelidir. Uygulamanın yapısına göre örnekleme frekansı da artmaktadır. Görüntü ve ses işleme sistemlerinde çok yüksek örnekleme frekanslarına ihtiyaç duyulurken, kontrol uygulamalarında daha düşük frekanslar yeterli olmaktadır. Kontrol uygulamalarında 1 kHz örnekleme frekansı yeterli olmasına karşılık haberleşme uygulamalarında 8 kHz, ses işlemede 8-10 kHz, müzik işleme uygulamalarında 40-48 kHz, video görüntü işlemede ise 14 MHz örnekleme frekanslarına gerek duyulmaktadır.

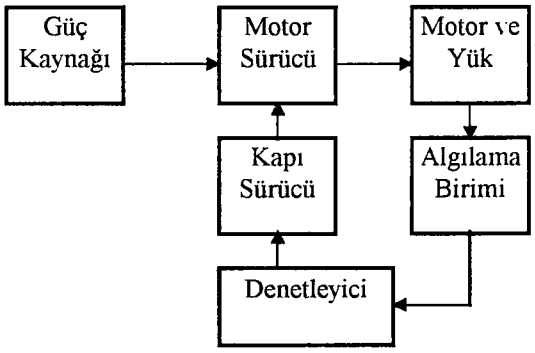
DSP sistemlerinin geliştirilmesi ile işaret işleme çok daha kolay ve ucuz bir çözüm olanağına kavuşmuştur.

5.5 DSP Uygulama Alanları

VLSI teknolojisindeki gelişmeler, DSP sistemlerinin çok kısa bir süre içinde kontrol uygulamalarının bütün alanlarına girmesine olanak sağlamıştır. DSP'nin genel işaret işleme alanında, dijital filtreleme, konvolusyon, korelosyon, hilbert dönüşümleri, FFT (hızlı fourier dönüşümü), adaptif filtreleme, ve sinyal üretimi gibi geniş uygulama alanları bulunmaktadır. Bunların başlıcaları; görme sistemleri, spektrum analizi, ses tanıma, robot ve motor kontrolu, radar işleme, modemler ve cep telefon sistemleri, adaptif yol kontrolu, işitme cihazlarıdır. Bütün bu örneklerden de anlaşıldığı gibi, dijital sistemlerin olduğu her alanda DSP kullanılmaya başlanmıştır.

Motor kontrol sistemlerinde DSP'nin kullanılması diğer uygulama alanları ile karşılaştırıldığında oldukça geç olmuştur. Bunun nedeni, mikroişlemcilerin yeterli düzeyde bir performans göstermesi ve maliyeti düşürerek, her türlü uygulamaya uyumlu olabilmesidir. Ancak motorların klasik kontrol algoritmaları yerine karmaşık kontrol algoritmaları ile

kontrolu söz konusu olduğunda, mikroişlemciler zayıf kalmaktadır. İşlem sayısı arttıkça mikroişlemci yavaşlamakta, bu da kontrol edebileceği sistemlerin performansını etkilemektedir. Bu gibi sorunlara çözüm olarak, mikroişlemcilerin paralel çalışması, kontrol edilen sistemin belirli aşamalarında ayrı işlemcilerin kullanılması yoluna gidilmişse de, hem böyle bir sistemin kontrolunun ve uygulanabilirliğinin zor olması, hem de maliyet artışı DSP'leri üstün hale getirmiştir. Şekil 5.6'da motor kontrol sisteminin genel birimleri gösterilmiştir.



Şekil 5.6 Motor kontrol sistemin birimleri

Kontrol edilecek sistemin özelliklerine uygun bir güç elektroniği devresi kullanılmalıdır. Alternatif akım şebekesinden beslenme durumunda güç akışını kontrol etmek amacıyla güç kaynağı katı kullanılmaktadır. Alternatif gerilim, doğrultucu ile doğrultulduktan sonra eğer gerekiyorsa gerilimin genliğini ayarlamak için doğru akım kıyıcısı kullanılabilir. Daha sonra kullanılan motora göre güç elektroniği güç katı devresi gelmektedir. Asenkron ve fırçasız doğru akım motor kontrolunda inverter kullanılır. Güç katının yapısı nasıl olursa olsun devrede kullanılan güç elektroniği anahtarlarını sürmek için sürücü devreye gerek vardır. Burada anahtarlama elemanı olarak IGBT kullanılmıştır. Gerilim kontrollu olan bu ve bunun gibi kontrol elemanların yüksek anahtarlama frekanslarına ulaşması, devrenin PWM işaretinin daha yüksek frekanslarda elde edilmesini sağlamaktadır.

Kontrol sistemindeki bir diğer önemli eleman da algılayıcılardır. Motorda hız veya konum kontrolu yapıldığında, kapalı çevrim bir kontrol için denetleyicinin gerekli olan bilgiyi almasını algılayıcılar sağlamaktadır (akım ve konum sensörleri gibi). Hız ve konum bilgisi analog işaretler olmasına rağmen, denetleyicinin dijital yapıda olması nedeniyle kullanılan algılayıcıların çıkışı dijital seviyededir.

Denetleyici yapısı ise, sistemden bağımsızdır, sisteme göre değişen sadece işlemcinin programıdır. Bu esnek yapı sayesinde, değişik uygulamalar da donanım yapısında bir değişiklik yapmadan sadece yazılımı değiştirerek kulllanılabilir. Tasarlanan kontrol sisteminin

yapısından bağımsız olarak, gerçekleştirilen yazılımın belirli bölümleri aynı kalmaktadır. Kurulan algoritmanın denetleyici sistemine hangi dil kullanılarak programlanacağı bir diğer tasarım aşamasıdır. DSP yazılım sistemleri, makina kodu yanında C dilini desteklemekte, böylece programın bu dilde yazılıp, işletilmesine olanak sağlamaktadır. Yalnız sistemin hızından tam olarak yararlanma durumu için, gerçek zamanda kontrol uygulamalarında işlemcinin makina dilinde programlanması tercih edilmelidir. Düşük seviyeli diller yardımıyla, işlemcinin bütün mimari üstünlükleri göz önüne alınıp, program gerçekleştirilmelidir.

Akış diyagramı belirlenen program, ilk olarak sistemin sabit ve değişkenlerinin tanımlanması aşamasını gerçeklemelidir. İşlemcinin bellek ve kütük yapıları gereken şekilde tanımlandıktan sonra, zamanlayıcı ve kesme işlemlerine izin verilmelidir. İşlemcinin zamanlama ünitesi gerçek zamandaki işaretleri algılamak ve işlemek amacıyla değişik kesme işlemlerini gerçekleyebilmelidir. Kapalı çevrim motor kontrol sisteminde, algılayıcı ünitesinden gelen işaretler zamanlayıcı ile değerlendirilmekte ve gereken işlemler yapılmaktadır. Zamanlama ve kesme işlemleri birer vektör yapısında olup, alt program gibi işlemektedir.

Sistem veri işleyecek ve saklayacak duruma geldiğinde, A/D çeviriciden gelen işaretler değerlendirilmeye alınmaktadır. Bu veriler bellekte veya kütüklerde saklanarak, kontrol işleminin hesabında kullanılmaktadır. Kullanılan kontrol yöntemine göre, veriler işlenip, gereken kontrol işareti üretildiğinde, D/A çeviriciye yollanarak fiziksel dünya ile gereken dilde anlaşma kurulmaktadır. Kurulan program yapısı, sonsuz döngü şeklinde çalışırken, herhangi bir birimden gelen dallanma emri ile normal çalışmasına ara vermektedir.

Kontrol sistemlerinde, kontrol edilen sistemin davranışını düzenlemek amacıyla denetleyiciler kullanılmaktadır. Klasik kontrol yapısı olarak sistemlerde en çok kullanılan yapılardan biri PI (oranlama ve entegre etme) yapısıdır. Sistemin tam ve doğru olarak kontrolunu gerçekleştiren bu yapı, istenen kontrol cevabını parametrelerini değiştirerek sağlamaktadır. Yapının bu kadar geniş uygulama alanı bulmasının sebebi, basit bir yapıda olması ve kolaylıkla sisteme uyumlu hale getirilebilmesidir.

6. DONANIM VE YAZILIM UYGULAMASI

6.1 Giriş

Yüksek bir performans geçişi ile cevap ihtiyacı duyan bir sisteme uygulanan bir kontrolör hızlı ve doğru olmalıdır. Mekaniksel olarak düşük bir frekans bandına sahip olduğu düşünülen bir makinanın geçici olarak çalışması istendiğinde çok hızlı hesaplamlara ihtiyaç duyulur. Bu noktada makinaya uygulanan kontrol devresi önemli bir rol oynar. Buna göre uygun gerçek zaman yazılımı geliştirilmelidir.

Bölüm 2'de asenkron makinanın analizi, bölüm 3'te ise bu makinanın matematiksel modeli, uygulanan kontrol şekli ve PI kontrolörün matematiksel ifadeleri verilmiştir.

Bir hız/pozisyon kontrol tasarımı gerçek zamanda etkin açısal hız/pozisyon ve akım üretimi ve işlenmesine gereksinim duyar. Yani daha hızlı kontrol için, daha hızlı sistem çalışmasına ihtiyaç vardır. Bunun için de şaft pozisyonunu eş zamanda dijital kod olarak izleme avantajına sahip olan enkoder kullanılmalıdır. Bu enkoder iki veya üç led'le üretilen darbelerle herbir geçiş zamanı bir slot olmak üzere bir darbe üretir. Bu anlatılanlara bağlı olarak tüm algoritmanın tamamlanma süresine t_{top} diyecek olursak, bu süre iki slot arasındaki minimum zaman aralığından daha az olmalıdır.

$$t_{top}<1/(n_{max}*s) \tag{6.1}$$

$$ss=2^m \tag{6.2}$$

burada, n_{max} maksimum kontrol edilebilir hız, s slot sayısı ve m ikili dijit sayısıdır. Denklem 6.2 s slot'luk bir enkoder da uygulanabilir ikili dijit ile slot sayısı arasındaki bağıntıyı göstermektedir. Yapılan çalışmada sadece hız değerlerini sayısal olarak hesaplayan birim kullanılmış olup, program yapısında ise enkoderden veri alma şekline uygun olarak gerekli yazılım yapılmıştır. Alınan hız değerleri YSA ile kontrol için yeterli olmaktadır.

6.2 Sistem Kontrol Algoritması

Bir 3-faz dengeli uyarma tasarımı, iki faz akımı ve şaft pozisyon değerlerinin ölçülmesine (bu çalışmada sadece hız sayısal değerleri) ihtiyaç duyar. 1800 d/d'dan az hız değişimine sahip olan makinalarda şu metod uygulanır:

Metod: İlkönce iki ardışık sayıcı okumaları arasındaki hız/pozisyon ve zaman farkı okunur. Buradan gerçek hızı bulmak için bir bölme işlemi yapılır ve bu yazılım böylece önemli derecede çevrim hızının değerini azaltmaya sağlar.

Bu algoritma, d-q referans çerçeve akımlarını stator referans faz akımlarına dönüştürmek için iki adet look-up tablosuna gereksinim duyar. Bunlardan biri, $Cos/Sin(\theta_m+\theta_r)$ 'i gerçekleştirmekte, diğeri ise, $|\vec{i}_s| = i^*_{sq} \sec\theta_r$ denklemini kullanarak referans stator akım vektörünün genliğini hesaplamak içindir. Moment açısının kontrolü, bölüm 3'de de bahsedildiği gibi, d eksen akım bileşeni kendi oransal değerinde sabit tutularak, PI hız kontrolörle üretilen ve sisteme yerleştirilmiş olan q ekseni referans akım değeriyle yapılır. Referans faz akımları aşağıdaki gibi sıralanır.

$$|\vec{i}_s| = i^*_{sq} \sec\theta_r$$

$$i^*_{sa} = |\vec{i}_s| \cos(\theta_m+\theta_r)$$

$$i^*_{sb} = |\vec{i}_s| \cos(\theta_m-120+\theta_r) \qquad (6.3)$$

$$i^*_{sc} = |\vec{i}_s| \cos(\theta_m+120+\theta_r)$$

$$\theta_r = \tan^{-1}(i^*_{sd}/i^*_{sq})$$

İşlemciyle veri kaynağı arasındaki iletişim iki yönde olur:

1- Yeni bir veri örnek grubunun (i_{sa}, i_{sb} ve θ_m) hazır olduğunu işlemciye bildirmek için bir kesme vektörü sağlamak (Kesme kullanılması),

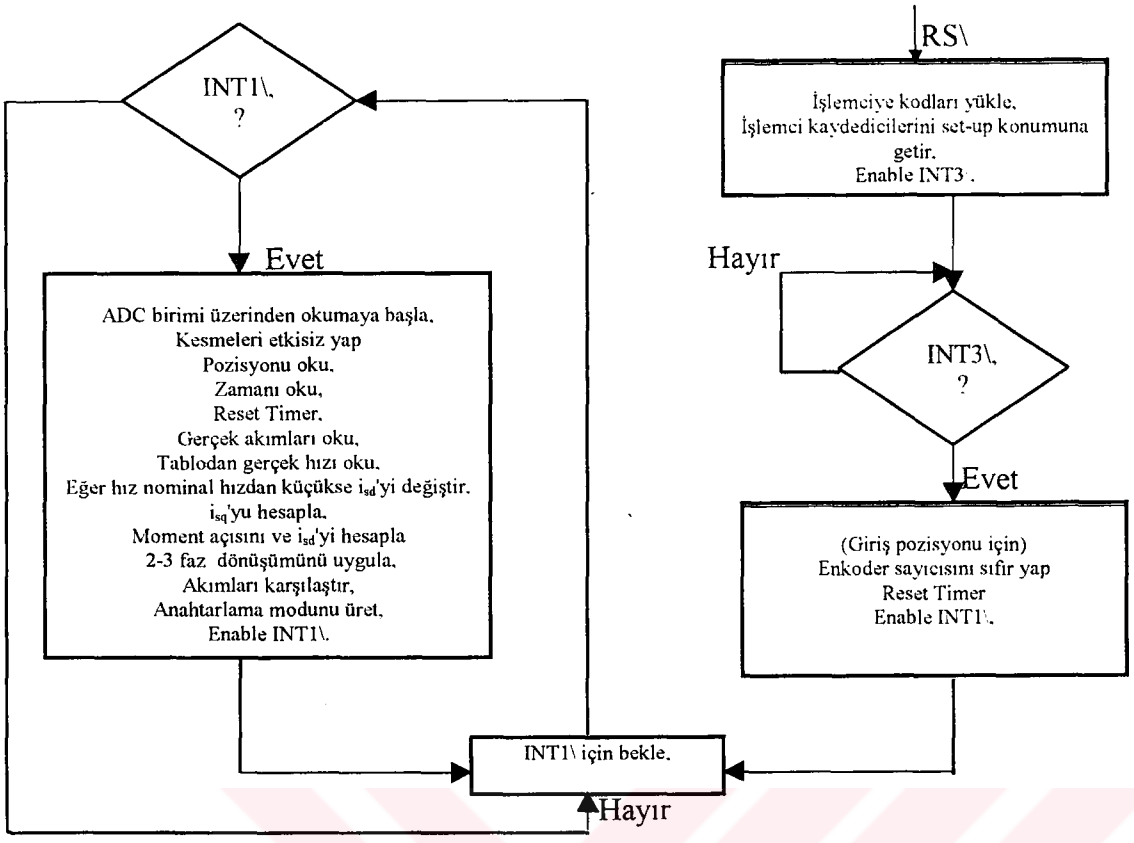
2- Veride oluşacak bir farkı izleme işlemine göre, algoritmayı çalıştırmak (Altyordam kullanılması).

Sistem kontrolüne yukarıda verilenlerden birincisi adapte edilmiştir. Fakat Başlangıç Kit (Starter Kit)'i kullanıldığından yapılan işlemin test edilmesi açısından ise, ikinci yaklaşım kullanılmıştır. Kesme kullanılması durumuna ait akış diyagramı şekil 6.1'de, altyordam kullanılmasına ait akış diyagramı ise şekil 6.2'de verilmiştir.

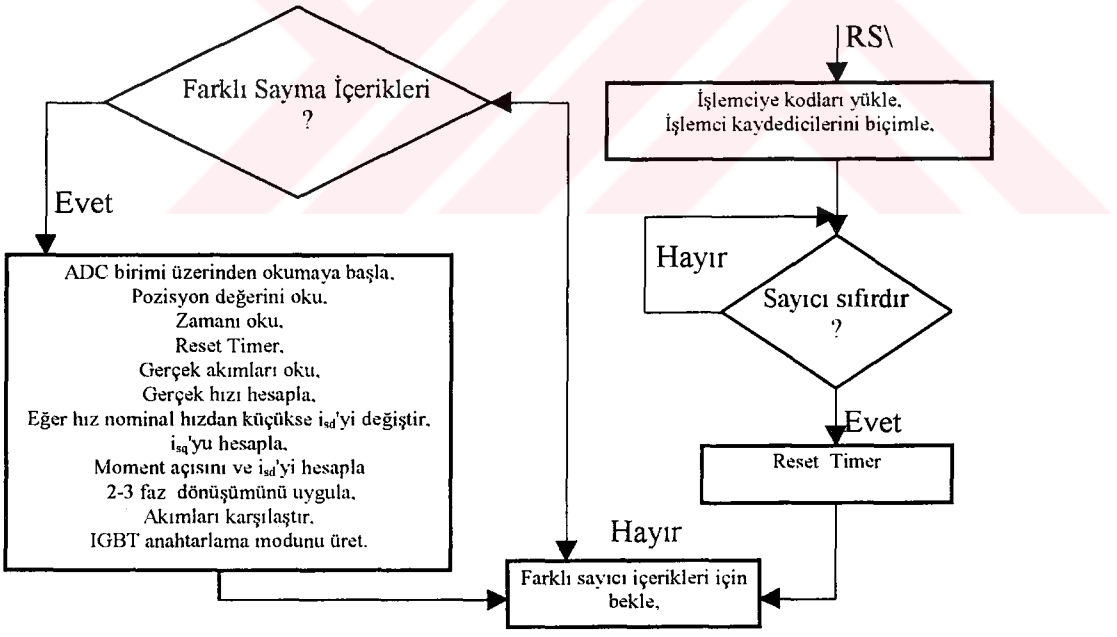
6.3 Sistemin Elemanlarının Tanıtılması

6.3.1 İnverter

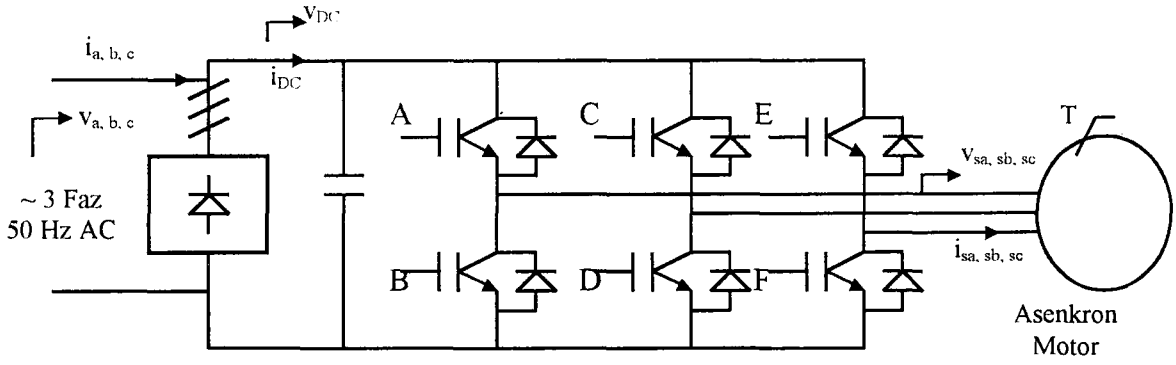
Yapılan çalışmada ABB firmasının 0.75 kW gücünde IGBT ana elemanlarıyla sürülen açık çevrim inverter modülü kullanılmıştır. Bu modülün prensip şeması şekil 6.3'te verildiği gibidir. Tam devre diyagramı ise Ek 7'de gösterilmektedir.



Şekil 6.1 Kesme kullanımasıyla ilgili akış diyagramı



Şekil 6.2 Altyordam kullanılmasıyla ilgili akış diyagramı



Şekil 6.3 Asenkron motorun gerilim beslemeli ac sürme devresi prensip şeması

6.3.2 Kullanılan motorlar

6.3.2.1 Asenkron motor

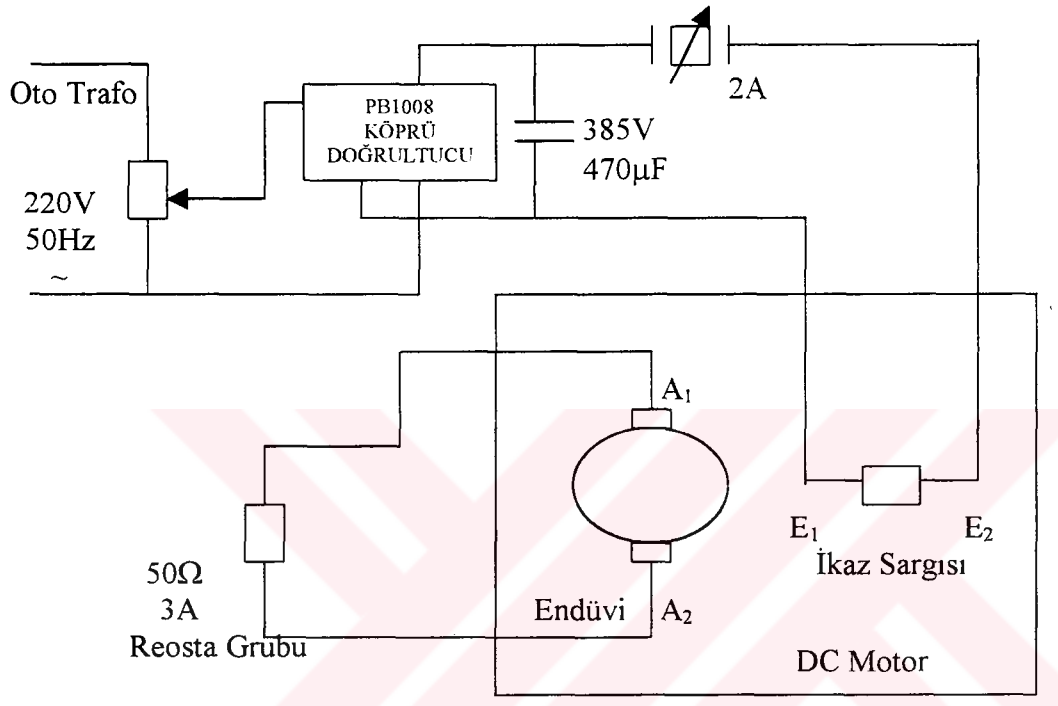
Kontrol edilen motor olarak, 0.55 kW, 2.6A, 220V, 50Hz, Cosϕ=0.79, 2p=4 kutuplu olan bir sincap kafesli asenkron motor kullanılmıştır. Çizelge 6.1'de motorun diğer parametrik değerleri verilmiştir.

Çizelge 6.1 Asenkron motorun parametrik değerleri

Simge	Nominal Değer	Birimi
R_s	0.17	Ω
R_r	0.133	Ω
L_s	31.4	mH
L_r	33.4	mH
L_m	3.18	mH
σL_s	31.09	mH
τ_r	0.251	mH/Ω
R_e	0.29	Ω
J	2.33	g-m^2
B	0.265	N-cm-s/rad
τ_e	0.0205	ms

6.3.2.2 DC motor

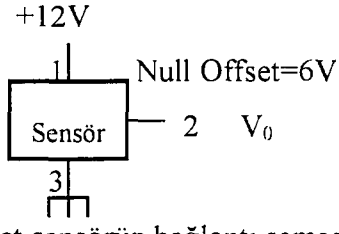
Asenkron motora yük bindirebilmek için etiket değerleri, 150-300V, 8.5-8.5A, 1-2 kW, 1500-3000 d/d, U_{Err}=220V, I_{Err}=0.6A ve dijital bir takometreye sahip olan özel yapım (labaratuvar tipi) DC motor kullanılmıştır. Bir oto trafo üzerinden doğrultulmuş gerilimle beslenerek DC jeneratör olarak çalıştırılan bu motorun bağlantı devre şeması şekil 6.4'te verilmiştir.



Şekil 6.4 DC motorun oto trafo üzerinden doğrultulmuş beslemeye sahip bağlantı şeması

6.3.3 Hall Effect transduser sensörler

Motor fazlarından akan ia ve ib faz akımlarını algılayabilmek için Escor firmasının Pro5 tipi 2 adet 5A'lik Hall-Effect sensörü kullanılmıştır. Sensörlerin bağlantı blok diyagramı şekil 6.5'te gösterildiği gibidir.



Şekil 6.5 Hall-Effect sensörün bağlantı şeması

6.3.4 Ara devreler

6.3.4.1 Ölçü amplifikatörleri

Sensörlerden ve takometreden gelen işaretlerin değişimleri gerilim cinsindendir. Fakat bu işaretlerin gerilim değerlerinin DSP tarafından izlenebilecek değerlere düşürülmesi amacıyla bahsedilen iki adet akım ve bir adet hız bilgisi için LM741 operasyonel amplifikatörleriyle, bu işaretlerden gelen gerilim değerleri ve bir kaynaktan üretilip potansiyometrik bir direnç üzerinden 6V'a indirilen gerilim değerinin karşılaştırılması amacıyla 3 adet ölçü amplifikatörü gerçekleştirilmiştir (Pastacı, 1997). Bunlara ait açık devre ve baskı devre şemaları Ek 8'de verildiği gibidir.

6.3.4.2 Bekletme devresi

DSP üzerinde bir adet ADC-DAC içeren arabirim elemanı bulunduğundan gelen bu işaretlerin bekletilip, sırasıyla gönderilmesi gerekmektedir. Gelen işaretin zaman olarak geciktirilmesi 4 adet buffer geçiş içeren DM7404 Hex Inverter ile sağlanıp, çıkışa 74AHC573 Latch'i (T.I. Advanced CMOS Data Book,1997) ve CD4016C üzerinden verilmektedir. Buna ait açık devre ve baskı devre şemaları ise Ek 9'da verilmiştir.

6.3.5 Analog Arabirim Devresi

Analog arabirim devresi analog ve dijital domenler arasında gerekli dönüşümü sağlar. Bu birim A/D, D/A, filtreleme v.b. bütün işlemleri kendi bünyesinde gerçekleştirir. 19.2 kHz maksimum örnekleme frekansına sahip ve genel amaçlı uygulamaların çoğunda kullanılabilen TLC320C40 analog arabirimi bu çalışmada da mevcut DSP kartı üzerinde yer almaktadır. Bünyesinde 14 bit ADC ve 14 bit DAC bulundurmaktadır. DSP kartı üzerindeki yerleşimi Ek 10'da verilen DSP devre şemalarında görülmektedir.
Bahsedilen fonksiyonlarının çoğu C50 DSP'sinin seri kapısı üzerinden yapılan bağlantıyla gerçeklenebilmektedir. İlgili yazılım Ek 5'de verilmiştir.

6.3.6 TMS320C5x DSP işlemcisi program yapısı genel özellikleri

Bu işlemci tam sayı çarpımı ve yüksek hız I/O komutları için optimize edilmiş hızından dolayı tercih edilmiştir. RISC komut seti 20 MHz saat (50 ns) hızında, tek çevrim opkod

çalışmasına izin verir. Bölü 5'de ayrıntılı bir şekilde açıklandığı gibi işlemcinin temel özellikleri şu şekilde özetlenebilir (T.I. C5x User's Guide,1997; T.I. C5x Starter Kit User's Guide, 1996):

1- Harvard mimarisinde çalışan 50 ns komut çevrimi,

2- T.I. DSP ailesinin birinci ve ikinci nesil sabit nokta ürünleriyle uyumlu çalışması,

3- RAM tabanlı çalışma,

4- 9K-kelimelik tek (single) çevrim çip üzeri program/veri RAM,

5- 2K-kelimelik tek çevrim çip üzeri işletim sistem ROM,

6- 1K-kelimelik çip üzeri çift (dual) kullanılabilir hafıza,

7- 224K-kelime x 16-bit maksimum adreslenebilir dış hafıza alanı (64K-kelimelik program, 64K-kelimelik veri, 64K-kelimelik hafıza-haritalanmış (memory-mapped) 16-bit I/O, 32K-kelimelik genel hafıza (global memory)),

8- 32 bit aritmatik-lojik ünitesi (ALU), 32 bit akümülatör (ACC) ve 32 bit akümülatör buffer (ACCB),

9- 16 bir paralel lojik ünitesi (PLU),

10- 16x16 bit paralel çarpma kapasitesi,

11- Tek (single) çevrim çarpma/biriktirme komutları, program ve veri hafıza alanları arasında veri transfer kapasitesi,

12- Sekiz adet yardımcı kaydedici (register), kesme boyunca kaydedicileri depolamak için onbir adet program bağlam anahtarlama kaydedicileri,

13- DSP'nin başka bir seri aletle direkt haberleşmesi için tam-ikili (full-duplex) senkron seri kapı,

14- Zaman bölme çarpma (division multiple access) uygulaması için seri kapı (TDM),

15- Dört adet maskelenebilir kesme (maskable interrupt), bir adeti timer kesmesi, diğerleri ise seri kapı kesmeleri için olan beş adet iç (internal) kesme, bir adet maskelenemeyen (non-maskable) kesme,

16- Program, veri ve I/O hafıza alanları ve RS\ kesme sinyali için onaltı adet yazılım programlanabilir durum bekleme jeneratörleri (wait state generators),

17- Gecikmiş dallanma, çağırma ve geri dönüş (branch, call ve return) komutları için dört adet derin hat çalışma (deep pipeline),

18- Saat opsiyonunu bir'le bölen çip üzeri saat jeneratör,

19- Düşük güçte mod operasyonu,

20- Aynı noktada kesişen (concurrent) dış bölme çarpma uygulaması için genişletilmiş tutma operasyonu,

21- 16 bit program sayıcı (PC),

22- 132 bacaklı dörtgen düz çip paketi.

Mevcut program hafızasını desteklemek üzere iki adet 256 kbit EPROM ve hız kontrolünün yanında pozisyon kontrolü de yapılacaksa bir adet enkoder kullanılmalıdır.

Yapılan çalışmada kullanılan eleman ve aletlerin özellikleri yukarıda tanımlandığı gibidir. Bu elemanlar kullanılarak gerçekleştirilen sistemin tam kontrol blok diyagramı ise şekil 6.6'da verilmiştir.

6.4 Yazılım Tanımlamaları

6.4.1 Operasyona hazır işlemi (Initialization)

Sisteme güç verilerek ve PCB'deki anahtara basılmak suretiyle birkaç adet saat (clock) çevrimi için bir lojik low RS\ sinyali uygulanır ve asıl program INIT denilen bir initialization işlemi yürütür. Bu işlem EPROM'lardan gelen program kodlarını okur ve işlemci ve ilgili dış program, I/O sinyal alanlarını biçimler. Mikroişlemciyi dış çevre birimlerle senkronize etmek için okuma komutları arasında bekleme durumları (wait states) kullanılır. IOWSR ve PDSWR adlı iki kaydedici bu işlem için saklanır. PDWSR dış program alan uygulamaları için kullanılırken, IOWSR işlemci ve I/O kapıları arasındaki bekleme durumlarını kontrol eder. Mikroişlemcinin bekleme durumlarının sayısı CWSR ile belirlenir. Hem ADC'nin hem de enkoderin okuma giriş zamanları işlemcininkinden daha yavaş olduğundan, yeterli sayıda bekleme durumu set edilmelidir. Bu bölümlerin her biri için bir kesme (interrupt) kullanımı bir de altyordam (subroutine) kullanımı vardır.

Kesme kullanımında, initialization işlemi kesme maskeleme kaydedicisi, IMR'deki karşılık gelen bit'i maskeleyerek giriş pozisyon işlemi için INT3 kesmesini enable eder, genel kesme maskeleme bit'i, INTM'yi temizler (clear işlemi), giriş pozisyon işlemi, INITPOS'a dallanmak için enkoderin CHI sinyalini tersleyerek (invert işlemi) elde edilen bir aktif low INT3\ alıncaya kadar, mikroişlemciyi IDLE durumuna sürer.

Altyordam kullanımında, giriş pozisyon işleminde direkt bir çağırma (call) kullanılabilir.

6.4.2 Giriş pozisyon işlemi

Kesme kullanımında, zaman (k-1)'i belirleyen K_1 olduğu yerde, ANGRDK_1 etiketli hafıza yerleşimine bir sıfır değeri yüklenir. Zamanlayıcı (timer), TIM durdurulur ve PRD'ye depolanan değerle yüklenir. Kesme işlemi aynı zamanda INT1\ kesmelerini sağlar ve genel kesme maskeleme bit'inin içeriğini temizler.

Altyordam kullanımında, sayıcı içeriklerini bu işlem bir sıfır değeri okununcaya kadar okumaya devem eder. Bundan sonra, bu işlem PRD'nin içeriğiyle timer'ı yükleyerek takip eden operasyona hazır hale getirir.

6.4.3 Ardışık veri girişleri

Kesme durumunda, mevcut INITPOS işleminden sonra bir INT1\ sinyali alınıncaya kadar bir idle modda işlemciyi desteklemek gereklidir. Altyordam kullanımında program direkt olarak bu işleme dallanır. Sonra, program,sayıcının düşük ve yüksek ağırlıklı byte'ların da 6 ve 4 nolu kapıları kullnarak ilk ve ardışık pozisyon değerini okumak için NEWDATA denilen kesme kullanımında INT1\ işlemine dallanır.

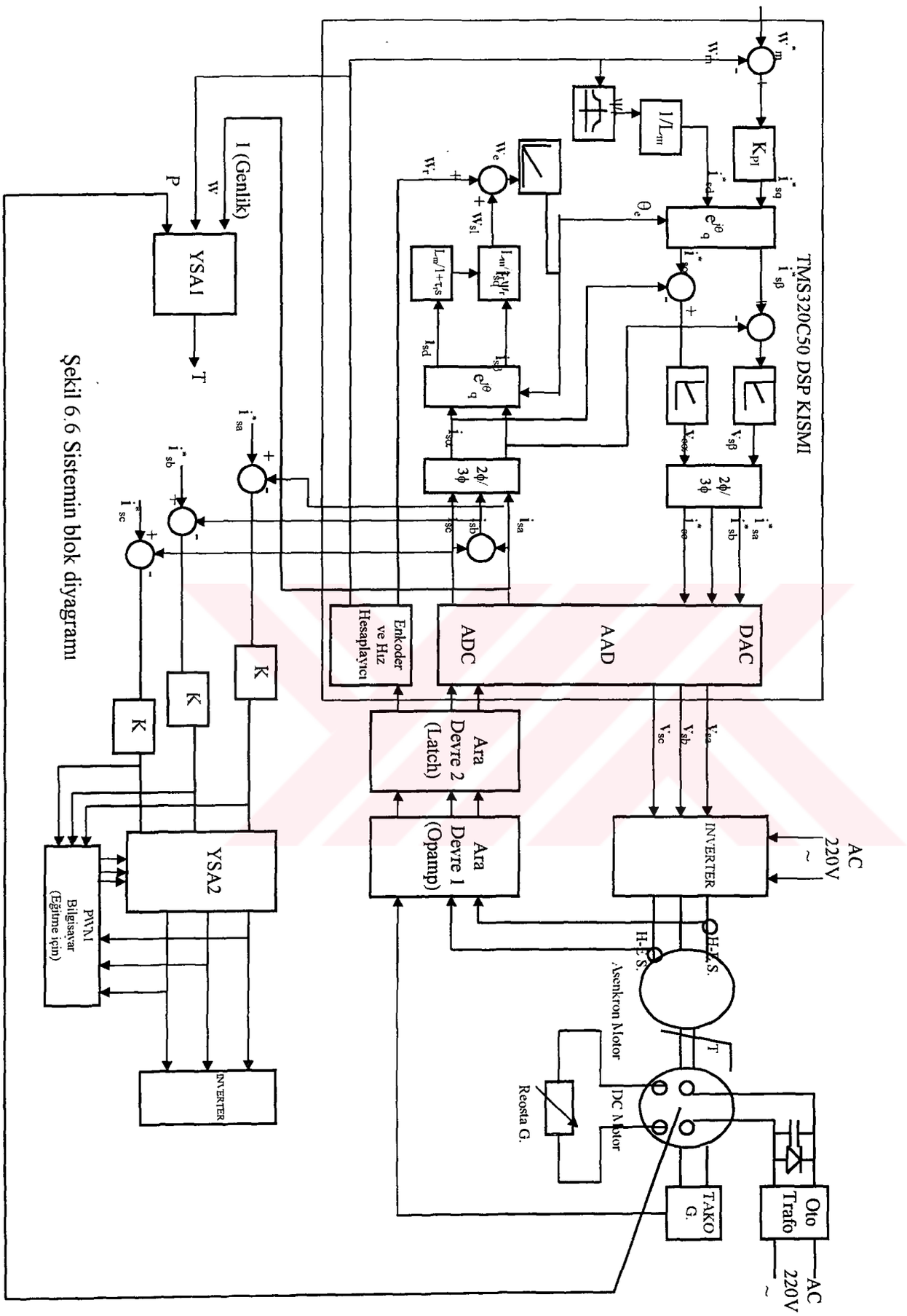
Altyordam kullanımında, farklı bir sayıcı değeri alınıncaya kadar, sayıcı PA4 ve PA6 kapılarını kullanarak okumaya devem eder.

Her iki durumda, işlem, faz akımları i_{sa} ve i_{sb} 'yi 0 ve 1 kapılarından okur ve bunları hafiza da sırasıyla IAACT ve ICACT'ye yerleştirir. Daha sonra, timer kaydedici durdurulur ve içeriği TIME denilen bir veri hafıza yerleşimine yüklenir ve buradan PRD kaydedicisinde depolanan değerle yüklenir. İki ardışık veri alınması arasındaki zaman farkı ölçülür. Sayıcının düşük ve yüksek ağırlıklı byte'ları ANGRDK denilen bir veri hafiza yerleşimine depolanan açısal pozisyon değerini oluşturmak için birleştirilir. ADC'den alınan değerler aynı zamanda gerçek değerlerine dönüştürülür. INT1\ işleminin sonunda bütün kesmeler kontrol algoritma program çevrimi tamamlanıncaya kadar servis dışı edilir (disable). Bu çalışmada program yapısında enkoder pozisyon takip kısmı verilmekte olup, ölçümlerde ise sadece hız sayısal değerlerinin alınması gerçeklenmiştir yani enkoder kullanılmamıştır. YSA eğitmeleri için hızın sayısal değerleri kontrol sisteminin blok diyagramından da görüldüğü (şekil 6.6) gibi yeterli olmaktadır (YSA1). Yine, YSA2 de vektör bölgesindeki sekiz mümkün dağılım bölgesine göre eğitmeler yapılıp buradan IGBT kapılarına anahtarlama şekli olarak doğru çıkışlar gönderilmektedir (off-line). Kontrol programıyla ilgili yazılım Ek 6'da verilmiştir.

6.5 Sonuçlar

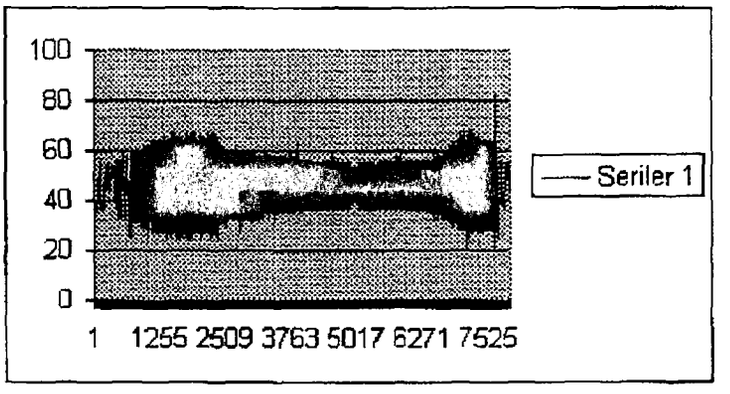
6.5.1 Yapılan çalışmaya ait olan ölçüm sonuçları

ADC üzerinden ölçümü yapılan i_{sa} ve i_{sb} faz akımlarının motorun boşta çalışmasında alınan eğrileri şekil 6.7'de, motor yükte çalışıyorken i_{sa} akımının eğrisi şekil 6.8'de, hız eğrisi şekil

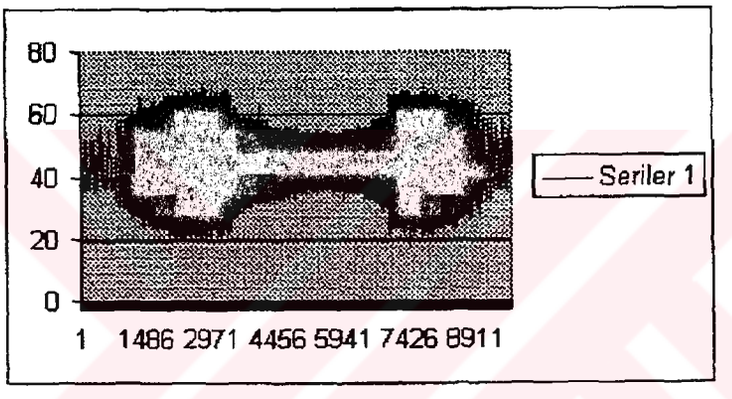


Şekil 6.6 Sistemin blok diyagramı

6.9'da, referans akım ile faz akımının aynı eksenli eğrileri şekil 6.10'da, referans hız eğrisi şekil 6.11'de, i_{sq} moment kontrolünü sağlayan akımın eğrisi şekil 6.12'de verilmiştir.

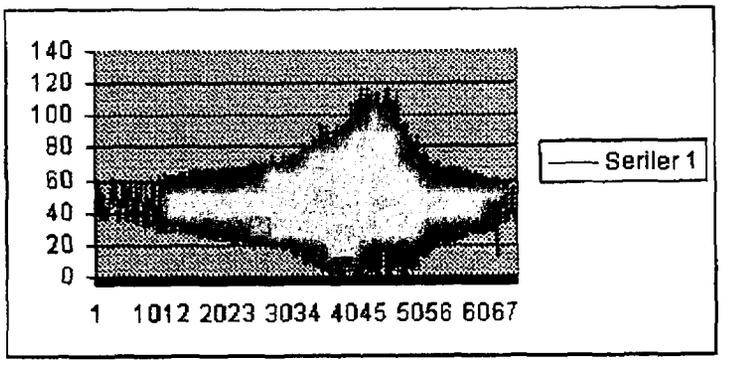


(a)

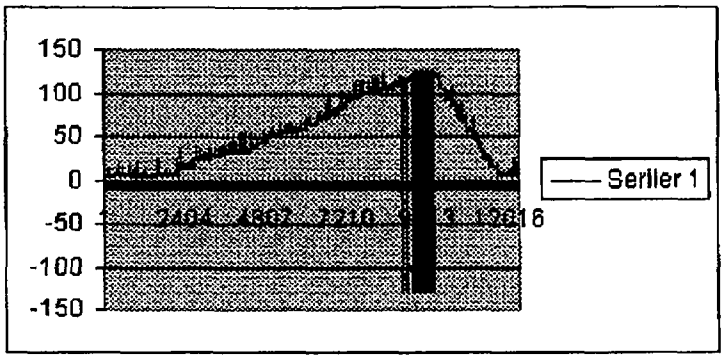


(b)

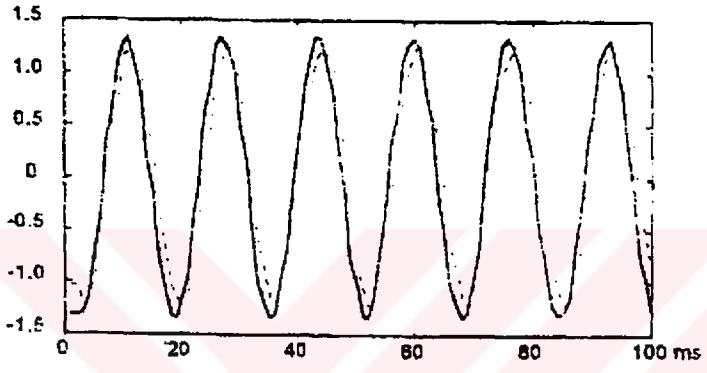
Şekil 6.7 Boşta çalışmada (a) i_{sa} akımının (b) i_{sb} akımının eğrileri



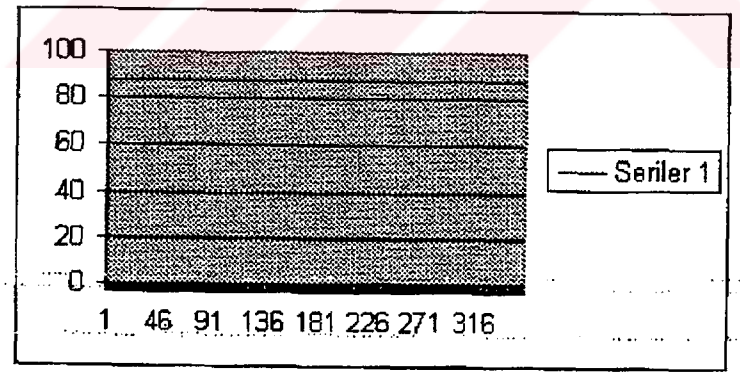
Şekil 6.8 Yükte çalışma da i_{sa} akımının eğrisi



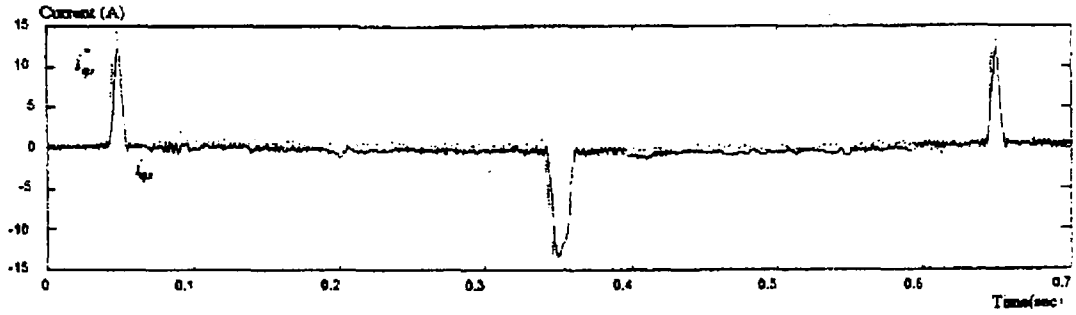
Şekil 6.9 Rotor hızının eğrisi



Şekil 6.10 Referans akım ile faz akımını aynı eksenli eğrileri



Şekil 6.11 Referans rotor hız eğrisi



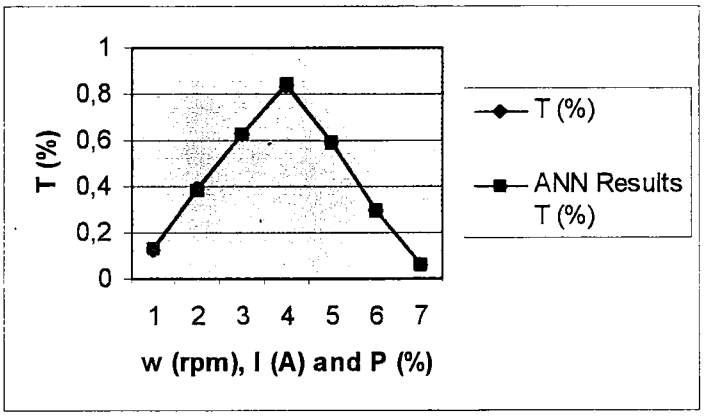
Şekil 6.12 i_{sq} akımının eğrisi

Elde edilen ölçüm sonuçları çizelge 6.2'de verilmiştir. Klasik hatanın geriyeyayılımı algoritmasının test sonuçları diyagramı şekil 6.13'de, hızlı hatanın geriyeyayılımı algoritmasının test sonuçları diyagramı şekil 6.14'te, her ikisine de uygulanan YSA mimarisinin blok diyagramı şekil 6.15'te, ağ mimarisi ise şekil 6.16'da gösterilmektedir. Yine aynı ölçümler için tasarlanan yüksek mertebeden ağ yapısı için blok diyagram şekil 6.17'de, ağ mimarisi şekil 6.18'de ve test sonuçlarının diyagramı klasik hatanın geriyeyayılımı algoritması için şekil 6.19'da, hızlı hatanın geriyeyayılımı algoritması için şekil 6.20'de verilmiştir. Çizelge 6.3 klasik hatanın geriyeyayılımı algoritması için, çizelge 6.4 hızlı hatanın geriyeyayılımı algoritması için, yine yüksek mertebeden ağ yapısında çizelge 6.5 klasik hatanın geriyeyayılımı algoritması için, çizelge 6.6 hızlı hatanın geriyeyayılımı algoritması için test fazı sonuçlarını göstermektedir.

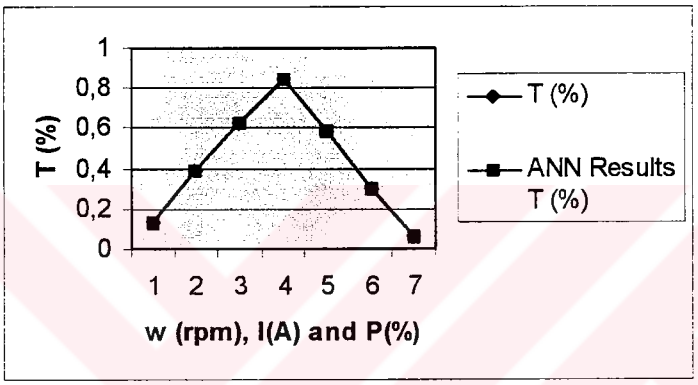
Çizelge 6.2 Ölçüm sonuçları

w (d/d)	I (A)	P (%)	T (%)
0	0	0	0
10.4719	2.506	0.000571	0.025
16.7551	1.850	0.00321	0.05
29.3215	2.029	0.00776	0.0875
54.4542	1.850	0.0289	0.1625
60.7374	1.372	0.0365	0.175
64.9262	1.491	0.0406	0.1875
71.2094	1.551	0.0462	0.2125
85.8701	1.432	0.0714	0.25
94.2477	1.312	0.0874	0.275
113.0977	0.954	0.1305	0.3375
121.4749	0.835	0.1495	0.3625

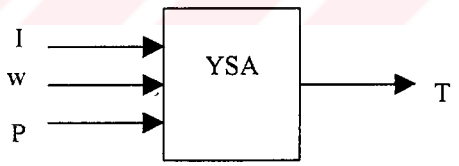
142.4188	0.775	0.2213	0.4375
161.2684	0.656	0.2638	0.475
178.0235	0.651	0.327	0.5375
184.3067	0.596	0.346	0.55
194.7787	0.537	0.389	0.5875
222.0058	0.412	0.4967	0.6625
232.4778	0.298	0.5594	0.7
245.0442	0.292	0.6428	0.75
253.4218	0.288	0.655	0.7625
265.9881	0.285	0.7347	0.8
272.2713	0.284	0.7526	0.8125
289.0265	0.281	0.8542	0.875
301.5928	0.280	0.9334	0.9
286.9321	0.282	0.8653	0.9
259.7049	0.286	0.6843	0.775
247.1386	0.292	0.6217	0.7375
232.4778	0.299	0.5501	0.6875
217.8170	0.414	0.4821	0.65
209.4395	0.416	0.4415	0.625
182.2123	0.598	0.339	0.55
161.2684	0.656	0.2778	0.4875
154.9852	0.661	0.2428	0.4625
136.1356	0.782	0.1836	0.4
123.5693	0.833	0.1542	0.3625
83.7758	1.193	0.0698	0.25
69.1150	1.553	0.0469	0.2
56.5486	1.848	0.0321	0.175
43.9822	1.860	0.018	0.125
14.6607	2.446	0.00241	0.05



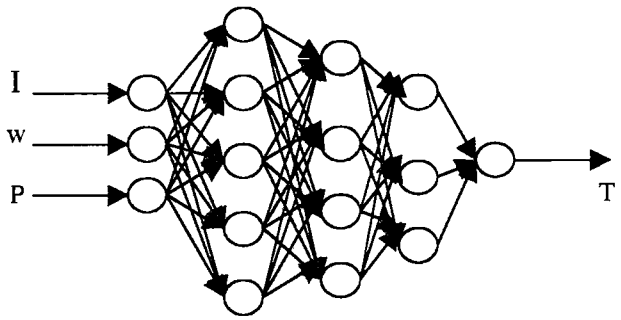
Şekil 6.13 Klasik hatanın geriyeyayılımı algoritmasının test sonuçları diyagramı



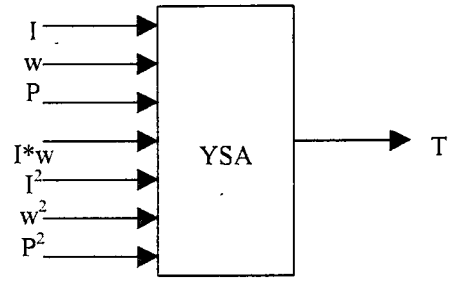
Şekil 6.14 Hızlı hatanın geriyeyayılımı algoritmasının test sonuçları diyagramı



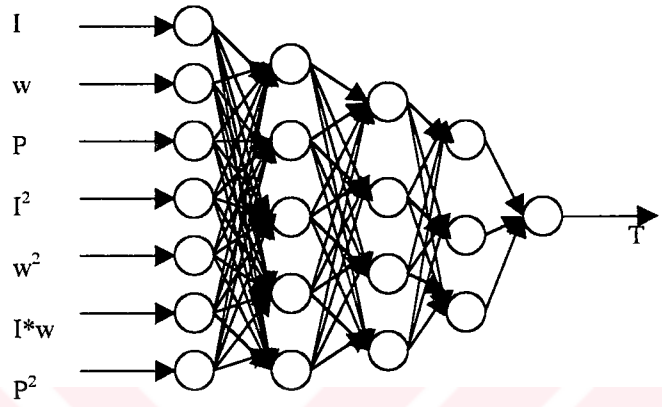
Şekil 6.15 YSA mimarisinin blok diyagramı



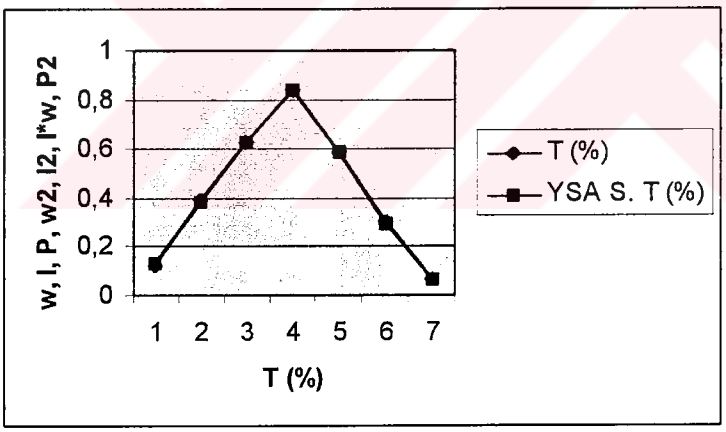
Şekil 6.16 YSA ağ mimarisi



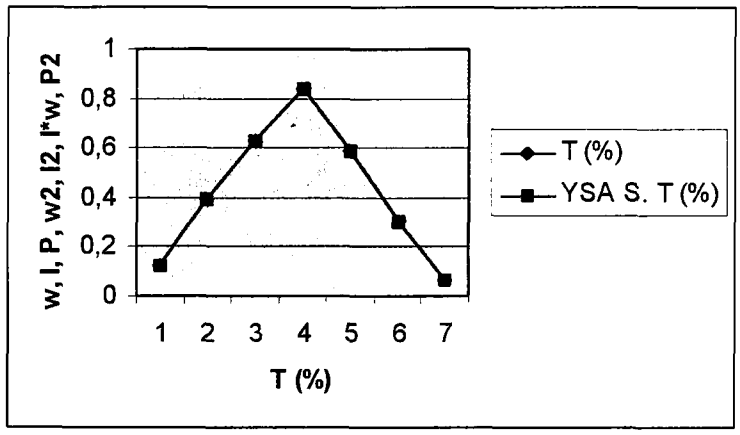
Şekil 6.17 YSA'da yüksek mertebeden ağ yapısı için blok diyagram



Şekil 6.18 YSA'da yüksek mertebeden ağ yapısı ağ mimarisi



Şekil 6.19 Yüksek mertebeden ağ yapısında klasik hatanın geriyeyayılımı algoritmasının test sonuçları diyagramı



Şekil 6.20 Yüksek mertebeden ağ yapısında hızlı hatanın geriyeyayılımı algoritmasının test sonuçları diyagramı

Çizelge 6.3 Klasik hatanın geriyeyayılımı algoritması için test fazı sonuçları

w (d/d)	I (A)	P (%)	T (%)	YSA Sonuçları T (%)
43.9822	1.860	0.02053	0.125	0.132078
127.7581	0.831	0.165	0.3875	0.379243
207.3451	0.417	0.4456	0.625	0.627734
278.5545	0.283	0.792	0.8375	0.841526
194.7787	0.537	0.3895	0.5875	0.587541
100.5309	1.306	0.00946	0.3	0.292568
20.9439	1.847	0.00412	0.0625	0.060119

Çizelge 6.4 Hızlı hatanın geriyeyayılımı algoritması için test fazı sonuçları

w (d/d)	I (A)	P (%)	T (%)	YSA Sonuçları T (%)
43.9822	1.860	0.02053	0.125	0.127778
127.7581	0.831	0.165	0.3875	0.385945
207.3451	0.417	0.4456	0.625	0.626175
278.5545	0.283	0.792	0.8375	0.838156
194.7787	0.537	0.3895	0.5875	0.587261
100.5309	1.306	0.00946	0.3	0.299588
20.9439	1.847	0.00412	0.0625	0.061510

Çizelge 6.5 Yüksek mertebeden ağ yapısında klasik hatanın geriyeyayılımı algoritması için test fazı sonuçları

w (d/d)	I (A)	P (%)	w^2	I^2	w*I	P^2	T (%)	YSA S. T (%)
43.9822	1.860	0.02053	1934.43	3.459	6692.3	0.00042	0.125	0.13017
127.758	0.831	0.165	16322.1	0.690	11270.4	0.02722	0.3875	0.38124
207.345	0.417	0.4456	42991.9	0.173	7472.0	0.19855	0.625	0.62553
278.554	0.283	0.792	77592.6	0.080	6213.6	0.62726	0.8375	0.83926
194.778	0.537	0.3895	37938.7	0.288	10937.7	0.15171	0.5875	0.58744
100.530	1.306	0.00946	10106.4	1.705	17237.5	0.00894	0.3	0.29467
20.9439	1.847	0.00412	438.64	3.411	1496.3	0.00001	0.0625	0.06225

Çizelge 6.6 Yüksek mertebeden ağ yapısında hızlı hatanın geriyeyayılımı algoritması için test fazı sonuçları

w (d/d)	I (A)	P (%)	w^2	I^2	w*I	P^2	T (%)	YSA S. T (%)
43.9822	1.860	0.02053	1934.43	3.459	6692.3	0.00042	0.125	0.12442
127.758	0.831	0.165	16322.1	0.690	11270.4	0.02722	0.3875	0.38885
207.345	0.417	0.4456	42991.9	0.173	7472.0	0.19855	0.625	0.62401
278.554	0.283	0.792	77592.6	0.080	6213.6	0.62726	0.8375	0.83688
194.778	0.537	0.3895	37938.7	0.288	10937.7	0.15171	0.5875	0.58748
100.530	1.306	0.00946	10106.4	1.705	17237.5	0.00894	0.3	0.30165
20.9439	1.847	0.00412	438.64	3.411	1496.3	0.00001	0.0625	0.06311

6.5.2 ABB firmasında yapılan çalışmanın ölçüm sonuçları ve karşılaştırmalı analiz

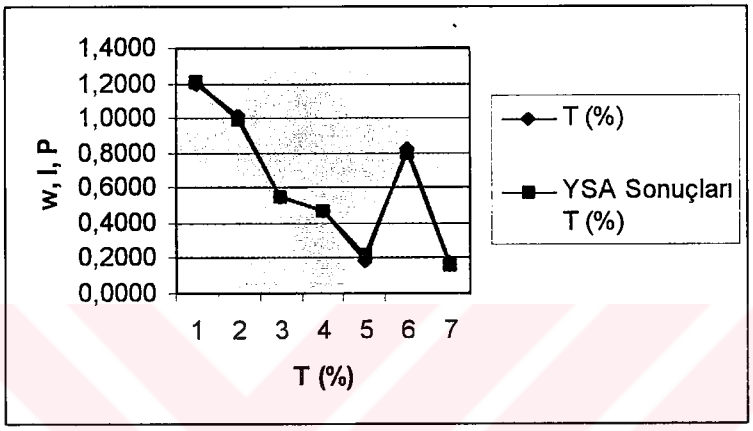
Çizelge 6.7 ABB firmasından alınan ölçüm sonuçlarını göstermektedir. Şekil 6.21 test sonuçlarının klasik hatanın geriyeyayılımı algoritması için diyagramını, şekil 6.22 ise hızlı hatanın geriyeyayılımı algoritması için diyagramını vermektedir. Tasarlanan YSA mimarileri bu ölçümler için de aynen kullanılmıştır. Çizelge 6.8'de klasik hatanın geriyeyayılımı algoritması için, çizelge 6.9'da ise hızlı hatanın geriyeyayılımı algoritması için test fazı sonuçları sunulmaktadır. Yine, yüksek mertebeden ağ yapısında ise çizelge 6.10 klasik hatanın geriyeyayılımı algoritması için, çizelge 6.11 hızlı hatanın geriyeyayılımı algoritması için test fazı sonuçlarını göstermektedir. Çizelge 6.12'de ise, hem her iki ölçüm sonuçları hem

de her iki algoritma için sistem hatası, eğitme ve test fazı sonuçları ve de iterasyon sayıları bakımından karşılaştırmalı bir analiz şekil sunulmaktadır. YSA algoritmasının akış diyagramı ise şekil 6.23'te verildiği gibidir. Bu ölçümlerde kullanılan asenkron motorun özellikleri ise, 18.5 kW, 36.5A, 380V, 50Hz, Cosϕ=0.86, 2p=4 kutuplu olan bir sincap kafesli asenkron motor kullanılmıştır.

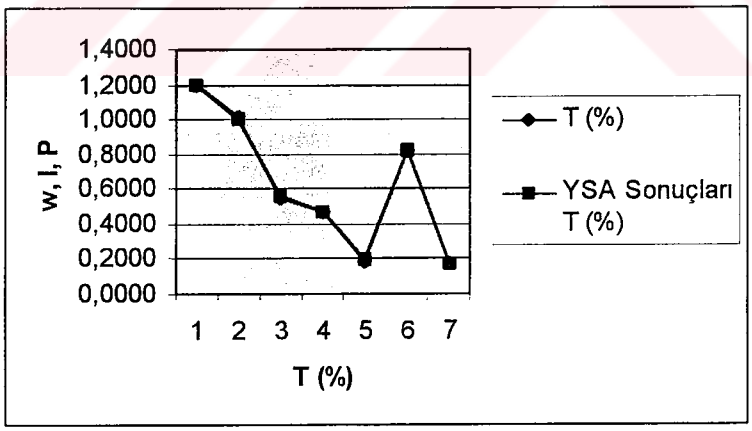
Çizelge 6.7 ABB firmasından alınan ölçüm sonuçları

w (d/d)	I (A)	P (%)	T (%)
0	0	0	0
2.0944	13.44	0.26	1.35
4.1888	13.45	0.27	1.37
8.3776	13.58	0.27	1.56
10.4720	13.32	0.27	1.24
12.5664	13.97	0.27	1.85
16.7552	13.50	0.28	1.48
18.8496	13.80	0.28	1.72
20.9440	13.55	0.28	1.53
23.0383	13.22	0.28	1.15
25.1327	12.23	0.28	0.31
41.8879	12.48	0.29	0.49
52.3599	12.20	0.29	0.28
62.8319	12.44	0.30	0.45
83.7758	12.72	0.36	0.64
94.2478	12.90	0.40	0.82
104.7198	12.48	0.42	0.49
115.1917	12.50	0.44	0.50
125.6637	12.46	0.44	0.47
136.1357	12.39	0.44	0.40
146.6077	12.01	0.44	0.16
167.5516	12.34	0.50	0.38
178.0236	12.56	0.52	0.53
188.4956	12.40	0.56	0.43
198.9675	12.42	0.58	0.45
209.4395	12.22	0.60	0.30

230.3835	12.10	0.68	0.18
240.8554	12.06	0.69	0.13
251.3274	12.18	0.71	0.28
261.7994	12.05	0.75	0.13
272.2714	12.11	0.76	0.18
293.2153	12.03	0.90	0.13
303.6873	12.10	0.93	0.18



Şekil 6.21 Klasik hatanın geriyeyayılımı algoritması için test sonuçlarının diyagramı



Şekil 6.22 Hızlı hatanın geriyeyayılımı algoritması için test sonuçlarının diyagramı

Çizelge 6.8 Klasik hatanın geriyeyayılımı algoritması için test fazı sonuçları

w (d/d)	I (A)	P (%)	T (%)	YSA Sonuçları T (%)
6.2832	13.27	0.27	1.2	1.20113
14.6608	13.08	0.27	1.01	0.98875
31.4159	12.57	0.28	0.55	0.54890
73.3038	12.45	0.35	0.47	0.47161
157.0796	12.09	0.47	0.18	0.21294
219.9115	12.90	0.61	0.82	0.79916
282.7433	12.07	0.85	0.17	0.16421

Çizelge 6.9 Hızlı hatanın geriyeyayılımı algoritması için test fazı sonuçları

w (d/d)	I (A)	P (%)	T (%)	YSA Sonuçları T (%)
6.2832	13.27	0.27	1.2	1.20015
14.6608	13.08	0.27	1.01	1.00054
31.4159	12.57	0.28	0.55	0.55951
73.3038	12.45	0.35	0.47	0.47007
157.0796	12.09	0.47	0.18	0.19074
219.9115	12.90	0.61	0.82	0.81940
282.7433	12.07	0.85	0.17	0.16987

Çizelge 6.10 Yüksek mertebeden ağ yapısı için klasik hatanın geriyeyayılımı algoritması test fazı sonuçları

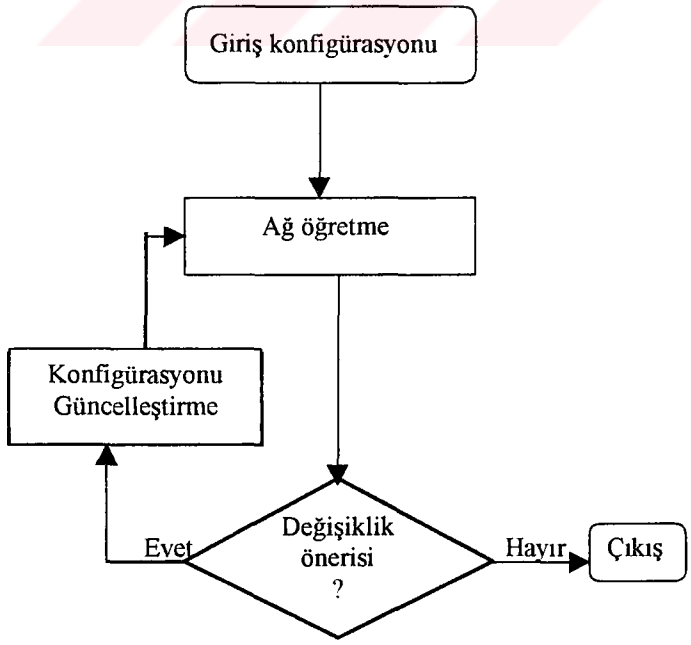
w (d/d)	I (A)	P (%)	w^2	I^2	w*I	P^2	T (%)	YSA S. T (%)
6.2832	13.27	0.27	39.478	176.09	83.378	0.0729	1.2	1.20109
14.6608	13.08	0.27	214.939	171.08	191.763	0.0729	1.01	0.99798
31.4159	12.57	0.28	986.958	158.0	394.897	0.0784	0.55	0.54950
73.3038	12.45	0.35	5373.44	155.0	912.632	0.1225	0.47	0.47070
157.079	12.09	0.47	24674.0	146.16	1899.09	0.2209	0.18	0.19514
219.911	12.90	0.61	48361.0	166.41	2836.85	0.3721	0.82	0.81425
282.743	12.07	0.85	79943.7	145.68	3412.71	0.7225	0.17	0.16938

Çizelge 6.11 Yüksek mertebeden ağ yapısı için hızlı hatanın geriyeyayılımı algoritması test fazı sonuçları

w (d/d)	I (A)	P (%)	w^2	I^2	w*I	P^2	T (%)	YSA S. T (%)
6.2832	13.27	0.27	39.478	176.09	83.378	0.0729	1.2	1.20076
14.6608	13.08	0.27	214.939	171.08	191.763	0.0729	1.01	1.10033
31.4159	12.57	0.28	986.958	158.0	394.897	0.0784	0.55	0.55967
73.3038	12.45	0.35	5373.44	155.0	912.632	0.1225	0.47	0.47001
157.079	12.09	0.47	24674.0	146.16	1899.09	0.2209	0.18	0.19072
219.911	12.90	0.61	48361.0	166.41	2836.85	0.3721	0.82	0.81971
282.743	12.07	0.85	79943.7	145.68	3412.71	0.7225	0.17	0.16979

Çizelge 6.12 Her iki ölçüm sonuçları için karşılaştırmalı analiz

	İterayon Sayısı	Klasik Ha. Ge. Al. Sistem Hatası	Hızlı Ha. Ge. Al. Sistem Hatası	Yük. Mer. Klasik Ha. Ge. Al. Sistem Hatası	Yük. Mer. Hızlı Ha. Ge. Al. Sistem Hatası
Yapılan Çalışma	300000	%0.0011	%0.0007	%0.0009	%0.0004
ABB'den Alınan Ölçümler	300000	%0.0012	%0.0003	%0.0010	%0.0003



Şekil 6.23 YSA algoritması akış diyagramı

6.5.3 Sonuçların yorumlanması

Yapılan bu çalışmada DSP uygulamaları ve YSA değerlerinin eğitme işlemleri bir 200-MMX işlemcili bilgisayar üzerinden yapılmış olup, 300000 iterasyon sonucunda klasik Hatanın Geriyeyayılımı Algoritması için %0.0011, Hızlı Hatanın Geriyeyayılımı Algoritması için %0.0007 ve yüksek mertebeden YSA'da, klasik Hatanın Geriyeyayılımı Algoritması için %0.0009 ve Hızlı Hatanın Geriyeyayılımı Algoritması için %0.0004 gibi çok az bir sistem hatasıyla motorun istenilen momente istenilen sürede (nano saniyeler mertebesinde) ulaştığı görülmektedir. ABB firmasının Kartal Fabrikası İnverter Teknik Servisi'nde de gelişmiş bir inverter modülü olan ACS600 (U_1 gerilimi 380...415V, U_2 gerilimi 0-380...415V, I_1N/I_{1hd}-60/44A, I_2N/I_{2hd}-62/47A, f_1-48...63Hz, f_2-0-300Hz) üzerinden benzer ölçümler yapılmış olup, YSA'ya uygulanan değişkenler aynı kalmak üzere, klasik Hatanın Geriyeyayılımı Algoritması için %0.0012 ve Hızlı Hatanın Geriyeyayılımı Algoritması için %0.0003 ve yüksek mertebeden YSA'da, klasik Hatanın Geriyeyayılımı Algoritması için %0.0010 ve Hızlı Hatanın Geriyeyayılımı Algoritması için %0.0003 olarak benzer şekilde çok az bir sistem hatasıyla motorun istenilen momente istenilen sürede (nano saniyeler mertebesinde) ulaştığı görülmüştür.

Yine, elde edilen sonuçların karşılaştırmalı bir analizinin verilmesi amacıyla ABB firmasından alınan ölçüm değerleri de, yapılan çalışmadaki üstünlüğü belirlemesi ve de bundan sonra yapılacak endüstriyel gelişmelere ışık tutması açısından önemli bir gelişmedir.

Bundan sonraki çalışmalarda YSA'dan elde edilen sonuçların ağırlık değerlerinin güncelleştirilerek sisteme tekrar verilmesi, PI kontrolörün yanı sıra PID kontrolörlü ve genetik algoritmayı içeren yapıların gerçeklenmesi bu tür kontrol sistemlerine yapılacak yeni iterasyonel yaklaşımlar olacaktır.

KAYNAKLAR

Ahmed, İ. (1991), Digital Control Applications with the TMS320 Family, Custom Printing Company, Owensville, Missouri.

Attaianese, C., Tomasso, G., Damiano, A., Morongio, I. ve Perfetto, A. (1997), " On Line Estimation of Speed and Parameters in Induction Motor Drives", ISIE'97, Guimaraes, Portugal, 1054-1059.

Blaschke, F. ve Bayer, K.H. (1977), "Stability Problems with the Control of Induction Machines Using the Method of Field Orientation", IFAC Control in Power Electronic and Electronic Drives, 483-492.

Boldea, I. ve Nasar, S.A. (1992), Vector Control of AC Drives, CRC Press.

Bose, B.K. (1986), Power Electronics and AC Drives, Prentice-Hall, New Jersey.

Bose, B.K. (1993), "Power Electronics and Motion Control-Technology Status and Recent Trends", IEEE Transactions on Industrial Applications, IA-29, 902-909.

Bose, B.K. (1994), " Expert System, Fuzzy Logic and Neural Network Applications in Power Electronics and Motion Control", Proceedings of the IEEE, Vol.82, No.8, 1303-1323.

Chassaing, R. (1992), Digital Signal Processing with C and the TMS320C30, A Wiley-Interscience Publication, New York, Bölüm 8.

Chow, M.Y. ve Yee, S.O. (1991), " Methodology for On-line Incipient Fault Detection in Single Phase Squirrel-Cage Induction Motors Using Artificial Neural Networks", IEEE Transactions on Energy Conversion, Vol.6, No.3, 536-545.

Chow, M.Y., Sharpe, R.N. ve Hung, J.C. (1993), " On the Application and Design of Artificial Neural Networks for Motor Fault Detection- Part I-II", IEEE Transactions on Industrial Applications, Vol.40, No.2, 181-196.

Çetin, İ. ve Schuisky, W. (1987), Elektrik Motörleri, Fatih Yayınevi, İstanbul, Bölüm 1

Çiprut, İ. (1994), DSP Tabanlı Bir Sistem ile Anahtarlamalı Relüktans Motorunun Kontrolu, Yüksek Lisans Tezi, İ.T.Ü. Fen Bilimleri Enstitüsü.

Di Gabriele, R., Parasiliti, F. ve Tursini, T. (1997), " Digital Field Oriented Control for Induction Motors: Implementation and Experimental Results", Universities Power Engineering Conference (UPEC'97).

Dote, Y. (1988), " Application of Modern Control Techniques to Motor Control", Proceedings of the IEEE, Vol.76, No.4, 438-454.

Dote, Y. (1990), Servo Motor and Motion Control Using Digital Signal Processors, Texas Instruments-Prentice Hall, Englewood Cliffs, New Jersey, Bölüm 1, 2, 3 ve 4.

Durak, C.T. (1995), Vektör Kontrol Yöntemiyle Denetime Yönelik Bir Asenkron Modeli, Yüksek Lisans Tezi, İTÜ Fen Bilimleri Enstitüsü.

Freeman, J.A. ve Skapura, D.M. (1992), Neural Networks Algorithms, Applications and Programming Techniques, Addison Wesley Publlishing Company, New York.

Garcia, G.O., Stephan, R.M. ve Watanabe, E.H. (1994), "Comparing the Indirect Field-Oriented Control with a Scalar Method", IEEE Transaction on Industrial Electronics, Vol.41, No.2.

Gülez, K., Karlık, B. ve Koçyiğit, Y. " Motor Arızalarının Erken Tanısı için Yapay Sinir Ağlarının Tasarımı", 2. Endüstriyel Otomasyon'95 Sempozyumu, 30-31 Mart 1995, İstanbul, 286-290.

Haykin, S. (1994), Neural Networks, Macmillan Publishing Company, New Jersey.

Karayiannis, N.B. ve Venetsanopoulas A.N. (1991), "Fast Learning Algorithms for Neural Networks", Elsevier Publishers, North Holland, 1141-1144.

Karayiannis, N.B. ve Venetsanopoulas A.N. (1992), "Fast Learning Algorithms for Neural Networks", IEEE Transaction Circuits and Systems-II Analog and Digital Systems Proceedings, Vol.39, No.7, 453-474.

Karayiannis, N.B. ve Venetsanopoulas A.N. (1994), Artificial Neural Networks- Learning Algorithms, Performance Evaluation and Applications, Kluwer Academic Publishers, Bölüm 4, 161-195.

Krishnan, R. ve Bharadwaj, A.S. (1991), "A Review of Parameter Sensitivity and Adaptation in Indirect Vector Controlled Induction Motor Drive Systems", IEEE Transaction on Power Electronics, Vol.6, No.4, 695-702.

Kung, Y.S., Liaw, C.M. ve Ouyang, M.S. (1995), " Adaptive Speed Control for Induction Motor Drives Using Neural Networks ", IEEE Transaction on Industrial Electronics, Vol.42, No.1, 25-32.

Lai, M.F., Nakano, M. ve Hsieh, G.C. (1996), " Application of Fuzzy Logic on the Phase-Locked Loop Speed Control of Induction Motor Drive ", IEEE Transaction on Industrial Electronics, Vol.43, No.6, 630-639.

Landau, I.D. (1990), System Identification and Control, Prentice Hall.

Leonhardt, W. (1985), Control of Electric Machines, Springer-Verlag.

Levine, W.S. (1996), The Control Handbook, CRC Press and IEEE Press, U.S.A.

Liaw, G.M. ve Lin, F.J. (1994), " A Robust Speed Controller For Induction Motor Drives ", IEEE Transactions on Industrial Electronics, IE-41, 308-315.

Lorenz, R.D., Lipo, T.A. ve Novotny, D.W. (1994), "Motion Control with Induction Motors", Proceedings of IEEE, Vol.82, No.8, 1215-1240.

Mangum, P.M. (1990), A Neural Network Approach to Real-Time Detection of Incipient Faults in Induction Motors, Yüksek Lisans Tezi, North Caroline State University, U.S.A.

Marven, C. ve Ewers, G. (1996), A Simple Approach to Digital Signal Processing, John Willey&Sons. Inc., U.S.A.

Narendra, K.S. ve Parthasarathy, K. (1990), "Identification and Control Dynamic Systems Using Neural Networks", IEEE Trans. Neural Networks, Vol.1, pp.4-27.

Nasar, S.A. (1987), Handbook of Electric Machines, Substitutes New York: McGraw Hill, Bölüm 3 ve 4.

Naylor, D.C.J. (1990), Artificial Neural Networks Review. University of Nottingham Press, Nottingham, Bölüm 6, 7 ve 16.

O'Kelly, D. (1991), Performance and Control of Electrical Machines, McGraw Hill Book Company, Cambridge, Bölüm 8.

Parlos, A.G., Chong, K.T. ve Atıya, A.F. (1994), " Application of the Recurrent Multilayer Perceptron in Modelling Complex Peocess Dynamics", IEEE Trans. Neural Networks, Vol.5, No.2.

Pastacı, H. (1997), Elektronik Devreler, İstanbul, 236-240.

Perdikaris, G.A. (1993), Computer Controlled Systems, Theory and Applications, Kluwer Academic Publishers, Netherlands.

Rumelhart, E., Hinton, G.E. ve Williams, R.J. (1986), "Learning Representation by Back-propagation Errors", Nature 323: 533-536.

Saçkan, A.H. (1994), Asenkron Motorlar, Birsen Yayınevi, Bölüm 1, 2, 5, 6, 7 ve 8.

Simoes, M.G. ve Bose, B.K. (1995), " Neural Network Based Estimation of Feedback Signals for a Vector Controlled Induction Motor Drive", IEEE Transactions on Industry Applications, Vol.31, No.3, 620-628.

Solla, S.A., Levin, E. ve Fleisher, M. (1988), Accelerated Learning in Layered Neural Networks Complex Systems, 2, 625-640.

Söderström, T. ve Stoica, P. (1989), System Identification, Prentice Hall.

Şirin, L. (1996), Rotor Direncinin Uyarlanmasıyla Asenkron Motorun Hız Algılayıcısız Alan Yönlendirmeli Denetimi, Yüksek Lisans tezi, İTÜ Fen Bilimleri Enstitüsü.

Tacer, M.E. ve Tunçay, R.N. (1990), Güç Elektroniği Devreleri, İ.T.Ü. Vakfi, Yayın No.29.

Temel, T. (1996), DSP Reluctance Machine Speed Controller Design Employing Vector Control, Yüksek Lisans Tezi, University of Newcastle Upon Tyne, England.

Texas Instruments TMS320C5x Starter Kit User's Guide, (1996).

Texas Instruments TMS320C50 User's Guide, (1997).

Texas Instruments TMS320C5x DSK Application Guide, (1997).

Texas Instruments C Source Debugger User's Guide, (1991).

Texas Instruments Data Acqusition Circuits Data Book, (1998), Bölüm 4, 3-20.

Texas Instruments Clark and Park Transforms on the TMS320C2xx Application Report, (1996).

Texas Instruments CMOS Data Book, (1997).

Texas Instruments yayını, (1998), Field Orientated Cotrol of 3-Phase AC-Motors, Literature Number: BPRA073.

Texas Instruments yayını, (1997), Sensorless Control with Kalman Filter on TMS320 Fixed-Point DSP, Literature Number: BPRA057.

Türkmen, Y. ve Geçtan, C. (1991), Kumanda Devreleri-1, Yeni Yol Matbaası, İzmir.

Tzau, Y. (1996), " DSP-Based Robust Control of an AC Induction Servo Drive for Motion Control", IEEE Transactions on Control Systems Technology, Vol.4, No.6, 614-626.

Werbos, P.J. (1990), "Backpropagation Through Time: What It Does and How to Do It", Proc. IEEE, Vol.78, 1550-1560.

Vainio, O., Ovaska, S.J. ve Pasanen, J.J. (1992), " A Digital Signal Processing Approach to Real-Time AC Motor Modeling ", IEEE Transactions on Industrial Electronics, Vol.39, No.1, 36-45.

Vas, P. (1990), Vector Control of AC Machines, Oxford University, New York.

Zhang, L., Wathanasam, C. ve Hardan, F. (1994), " An Efficient Microprocessor-Based Pulse With Modulator Using Space Vector Modulation Strategy", IEEE.

Zurada, J.M. (1992), Introduction to Artificial Neural Systems, West Publishing Company, New York.

EKLER

EK 1 Hızlı Delta Kuralı Matematiksel İfadeler

Hızlı Delta Kuralı öğrenme algoritması gradyen iniş metodunu kullanan denklem 4.2'de belirlenen, nesnel fonksiyon $G_k(\lambda)$'yı minimize ederek çıkarılır. Ağın snaptik ağırlıkları w_{pq} için güncel denklem 1.1'de verildiği gibidir.

$$w_{p,k} - w_{p,k-1} = -\alpha \frac{\partial G_k(\lambda)}{\partial w_p} = -\alpha \sum_{i=1}^{n_0} \left[\lambda \phi_2'(e_{i,k}) + (1-\lambda)\phi_1'(e_{i,k})\right] \frac{\hat{c} e_{i,k}}{\hat{c} w_p} \tag{1.1}$$

$\hat{y}_{i,k} = \sigma(\bar{y}_{i,k}) = \sigma(x_k^* w_i) = \sigma(\sum_{j=0}^{n_i} w_{ij} x_{j,k}) \forall i = 1,2,.....,n_0$ 'deki $\bar{y}_{i,k}$ 'nin belirlenmesine göre,

$$\frac{\partial e_{i,k}}{\partial w_p} = -\frac{\partial \hat{y}_{i,k}}{\partial w_p} = -\frac{d\sigma(\bar{y}_{i,k})}{d\bar{y}_{i,k}} \frac{\partial}{\partial w_p} (\sum_{j=1}^{n_i} w_{ij} x_{j,k}) = -\sigma'(\bar{y}_{i,k}) x_k \delta_{ip} \tag{1.2}$$

i=p ise δ_{ip}=1 ve i≠p ise δ_{ip}=0'dır. denklem 1.1 ve 1.2'nin birleşiminden,

$$w_{p,k} = w_{p,k-1} + \alpha \varepsilon_{p,k}^0(\lambda) x_k \tag{1.3}$$

$$\varepsilon_{p,k}^0(\lambda) = \sigma'(\bar{y}_{i,k})\left[\lambda \phi_2'(e_{p,k}) + (1-\lambda)\phi_1'(e_{p,k})\right] \tag{1.4}$$

Ağın çıkışı analog ise σ(x)=x ve σ'(x)=1. Bu durumda $\phi_1(.)$, $\phi_1(x)=(1/\beta)$ ile belirlenir. Buradan $\phi'_1(x)=\tanh(\beta x)$ olduğu kolaylıkla çıkarılabilir. Ve de, $\phi_2(x)=1/2\ x^2$ ve $\phi'_2(x)=x$'tir. Bu durumda, w_{pq} snaptik ağırlıkları denklem 1.3 ile güncelleştirilebilir.

$$\varepsilon_{p,k}^0(\lambda) = \lambda(y_{p,k} - \hat{y}_{p,k}) + (1-\lambda)\tanh\left[\beta(y_{p,k} - \hat{y}_{p,k})\right] \tag{1.5}$$

Ağın çıkışı binary ise σ(x)=tanh(x)'tir. $\sigma'(\bar{y}_{p,k}) = 1 - \sigma(\bar{y}_{p,k})^2 = 1 - \hat{y}_{p,k}^2$ olduğu kolaylıkla görülebilir $E' = \sum_{k=1}^{m} \sum_{i=1}^{n_0} (1 - y_{i,k} \hat{y}_{i,k})$'e göre p'nci birimle ilgili $\phi_1(.)$ fonksiyonu $\phi_1(e_{p,k})=y_{p,k}-e_{p,k}$ ile verilir. Açık bir şekilde, $\phi'_1(e_{p,k})=y_{p,k}$ 'dır. Ve de, $\phi_2(e_{p.k})=1/2e_{p,k}^2$ ve $\phi'_2(e_{p,k})=e_{p,k}$'dır. Bu durumda snaptik ağırlıklar w_{pq} denklem 1.3'le elde edilir.

$$\varepsilon_{p,k}^0(\lambda) = (1 - \hat{y}_{p,k}^2)\left[\ \lambda(y_{p,k} - \hat{y}_{p,k}) + (1-\lambda) y_{p,k}\ \right] = (1 - \hat{y}_{p,k}^2)(y_{p,k} - \lambda \hat{y}_{p,k}) \tag{1.6}$$

EK 2 Hızlı Geriyeyayılım Algoritması Matematiksel İfadeler

Hızlı Geriye Yayılım algoritması denklem 4.2'deki $G_k(\lambda)$'yı minimize ederek elde edilir. Snaptik ağırlıklar w_{pq} için güncel denklem 2.1'de verildiği gibidir.

$$w_{p,k} - w_{p,k-1} = -\alpha \frac{\partial G_k(\lambda)}{\partial w_p} = -\alpha \sum_{i=1}^{n_0} \left[\lambda \phi_2'(e_{i,k}) + (1-\lambda)\phi_1'(e_{i,k})\right] \frac{\hat{c} e_{i,k}}{\hat{c} w_p} \tag{2.1}$$

$\hat{y}_{i,k} = \sigma(\bar{y}_{i,k}) = \sigma(\hat{h}_k^* w_i) = \sigma(\sum_{j=0}^{n_h} w_{ij} \hat{h}_{j,k})$'teki $\hat{y}_{i,k}$'nin belirlenmesinden,

$$\frac{\partial e_{i,k}}{\partial w_p} = -\frac{\partial \hat{y}_{i,k}}{\partial w_p} = -\frac{d\sigma(\bar{y}_{i,k})}{d\bar{y}_{i,k}} \frac{\partial}{\partial w_p} (\sum_{j=1}^{n_h} w_{ij} \hat{h}_{j,k}) = -\sigma'(\bar{y}_{i,k}) \hat{h}_k \delta_{ip} \tag{2.2}$$

Bunu 2.1'de yerine koyarak,

$$w_{p,k} = w_{p,k-1} + \alpha \varepsilon_{p,k}^0(\lambda) \hat{h}_k \tag{2.3}$$

$$\varepsilon_{p,k}^0(\lambda) = \sigma'(\bar{y}_{i,k}) \left[\lambda \phi_2'(e_{p,k}) + (1-\lambda)\phi_1'(e_{p,k})\right] \tag{2.4}$$

Snaptik ağırlıklar v_{pq} için güncel denklem benzer şekilde elde edilebilir. $G_k(\lambda)$'nın belirlenmesinden,

$$v_{p,k} - v_{p,k-1} = -\alpha \frac{\partial G_k(\lambda)}{\partial v_p} = -\alpha \sum_{i=1}^{n_0} \left[\lambda \phi_2'(e_{i,k}) + (1-\lambda)\phi_1'(e_{i,k})\right] \frac{\hat{c} e_{i,k}}{\hat{c} v_p} \tag{2.5}$$

$\hat{y}_{i,k} = \sigma(\bar{y}_{i,k}) = \sigma(\hat{h}_k^* w_i) = \sigma(\sum_{j=0}^{n_h} w_{ij} \hat{h}_{j,k})$'teki $\hat{y}_{i,k}$'nin belirlenmesinden,

$$\frac{\partial e_{i,k}}{\partial v_p} = -\frac{\partial \hat{y}_{i,k}}{\partial v_p} = -\frac{d\sigma(\bar{y}_{i,k})}{d\bar{y}_{i,k}} \frac{\partial}{\partial v_p} (\sum_{j=1}^{n_h} w_{ij} \hat{h}_{j,k}) = -\sigma'(\bar{y}_{i,k}) \sum_{j=1}^{n_h} w_{ij} \frac{\partial \hat{h}_{j,k}}{\partial v_p} \tag{2.6}$$

Sonuç olarak, $\hat{h}_{j,k}$'nın belirlenmesi denklem 2.7'yi verir,

$$\frac{\partial \hat{h}_{j,k}}{\partial v_p} = \frac{d\rho(\bar{h}_{j,k})}{d\bar{h}_{j,k}} \frac{\partial}{\partial v_p} \left(\sum_{l=1}^{n_i} v_{jl} x_{l,k}\right) = \rho'(\bar{h}_{j,k}) x_k \delta_{jp} \tag{2.7}$$

denklem 4.4 ve 4.5'in denklem 4.3 ile birleşmesiylesnaptik ağırlıklar v_{pq} için aşağıdaki güncel denklem elde edilir.

$$v_{p,k} = v_{p,k-1} + \alpha \varepsilon^h_{p,k}(\lambda) x_k \tag{2.8}$$

$$\varepsilon^h_{p,k}(\lambda) = \rho'(\bar{h}_{i,k}) \sum_{i=1}^{n_0} \sigma'(\bar{y}_{i,k}) \left[\lambda \phi_2'(e_{i,k}) + (1-\lambda)\phi_1'(e_{i,k})\right] w_{ip} \tag{2.9}$$

Ağın çıkışı analog ise, $\rho(x)=\tanh(x)$ ve $\rho'(x)=1-\rho(x)^2$ iken $\sigma(x)=x$ ve $\sigma'(x)=1$'dir. Bu durumda, $\phi_1(.)$, $\phi_1(x)=(1/\beta)\ln(\cosh\beta x)$'tir. Buradan $\phi_1(x)=\tanh(\beta x)$ olarak kolaylıkla elde edilebilir. Ve de, $\phi_2(x)=1/2\ x^2$ ve $\phi'_2(x)=x$ 'tir. Bu durumda, snaptik ağırlıklar w_{pq} denklem 2.3 ile güncelleştirilebilir,

$$\varepsilon^0_{p,k}(\lambda) = \lambda(y_{p,k} - \hat{y}_{p,k}) + (1-\lambda)\tanh\left[\beta(y_{p,k} - \hat{y}_{p,k})\right] \tag{2.10}$$

Snaptik ağırlıklar v_{pq} denklem 2.8 ile güncelleştirilebilir,

$$\varepsilon^h_{p,k}(\lambda) = (1 - \hat{h}^2_{p,k}) \sum_{i=1}^{n_0} \varepsilon^0_{i,k}(\lambda) w_{ip} \tag{2.11}$$

ve $\varepsilon^0_{p,k}(\lambda)$ denklem 2.10'la verilir.

Ağın çıkışı binary ise, $\sigma(x)=\tanh(x)$ ve $\sigma'(x)= 1-\sigma(x)^2$ 'dir. Buna ilave olarak, $\rho(x)=\tanh(x)$ ve $\rho'(x)=1-\rho(x)^2$ 'dir. $E' = \sum_{k=1}^{m} d(y_k, \hat{y}_k) = \sum_{k=1}^{m} \sum_{i=1}^{n_0} (1 - y_{i,k}\, \hat{y}_{i,k})$ 'e göre, p'nci birimle ilgili $\phi_1(.)$ fonksiyonu $\phi_1(e_{p,k})= y_{p,k}\ e_{p,k}$ ile verilir. Açık bir şekilde, $\phi'_1(e_{p,k})= y_{p,k}$ 'dır. Ve de, $\phi_2(e_{p,k})= 1/2\ e_{p,k}^2$ ve $\phi'_2(e_{p,k})= e_{p,k}$ 'dır. Bu durumda, snaptik ağırlıklar w_{pq} denklem 2.3 ile güncelleştirilebilir ve denklem 2.4 aşağıdaki denklem olarak basitleşir,

$$\varepsilon^0_{p,k}(\lambda) = (1 - \hat{y}^2_{p,k})(y_{p,k} - \lambda \hat{y}_{p,k}) \tag{2.12}$$

Sonuç olarak, snaptik ağırlıklar v_{pq} için güncel denklem 2.8'le verilidiği gibidir,

$$\varepsilon^{h}_{p,k}(\lambda) = (1-\hat{h}^{2}_{p,k})\sum_{i=1}^{n_0}\varepsilon^{0}_{i,k}(\lambda)w_{ip} \qquad (2.13)$$

ve $\varepsilon^{0}_{p,k}(\lambda)$ 2.12'yle verilir.

Yukarıdaki analizin bir genellemesi birden fazla gizli katmana sahip sinir ağlarının eğitilmesinde Hızlı Geriye Yayılım algoritmasına izin verir. Bu ağla sağlanan tahminler, $\hat{h}^{(1)}_{k} = \hat{h}_{k}$ için $\hat{y}_{i,k} = \sigma(\sum_{j=0}^{n_1} v^{(0)}_{ij}\hat{h}^{(1)}_{j,k})\ \forall i = 1,2,.....,n_0$ 'le tanımlı olan $\hat{y}_{i,k} = \sigma(\bar{y}_{i,k}) = \sigma(\hat{h}^{*}_{k}w) = \sigma(\sum_{j=0}^{n_h} w_{ij}\hat{h}_{j,k})$'le verilir. Bir sonuç olarak, snaptik ağırlıklar w_{pq} için güncel denklem aşağıdaki gibi $\hat{h}^{(1)}_{k} = \hat{h}_{k}$ için denklem 2.3'den elde edilebilir.

$$w_{p,k} = w_{p,k-1} + \alpha\varepsilon^{0}_{p,k}(\lambda)\hat{h}^{(1)}_{k} \qquad (2.14)$$

Burada $\varepsilon^{(0)}_{p,k} = \varepsilon^{0}_{p,k}(\lambda)$ ağın çıkışı analog ise 2.10 ile, binary ise 2.12 ile verilir. Snaptik ağırlıklar $v^{(r)}_{pq}$, r= 1,2,....,L aşağıdaki denklem aracılığıyla denklem 4.2'de belirtilen $G_k(\lambda)$ nesnel fonksiyonunu minimize ederek güncelleştirilebilir.

$$v^{(r)}{}_{p,k} - v^{(r)}{}_{p,k-1} = -\alpha\frac{\partial G_k(\lambda)}{\partial v^{(r)}{}_{p}} = -\alpha\sum_{i=1}^{n_0}\left[\lambda\phi_2'(e_{i,k}) + (1-\lambda)\phi_1'(e_{i,k})\right]\frac{\partial e_{i,k}}{\partial v^{(r)}{}_{p}} \qquad (2.15)$$

Denklem 2.15'in snaptik ağırlıklar $v^{(1)}_{pq}$ için aşağıdaki güncel denklem şeklinde sonuçlandığı gösterilebilir.

$$v^{(1)}{}_{p,k} = v^{(1)}{}_{p,k-1} + \alpha\varepsilon^{(1)}_{p,k}(\lambda)\hat{h}^{(2)}_{k} \qquad (2.16)$$

$$\varepsilon^{(1)}_{p,k}(\lambda) = (1-\hat{h}^{(1)^2}_{p,k})\sum_{i=1}^{n_0}\varepsilon^{0}_{i,k}(\lambda)w_{ip} \qquad \text{olduğu yerde,} \qquad (2.17)$$

ve $\varepsilon_{i,k}^{(l)}(\lambda)$ denklem 2.10 veya 2.12'yle ağın çıkışı sırasıyla analog veya binary olması durumunda belirlenir. Yukarıdaki sonuçların direkt olarak genellemesi $v_{pq}^{(r)}$, r=2,3,.....,L snaptik ağırlıklar için aşağıdaki güncel denklemi sağlar.

$$v_{p,k}^{(r)} = v_{p,k-1}^{(r)} + \alpha \varepsilon_{p,k}^{(r)}(\lambda) \hat{h}_k^{(r+1)} \quad \forall r = 0,1,....,L \tag{2.18}$$

$\hat{h}_k^{(L+1)}$ =x_k ve

$$\varepsilon_{p,k}^{(r)}(\lambda) = (1 - \hat{h}_{p,k}^{(r)\,2}) \sum_{l=1}^{n_{r-1}} \varepsilon_{l,k}^{(r-1)}(\lambda) v_{lp}^{(r-1)} \quad \forall r = 1,2,.....,L \tag{2.19}$$

EK 3 Klasik Hatanın Geriyeyayılım Algoritması YSA Programı

Farcalloc ve huge komutlarıyla matris yapısı iptal edilip sınırsız data ve katman girilmesi mümkündür.
Eğitme seti dat uzantılı dosyadan okunup, sonuçlar girişlerin en sonuna eklenmektedir. Test fazında ise dosyadan okunup, dosyaya sonuçlar kaydedilmektedir. 1 ile mevcut bulunan ağırlık dosyalarından işleme devam edilmektedir.
*.dat eğitme dosyası, buradaki * eğitme dosyasının ismini belirtmektedir, dosyanın ilk kısmı girişler, ikinci kısmı o girişler için doğru çıkışlardır.
*_v.dat eğitme özelliklerinin olduğu dosyadır.
*_w.dat ağırlıkların depolandığı dosyadır.
*.dat test (çıkış) dosyası, buradaki * test için hazırlanan giriş setlerinin olduğu dosya ismini belirtmektedir.
*ht.dat, test dosyasındaki setler için YSA' nın bulduğu sonuçların depolandığı dosyadır.
hata+iter dosyası, ismi eğitme dosyasının ismi ile aynı, uzantısı .err ve eğer eski ağırlıklar üzerine devam ediliyorsa hata dosyasının da üzerine devam edip, silmemekte aksi taktirde silmektedir..
*y.cns dosyasına da, eğittiği dosya için ürettiği çıkışları kaydedmektedir, fakat bu grafik kısmını içermemektedir.

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <conio.h>
#include <stdlib.h>
#include <process.h>
#include <alloc.h>
#include<ctype.h>
#ifndef VAX
#include<alloc.h>
#endif
#define NMXUNIT   10
#define NMXHLR   5   /* maksimum gizli katman (değiştirilebilir)*/
#define NMXOATTR 32   /* maksimum çıkış sayısı (değiştirilebilir)*/
#define NMXINP   50  /*maksimum giriş grubu sayısı (değiştirilebilir)*/
#define NMXIATTR  500  /*maksimum giriş sayısı   (değiştirilebilir) */
#define SEXIT     3
#define RESTRT   2
#define FEXIT    1
#define CONTNE  0
float eta, alpha, err_curr, maxe, maxep;
float far *wtptr[NMXHLR+1];
float  far *outptr[NMXHLR+2];
float  far *errptr[NMXHLR+2];
float far *delw[NMXHLR+1];
float  far ep[NMXINP];
int long nunit[NMXHLR+2],nhlayer,ninput,ninattr,noutattr,sd;
long i;
```

```
int long  result,errorcode;
 int long cnt_num,cnt,eski;
int long nsnew,nsold;
char task_name[20],dr;
FILE *fp1,*fp2,*fp3,*fopen(),*fpf,*foutf1,*foutf2;
int long fplot10;
long randseed=568731L;
char def[30],def1[20];
char *sec[4]
={"1- DOSYALARI OLUŞTURMA", "2-EĞİTİM VE DENEME İŞLEMİ ", "3-ÇIKIŞ
GRAFİĞİ", "4-ÇIKIŞ"};
void *p;
char atama1[10],atama2[10],atama3[10];
int long hih,hih1,hih2,hih3,fft,d1,d2;
float cn,cn1[10000],enbu=0.0,enku=100.0;
double fe,fe1,de,de1,de2,se,cnt1,fft1;
float huge *target;
float huge *input;
float huge *outpt;
          void  yer(void){

input=(float huge *)farcalloc(275000,sizeof(float));
/* float far target[NMXINP][NMXOATTR];*/
outpt=(float huge *)farcalloc(275000,sizeof(float));
/*float far  input[NMXINP][NMXIATTR];   */
target=(float huge *)farcalloc(275000,sizeof(float));
/*float far outpt[NMXINP][NMXOATTR];*/
}

void gitlan(){ clrscr();printf("Performed Iteration=%ld",cnt);
  printf("\n This Time Error    =%f\n",err_curr);
  printf("Çıkmak istiyor musunuz? 'd(dur)-herhangi bir tuş'a basın");
  i=getch();if(getch()=='d')cnt=cnt_num-1;clrscr();printf("\n Performs iterations PLEASE
DON'T TOUCH!!!\n If you want to use the computer:\n1) please press a button then
wait\n2) appears a table that contains error term\n (it takes approximately 1 minute)\n3)
after that please press 'd' button\n again please wait a minute\n4) then you see a message
that reads 'please press a button' \n please press a button after that wait a minute. Thanks a
lot for doing these steps.");

 }
int random()
 {
 randseed=15625L*randseed+22221L;
 return((randseed>>16)&0x7FFF);
 }
init()
 {
  int long len1,len2,i,k;
```

```
 float huge *p1,huge *p2,huge *p3,huge *p4;
 len1=len2=0;
 nunit[nhlayer+2]=0;
 for(i=0;i<(nhlayer+2);i++){
 len1+=(nunit[i]+1)*nunit[i+1];
 len2+=nunit[i]+1;
 }
 p1=(float huge *)farcalloc(400000,sizeof(float));
 p2=(float huge *) farcalloc(len2+1,sizeof(float));
 p3=(float huge *) farcalloc(len2+1,sizeof(float));
 p4=(float huge *) farcalloc(400000,sizeof(float));
 wtptr[0]=p1;
 outptr[0]=p2;
 errptr[0]=p3;
 delw[0]=p4;
 for (i=1;i<(nhlayer+1);i++){
 wtptr[i]=wtptr[i-1]+nunit[i]*(nunit[i-1]+1);
 delw[i]=delw[i-1]+nunit[i]*(nunit[i-1]+1);
 }
  for (i=1;i<(nhlayer+2);i++){
 outptr[i]=outptr[i-1]+nunit[i-1]+1;
 errptr[i]=errptr[i-1]+nunit[i-1]+1;
 }
 for (i=0;i<nhlayer+1;i++){
 *(outptr[i]+nunit[i])=1.0;
 }
 return(0);
 }

initwt()
 {
 int long  i,j ;
 for (j=0;j<nhlayer+1;j++)
 for (i=0;i<(nunit[j]+1)*nunit[j+1];i++){
    *(wtptr[j]+i)=random()/pow(2.0,15.0)-0.5;
    *(delw[j]+i)=0.0;
  }
  return(0);
  }
set_up()
 {
 int long i;
 eta=0.9;
 printf("\nMonentum rate eta (default=0.9)?: ");
 scanf("%f",&eta);
 alpha=0.7;
 printf("\nLearning rate alfa (default=0.7)?: ");
 scanf("%f",&alpha);
```

```
 maxe=0.0000001;maxep=0.0000001;
 printf("\nMax total error (default=0.00001)?: ");
 scanf("%f",&maxe);
  printf("\nMax individual error(default=0.00001)?: ");
 scanf("%f",&maxep);
 cnt_num=1000;
 printf("\nMax number of iteration  (default=1000)?: ");
 scanf("%ld",&cnt_num);
 printf("\nNumber of hidden layers?: ");
 scanf("%ld",&nhlayer);
   for(i=0;i<nhlayer;i++) {
   printf("\n\tNumber of units for hidden layer %ld?:",
   i+1);
   scanf("%ld",&nunit[i+1]);
   }
   printf("\nCreate error file ? If so type 1, or type 0 : ");
         scanf("%ld",&fplot10);clrscr();
   printf("\nPerforms iterations PLEASE DON'T TOUCH!!!\n If you want to use the
computer:\n1) please press a button then wait\n2) appears a table that contains error term\n
(it takes approximately 1 minute)\n3) after that please press 'd' button\n again please wait a
minute\n4) then you see a message that reads 'please press a button' \n please press a button
after that wait a minute. Thanks a lot for doing these steps.");
         nunit[nhlayer+1]=noutattr;
   nunit[0]=ninattr;
   return(0);
   }
dread(char *taskname)

   {
    int long i,j,c;
    char var_file_name[20];
    strcpy(var_file_name,taskname);
    strcat(var_file_name,"_v.dat");
    if (( fp1=fopen(var_file_name,"r"))==NULL)
    {
     perror("\n Cannot open data file ");
     exit(0);
     }

fscanf(fp1,"%lu%lu%lu%f%f%lu%lu\n",&ninput,&noutattr,&ninattr,&eta,&alpha,&nhlay
er,&cnt_num);
     for(i=0; i<nhlayer+2; i++)
                  fscanf(fp1,"%ld",&nunit[i]);
     if ((c=fclose(fp1))!=0)
       printf("\nFile cannot be closedSSSS %ld ",c);
       return(0);
      }
wtread(char *taskname)
```

```
        {
      int long  i,j,c ;
      char wt_file_name[20];
      strcpy(wt_file_name,taskname);
      strcat(wt_file_name,"_w.dat");
      if ((fp2=fopen(wt_file_name,"r"))==NULL)
      {
       perror("\n Cannot open data file");
       exit(0);
       }
       for (i=0; i<nhlayer+1;i++){
         for (j=0; j<(nunit[i]+1)*nunit[i+1];j++) {
                    fscanf (fp2,"%f",(wtptr[i]+j));
          }
        }
       if ((c=fclose(fp2))!=0)
         printf("\nFile cannot be closedSSS %ld",c);
       return(0);
       }
dwrite(char *taskname)
       {
        int long  i,j,c;
       char var_file_name[20];
      strcpy(var_file_name,taskname);
      strcat(var_file_name,"_v.dat");
      if ((fp1=fopen(var_file_name,"w+"))==NULL)
      {
       perror("\n Cannot open data file");
       exit(0);
       }
      fprintf(fp1,"%lu%lu%lu%f%f%lu%lu\n",ninput,noutattr,ninattr,eta,alpha,nhlayer,c
nt_num);
   for (i=0; i<nhlayer+2;i++){
         fprintf(fp1,"%ld ",nunit[i]);
        }

       fprintf(fp1,"\n");
       for (i=0; i<ninput;i++){
         for (j=(i*noutattr); j<((i+1)*noutattr);j++)
          fprintf (fp1,"%f      ",outpt[j]);
          fprintf (fp1,"\n");
         }
       if ((c=fclose(fp1))!=0)
         printf("\nFile cannot be closedw %ld",c);
       return(0);
       }
wtwrite( char *taskname)
       {
```

```
        int long  i,j,c,k;
              char wt_file_name[20];
      strcpy(wt_file_name,taskname);
      strcat(wt_file_name,"_w.dat");
      if ((fp2=fopen(wt_file_name,"w+"))==NULL)
      {
       perror("\n Cannot open data file");
       exit(0);
       }
       k=0;
       for (i=0; i<nhlayer+1;i++)
          for (j=(i*((nunit[i]+1)*nunit[i+1])); j<((i+1)*((nunit[i]+1)*nunit[i+1]));j++){
          if(k==8) {
          k=0;
          fprintf (fp2,"\n");
          }
          fprintf (fp2,"%f ",*(wtptr[i]+j%((nunit[i]+1)*nunit[i+1])));
          k++;
       }
       if ((c=fclose(fp2))!=0)
         printf("\nFile cannot be closedSS %ld",c);
       return(0);
       }
void forward(i)
       {

        int long m,n,p,offset;
        float net;

       for (m=(i*ninattr);m<((i+1)*ninattr);m++)
          *(outptr[0]+(m%ninattr))=input[m];
          for (m=1; m<nhlayer+2;m++){
         for (n=0; n<nunit[m];n++){

          net=0.0;
                for (p=0; p<nunit[m-1]+1;p++){
               offset=(nunit[m-1]+1)*n+p;
                net += *(wtptr[m-1]+offset)
      *(*(outptr[m-1]+p));
      }
      *(outptr[m]+n)= 1/(1+exp(-net));
           }
           }
        for(n=((i)*nunit[nhlayer+1]); n<((i+1)*nunit[nhlayer+1]);n++)
          outpt[n]=*(outptr[nhlayer+1]+(n%nunit[nhlayer+1]));
           }

int long introspective (int long nfrom,int long nto)
```

```
    {
    int long i,flag;
    if (cnt>=cnt_num) return(FEXIT);
    nsnew=0;
    flag=1;
    for (i=nfrom;(i<nto)&&(flag==1);i++){
    if (ep[i]<=maxep)nsnew++;
    else flag=0;
    }
    if (flag==1) return(SEXIT);
    if (err_curr<=maxe) return(SEXIT);
    return(CONTNE);
    }
int long rumelhart(int long from_snum,int long to_snum)
    {
    int long i,j,k,m,n,p,offset,index;
    float out;
    char err_file_name[20],sonuc[20];
    strcpy(err_file_name,task_name);
    strcat(err_file_name,".err");
    nsold=0;
    cnt=0;
    result=CONTNE;
    if ((fp3=fopen(err_file_name,"w"))==NULL)
    {
    perror("Cannot open error file");
    exit(0);
    }
    do {
    err_curr=0.0;
    for (i=from_snum;i<to_snum;i++){
    forward(i);
    for(m=(i*nunit[nhlayer+1]);m<((i+1)*nunit[nhlayer+1]);m++){
    out=*(outptr[nhlayer+1]+(m%nunit[nhlayer+1]));
    *(errptr[nhlayer+1]+(m%nunit[nhlayer+1]))=(target[m]-out)
                    *(1-out)*out;
    }
  for (m=nhlayer+1;m>=1;m--){
  for (n=0; n<nunit[m-1]+1;n++){
   *(errptr[m-1]+n)=0.0;
   for (p=0; p<nunit[m];p++){
        offset=(nunit[m-1]+1)*p+n;
   *(delw[m-1]+offset)=eta*(*(errptr[m]+p))
   *(*(outptr[m-1]+n))
    +alpha*(*(delw[m-1]+offset));
   *(errptr[m-1]+n)+=*(errptr[m]+p)
   *(*(wtptr[m-1]+offset));
    }
```

```
    *(errptr[m-1]+n)=*(errptr[m-1]+n)*
    (1-*(outptr[m-1]+n))*(*(outptr[m-1]+n));
}
}
     for (m=1;m<nhlayer+2;m++){
for(n=0;n<nunit[m];n++){
for(p=0;p<nunit[m-1]+1;p++){

              offset=(nunit[m-1]+1)*n+p;
      *(wtptr[m-1]+offset)+=*(delw[m-1]+offset);
      }
      }
      }
      ep[i]=0.0;
      for(m=(i*nunit[nhlayer+1]);m<((i+1)*nunit[nhlayer+1]);m++){
       ep[i]+=fabs((target[m]-
       *(outptr[nhlayer+1]+(m%nunit[nhlayer+1]))));
      }
      err_curr+=ep[i]*ep[i];
      }
      err_curr=0.5*err_curr/ninput;

      if(kbhit())gitlan();
      fprintf(fp3,"%ld%f\n",cnt,err_curr);
      cnt++;
      result=introspective(from_snum,to_snum);
      }
      while (result==CONTNE);

          for (i=from_snum; i<to_snum;i++) forward(i);
      for(i=0; i<nhlayer+1;i++){
          index=0;
      for (j=0;j<nunit[i+1];j++)
          {
          printf("\n\n Weights between unit %ld of layer %ld", j,i+1);
          printf("and units of layer %ld\n",i);   for (k=0; k<nunit[i];k++)
          printf ("%f",*(wtptr[i]+index++));
          printf ("\n Threshold of unit %ld of layer%ldis%f",  j,i+1,*(wtptr[i]+index++));
      }
      }

      strcpy(sonuc,task_name);
      strcat(sonuc,"y.cns");
       if ((fpf=fopen(sonuc,"w"))==NULL)
          {
      perror("Veri dosyasi açilamiyor");
      exit(0);
       }
```

```
    for(i=0;i<ninput;i++){/*\nsample%ldoutput%ld=target%ld=%fi,j%noutattr,j%noutattr,
target[j]*/
          for (j=(i*noutattr);j<((i+1)*noutattr);j++){ Yukarıda yazılan kısmı fprintften
çıkardık
         printf("\n\n  output %ld=%f outpt %ld=%f ", /*set sonuçlarını ekrana bastırma işi*/
        j%noutattr,outpt[j],j%noutattr,*(outptr[nhlayer+1]+(j%noutattr)));

        fprintf(fpf,"%f\n",outpt[j]); }}
            printf("\n\n Total number of iteration is %ld",cnt);
    printf("\n Normalized system error is %f\n\n\n",err_curr);
    fclose(fpf);
    printf("\n Please press a buton for continue");
    getch();
    return(result);
    }
user_session()
    { int long i,j,showdata;              /*  Burada ismi verilen bir */
        char fnam[20],dtype[20];          /*dosyadan belirtilen sayıdaki giriş */
        FILE *fp;                         /* için yine girilen sayıdaki verileri okuyor*/
        printf("\n Start of learning session");
        printf("\n\t Enter the task name : ");
        scanf("%s",task_name);
        printf("\n How many features in input pattern?:");
        scanf("%ld",&ninattr);
        printf("\n How many output units?: ");
        scanf("%ld",&noutattr);
        printf("\n Total number of input samples?: ");
        scanf("%ld",&ninput);
        /* Burada input[i][j] tarzindaki veri yazmada i 0'dan ninput-1(giris orneklerinin
toplam sayisi)'na kadar giris yapilacak*/
        /*Burada input[i][j] tarzindaki veri yazmada j 0'dan ninattr-1(her numune icin giris
sayisi)'na kadar giris yapilacak*/

        strcpy(fnam,task_name);
        strcat(fnam,".dat");
        printf("\n Input File name is %s",fnam);
        if ((fp=fopen(fnam,"r"))==NULL)
        {
        printf("\n File %s does not exits", fnam);
        exit(0);
        }

        printf("\n Do you want to look at data just read?");
        printf("\n   Answer yes or no :");
        scanf("%s",dtype);
        showdata=((dtype[0]=='y')||(dtype[0]=='Y'));
         for (i=0;i<ninput;i++){
```

```
            for (j=(i*ninattr);j<((i+1)*ninattr);j++) {
            fscanf(fp,"%f",&input[j]);
                                if (showdata)printf("%ld  %f\n ",j,input[j]);
            }}


           for (i=0;i<ninput;i++){
           for (j=(i*noutattr);j<((i+1)*noutattr);j++) {
              fscanf(fp,"%f",&target[j]);
            if (showdata)printf(" %f  ",target[j]);
           }
           }
           if ((i=fclose(fp))!=0)
          {
          printf("\n File56 cannot be closed %ld",i);
          exit(0);
          }

           return(0);
      }
learning()
      {
          int long result;
          user_session();
          set_up();
          init();
          do {
       initwt();
       result=rumelhart(0,ninput);
       } while (result==RESTRT);
       if (result==FEXIT)
       {
       printf("\n Max number of iterations reaced,");
       printf("\n but failed to decrease system");
       printf("\n error sufficiently");
       }
       dwrite(task_name);
       wtwrite(task_name);
       return(0);
       }

/* Aşağıdaki fonksiyon önceden belli bir iterasyonla eğitilmiş dosyanın iterasyon sayısını
arttırmak için eklenen kısım */
learning1()
       {
     eski=1;
     int long result;
     int long i,j,showdata;              /*  Burada ismi verilen bir */
```

```
char fnam[20],dtype[20];        /*dosyadan belirtilen sayıdaki giriş */
FILE *fp;                       /* için yine girilen sayıdaki verileri okuyor*/
 printf("\n Start of learning1 session");
 printf("\n\t Enter the task name : ");
 scanf("%s",task_name);

            dread(task_name);
            init();
         strcpy(fnam,task_name);
    strcat(fnam,".dat");
    printf("\n Input File name is %s",fnam);
    if ((fp=fopen(fnam,"r"))==NULL)
   {
    printf("\n File %s does not exits", fnam);
    exit(0);
    }

     printf("\n Do you want to look at data just read?");
     printf("\n   Answer yes or no :");
     scanf("%s",dtype);
     showdata=((dtype[0]=='y')||(dtype[0]=='Y'));
       for (i=0;i<ninput;i++){
              for (j=(i*ninattr);j<((i+1)*ninattr);j++) {
                     fscanf(fp,"%f",&target[j]);
                    if (showdata)printf("%f\n",target[j]);
              }}

              for (i=0;i<ninput;i++){
              for (j=(i*noutattr);j<((i+1)*noutattr);j++) {
                     fscanf(fp,"%f",&target[j]);
                    if (showdata)printf(" %f\n",target[j]);
              }
              }
              if ((i=fclose(fp))!=0)
     {
      printf("\n File56 cannot be closed %ld",i);
      exit(0);
      }


printf("\nMax number of iteration  (default=1000)?: ");
scanf("%ld",&cnt_num);
              printf("\nCreate error file ? If so type 1, or type 0 : ");
      scanf("%ld",&fplot10);clrscr();
      printf("\nPerforms iterations PLEASE DON'T TOUCH!!!");
      nunit[nhlayer+1]=noutattr;
      nunit[0]=ninattr;
              do {
```

```
        wtread(task_name);
        result=rumelhart(0,ninput);
        } while (result==RESTRT);
        if (result==FEXIT)
        {
        printf("\n Max number of iterations reaced,");
        printf("\n but failed to decrease system");
        printf("\n error sufficiently");
        }
        dwrite(task_name);
        wtwrite(task_name);
        return(0);
        }

output_generation()
        {
         int long i,m,nsample;
         char ans[10];
         char dfile[20];
         printf("\n Generation of outputs for a new pattern");
         printf("\n\t Present task name is %s", task_name);
        printf("\n\t Work on a different task?");
         printf("\n\t Answer yes or no :");
         scanf("%s",ans);
                if ((ans[0]=='y')||(ans[0]=='Y'))
        {
         printf("\n\t Type the task name :");
         scanf("%s",task_name);
                dread(task_name);
                init();
                wtread(task_name);
                }
  printf("\n Enter file name for patterns to ");
  printf(" be processed: ");
  scanf("%s",dfile);
  if ((fp1=fopen(dfile,"r"))==NULL)
  {
  perror("Connat open dfile");
  exit(0);
  }


                /* printf("\n Enter number of patterns for processing :");
                        scanf("%ld",&nsample);
                        for (i=0;i<nsample;i++)
                        for (m=(i*ninattr); m<((i+1)*ninattr);m++)
                { printf("%ld veri=",((m%ninattr)+1));
```

```
                scanf("%f",&input[m]); }
                tek tek veri girme kısmı*/

        /* *.dat veri dosyasından(dosya1) istenilen sayıdaki set girişi okuyup
sonucu (YSA sonucunu yani) *ht.dat dosyasına kaydediyor
       nsample=1 satırından for (i=0; i<nsample;i++) satırına kadar devam ediyor.*/
        nsample=1/*aslında bu gerekmiyor sanki bu test sayısı daha sonra isteniyor*/;
                char dosya1[20],dosya2[20],cont[20];
                float ss;
                printf("veri dosyasının ismini giriniz:");
                scanf ("%s",dosya1);
                strcpy(dosya2,task_name);
                strcat(dosya2,"ht .dat");
                printf("%s\n",dosya2);
                strcat(dosya1,".dat");
                printf("%s\n",dosya1);

       if((foutf1=fopen(dosya1,"r"))==NULL)
                {
       perror("Connat open dfile");
       exit(0);
       }
        printf("\n Enter number of patterns for processing:\n(test dosyası set sayısı yani) ");
       scanf("%ld",&nsample);
       /*test datalarının bulunduğu dosyadaki set sayısı*/

        for (i=0;i<nsample;i++){
       for (m=(i*ninattr); m<((i+1)*ninattr);m++)
                {
 fscanf(foutf1,"%f",&input[m]);

   }            }

   fclose(foutf1);
   if((foutf2=fopen(dosya2,"w+"))==NULL)
        {
  perror("Connat open dfile");
  exit(0);
  }


  for (i=0; i<nsample;i++)
 {
        forward(i);
        for(m=(i*noutattr);m<((i+1)*noutattr);m++){

printf("\nsample%ldoutput%ld=%f",i,(m%noutattr),ss=*(outptr[nhlayer+1]+(m%noutattr))
);
```

```
printf("\n");
fprintf(foutf2,"%f\n",   /*Çıkış hatasını hata.dat dosyasına kopyalıyor*/
                  *(outptr[nhlayer+1]+(m%noutattr)) );
printf("%f\n",outpt[m],   /*Çıkış hatasını hata.dat dosyasına kopyalıyor*/
                  *(outptr[nhlayer+1]+(m%noutattr)) );
          }
                    }
    if ((i=fclose(foutf2))!=0)
          printf("\nFile cannot be closedS %ld",i);
          printf("\nOutput have been generated ");
          return(0);
          }

void der()
  {
         FILE *f;
         if ((f=fopen("veri2.dat","r"))==NULL)
         {
          perror("Cannot open err_file");
          exit(0);
         }
         i=1;
         while(!(feof(f)))
         {
           fscanf(f," %ld %f\n",&cnt1,&cn);
           i++;
         }
          fclose(f);
          fft=i;
          se=550.0;
          de=fft/se;
          de1=fft/2.0 ;
          de2=fft/4.0  ;
          hih2=floor(de1);
          hih1=floor(de2);

          hih=floor(de);
          if ((f=fopen("veri2.dat","r"))==NULL)
         {
          perror("Cannot open err_file");
          exit(0);
         }
         sd=0;

         for(i=1;i<fft;i++)
         {
          fscanf(f," %ld %f\n",&cnt1,&cn);
          if(enbu<cn)enbu=cn;
```

```
        if(enku>cn)enku=cn;
        if(i/(hih+1.0)==floor(i/(hih+1.0))){ sd=sd+1;cn1[sd]=cn*10;}
        }
         itoa(hih1, atama1, 10);
        itoa(hih2, atama2, 10);
        itoa(fft, atama3, 10);

         if ((i=fclose(f))!=0)
        printf("\nFile cannot be closed13 %ld",i);
}

int long  main1()
{       char select[20],cont[20];
        strcpy(task_name,"*********");
         do { printf("Be careful Sir! max input #=%d   max output #=%d\n        max     set
#=%d",NMXIATTR,NMXOATTR,NMXINP);
         printf("\n*Select L(earning) or O(utput generatoin)\n or 1(continue from old
weights file)*\n");
        do {

         scanf ("%s",select);
      switch(select[0]){
      case 'o':
      case 'O':
         output_generation();
         break;
         case 'l':
         case 'L':
         learning();

                break;
          case '1':
          learning1();
          break;
          default:
           printf("\n answer learning(l) or output generation(o)\n or 1(continue from old
weights file)");
          break;
                }
        }
          while ((select[0]!='o')&&(select[0]!='O')
          &&(select[0]!='l')&&(select[0]!='L')&&(select[0]!='1'));
          printf("\nDo you want to continue?");
          scanf("%s",cont);
          }

   while ((cont[0]=='y')||(cont[0]=='Y'));
         printf("\nIt is all finished. ");
```

```
            printf("\n Good bye");

            return(0);
}
int long main(void)
{
yer();
main1();
return 0;
}
```

EK 4 Hızlı Hatanın Geriyeyayılımı Algoritması YSA Programı

Farcalloc ve huge komutlarıyla matris yapısı iptal edilip sınırsız data ve katman girilmesi mümkündür.
Eğitme seti dat uzantılı dosyadan okunup, sonuçlar girişlerin en sonuna eklenmektedir. Test fazında ise dosyadan okunup, dosyaya sonuçlar kaydedilmektedir. 1 ile mevcut bulunan ağırlık dosyalarından işleme devam edilmektedir.
*.dat eğitme dosyası, buradaki * eğitme dosyasının ismini belirtmektedir, dosyanın ilk kısmı girişler, ikinci kısmı o girişler için doğru çıkışlardır.
*_v.dat eğitme özelliklerinin olduğu dosyadır.
*_w.dat ağırlıkların depolandığı dosyadır.
*.dat test (çıkış) dosyası, buradaki * test için hazırlanan giriş setlerinin olduğu dosya ismini belirtmektedir.
*ht.dat, test dosyasındaki setler için YSA' nın bulduğu sonuçların depolandığı dosyadır.
hata+iter dosyası, ismi eğitme dosyasının ismi ile aynı, uzantısı .err ve eğer eski ağırlıklar üzerine devam ediliyorsa hata dosyasının da üzerine devam edip, silmemekte aksi taktirde silmektedir..
*y.cns dosyasına da, eğittiği dosya için ürettiği çıkışları kaydedmektedir, fakat bu grafik kısmını içermemektedir.

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <conio.h>
#include <stdlib.h>
#include <process.h>
#include <alloc.h>
#include<ctype.h>
#ifndef VAX
#include<alloc.h>
#endif
#define NMXUNIT   10
#define NMXHLR   5   /*maksimum gizli katman (değiştirilebilir)*/
#define NMXOATTR 32   /* (maksimum çıkış sayısı değiştirilebilir)*/
#define NMXINP   50  /*maksimum giriş grubu sayısı (değiştirilebilir)*/
#define NMXIATTR  500  /*maksimum giriş sayısı   (değiştirilebilir) */
#define SEXIT     3
#define RESTRT   2
#define FEXIT   1
#define CONTNE  0
float eta, alpha, err_curr, maxe, maxep,beta=.3,lamda=1;
float far *wtptr[NMXHLR+1];
float  far *outptr[NMXHLR+2];
float  far *errptr[NMXHLR+2];
float far *delw[NMXHLR+1];
float  far ep[NMXINP];
int long nunit[NMXHLR+2],nhlayer,ninput,ninattr,noutattr,sd;
long i;
```

```
int long  result,errorcode;
 int long cnt_num,cnt,eski;
int long nsnew,nsold;
char task_name[20],dr;
FILE *fp1,*fp2,*fp3,*fopen(),*fpf,*foutf1,*foutf2;
int long fplot10;
long randseed=568731L;
char def[30],def1[20];
char *sec[4]
={"1- DOSYALARI OLUŞTURMA", "2-EĞİTİM VE DENEME İŞLEMİ ", "3-ÇIKIŞ
GRAFİĞİ", "4-ÇIKIŞ"};
void *p;
char atama1[10],atama2[10],atama3[10];
int long hih,hih1,hih2,hih3,fft,d1,d2;
float cn,cn1[10000],enbu=0.0,enku=100.0;
double fe,fe1,de,de1,de2,se,cnt1,fft1;
float ere;
float huge *target;
float huge *input;
float huge *outpt;
                void  yer(void){

input=(float huge *)farcalloc(27500,sizeof(float));
/* float far target[NMXINP][NMXOATTR];*/
outpt=(float huge *)farcalloc(27500,sizeof(float));
/*float far  input[NMXINP][NMXIATTR];   */
target=(float huge *)farcalloc(27500,sizeof(float));
/*float far outpt[NMXINP][NMXOATTR];*/
}

void gitlan(){ clrscr();printf("Performed Iteration=%ld",cnt);
  printf("\n This Time Error    =%f\n",err_curr);
  printf("Çıkmak istiyor musunuz? 'd(dur)- herhangi bir tuş'a basın");
  i=getch();if(getch()=='d')cnt=cnt_num-1;clrscr();printf ("\n Performs iterations PLEASE
DON'T TOUCH!!!\n If you want to use the computer:\n1) please press a button then
wait\n2) appears a table that contains error term\n (it takes approximately 1 minute)\n3)
after that please press 'd' button\n again please wait a minute\n4) then you see a message
that reads 'please press a button' \n please press a button after that wait a minute. Thanks a
lot for doing these steps.");

 }
int random()
 {
 randseed=15625L*randseed+22221L;
 return((randseed>>16)&0x7FFF);
 }
init()
 {
```

```
 int long len1,len2,i,k;
 float huge *p1,huge *p2,huge *p3,huge *p4;
 len1=len2=0;
 nunit[nhlayer+2]=0;
 for(i=0;i<(nhlayer+2);i++){
 len1+=(nunit[i]+1)*nunit[i+1];
 len2+=nunit[i]+1;
 }
 p1=(float huge *)farcalloc(40000,sizeof(float));
 p2=(float huge *) farcalloc(len2+1,sizeof(float));
 p3=(float huge *) farcalloc(len2+1,sizeof(float));
 p4=(float huge *) farcalloc(40000,sizeof(float));
 wtptr[0]=p1;
 outptr[0]=p2;
 errptr[0]=p3;
 delw[0]=p4;
 for (i=1;i<(nhlayer+1);i++){
 wtptr[i]=wtptr[i-1]+nunit[i]*(nunit[i-1]+1);
 delw[i]=delw[i-1]+nunit[i]*(nunit[i-1]+1);
 }
  for (i=1;i<(nhlayer+2);i++){
 outptr[i]=outptr[i-1]+nunit[i-1]+1;
 errptr[i]=errptr[i-1]+nunit[i-1]+1;
 }
 for (i=0;i<nhlayer+1;i++){
 *(outptr[i]+nunit[i])=1.0;
 }
 return(0);
 }

initwt()
 {
 int long  i,j ;
 for (j=0;j<nhlayer+1;j++)
 for (i=0;i<(nunit[j]+1)*nunit[j+1];i++){
          *(wtptr[j]+i)=random()/pow(2.0,15.0)-0.5;
   *(delw[j]+i)=0.0;
  }
  return(0);
  }
set_up()
 {
 int long i;
 eta=0.9;
 printf("\n Monentum rate eta (default=0.9)?: ");
 scanf("%f",&eta);
 alpha=0.7;
 printf("\n Learning rate alfa (default=0.7)?: ");
```

```
scanf("%f",&alpha);
maxe=0.0000001;maxep=0.0000001;
printf("\nMax total error (default=0.00001)?: ");
scanf("%f",&maxe);
printf("\nMax individual error(default=0.00001)?: ");
scanf("%f",&maxep);
cnt_num=1000;
printf("\nMax number of iteration  (default=1000)?: ");
scanf("%ld",&cnt_num);
printf("\nNumber of hidden layers?: ");
scanf("%ld",&nhlayer);
        for(i=0;i<nhlayer;i++) {
        printf("\n\tNumber of units for hidden layer %ld?:",
        i+1);
        scanf("%ld",&nunit[i+1]);
  }
  printf("\nCreate error file ? If so type 1, or type 0 : ");
        scanf("%ld",&fplot10);clrscr();
  printf("\n Performed iterations PLEASE DON'T TOUCH!!!\n If you want to use the
computer:\n1) please press a button then wait\n2) appears a table that contains error term\n
(it takes approximately 1 minute)\n3) after that please press 'd' button\n again please wait a
minute\n4) then you see a message that reads 'please press a button' \n please press a button
after that wait a minute. Thanks a lot for doing these steps.");
  nunit[nhlayer+1]=noutattr;
  nunit[0]=ninattr;
  return(0);
  }
dread(char *taskname)

  {
   int long i,j,c;
   char var_file_name[20];
         strcpy(var_file_name,taskname);
   strcat(var_file_name,"_v.dat");
   if (( fp1=fopen(var_file_name,"r"))==NULL)
   {
    perror("\n Cannot open data file ");
    exit(0);
    }

fscanf(fp1,"%lu%lu%lu%f%f%lu%lu\n",&ninput,&noutattr,&ninattr,&eta,&alpha,&nhlay
er,&cnt_num);
    for(i=0; i<nhlayer+2; i++)
     fscanf(fp1,"%ld",&nunit[i]);
    if ((c=fclose(fp1))!=0)
     printf("\nFile cannot be closedSSSS %ld ",c);
     return(0);
                }
```

```
wtread(char *taskname)
          {
        int long  i,j,c ;
        char wt_file_name[20];
        strcpy(wt_file_name,taskname);
        strcat(wt_file_name,"_w.dat");
        if ((fp2=fopen(wt_file_name,"r"))==NULL)
        {
         perror("\n Cannot open data file");
         exit(0);
         }
         for (i=0; i<nhlayer+1;i++){
           for (j=0; j<(nunit[i]+1)*nunit[i+1];j++) {
                        fscanf (fp2,"%f",(wtptr[i]+j));
            }
                }
          if ((c=fclose(fp2))!=0)
            printf("\nFile cannot be closedSSS %ld",c);
          return(0);
          }
dwrite(char *taskname)
          {
           int long  i,j,c;
          char var_file_name[20];
        strcpy(var_file_name,taskname);
        strcat(var_file_name,"_v.dat");
        if ((fp1=fopen(var_file_name,"w+"))==NULL)
        {
         perror("\n Cannot open data file");
         exit(0);
         }
        fprintf(fp1,"%lu%lu%lu%f%f%lu%lu\n",ninput,noutattr,ninattr,eta,alpha,nhlayer,c
nt_num);
    for (i=0; i<nhlayer+2;i++){
           fprintf(fp1,"%ld ",nunit[i]);
          }

         fprintf(fp1,"\n");
         for (i=0; i<ninput;i++){
           for (j=(i*noutattr); j<((i+1)*noutattr);j++)
                        fprintf (fp1,"%f     ",outpt[j]);
            fprintf (fp1,"\n");
           }
          if ((c=fclose(fp1))!=0)
            printf("\nFile cannot be closedw %ld",c);
          return(0);
          }
wtwrite( char *taskname)
```

```
        {
         int long  i,j,c,k;
         char wt_file_name[20];
       strcpy(wt_file_name,taskname);
       strcat(wt_file_name,"_w.dat");
       if ((fp2=fopen(wt_file_name,"w+"))==NULL)
       {
        perror("\n Cannot open data file");
        exit(0);
        }
        k=0;
        for (i=0; i<nhlayer+1;i++)
              for                                          (j=(i*((nunit[i]+1)*nunit[i+1]));
j<((i+1)*((nunit[i]+1)*nunit[i+1]));j++){
           if(k==8) {
           k=0;
           fprintf (fp2,"\n");
           }
              fprintf (fp2,"%f ",*(wtptr[i]+j%((nunit[i]+1)*nunit[i+1])));
              k++;
        }
        if ((c=fclose(fp2))!=0)
               printf("\nFile cannot be closedSS %ld",c);
        return(0);
        }
void forward(i)
        {

         int long m,n,p,offset;
        float net;

        for (m=(i*ninattr);m<((i+1)*ninattr);m++)
                      *(outptr[0]+(m%ninattr))=input[m];
          for (m=1; m<nhlayer+1;m++){
          for (n=0; n<nunit[m];n++){

          net=0.0;
               for (p=0; p<nunit[m-1]+1;p++){
               offset=(nunit[m-1]+1)*n+p;
              net += *(wtptr[m-1]+offset)
       *(*(outptr[m-1]+p));
              }
       *(outptr[m]+n)= tanh(beta*net);

            }
            }
      m=nhlayer+1;
for (n=0; n<nunit[m];n++){
```

```
    net=0.0;
     for (p=0; p<nunit[m-1]+1;p++){
     offset=(nunit[m-1]+1)*n+p;
     net += *(wtptr[m-1]+offset)
*(*(outptr[m-1]+p));
                }
*(outptr[m]+n)= net;

     }

for(n=((i)*nunit[nhlayer+1]); n<((i+1)*nunit[nhlayer+1]);n++)
     outpt[n]=*(outptr[nhlayer+1]+(n%nunit[nhlayer+1]));
     }

int long introspective (int long nfrom,int long nto)
     {
     int long i,flag;
     if (cnt>=cnt_num) return(FEXIT);
     nsnew=0;
     flag=1;
     for (i=nfrom;(i<nto)&&(flag==1);i++){
     if (ep[i]<=maxep)nsnew++;
     else flag=0;
     }
     if (flag==1) return(SEXIT);
     if (err_curr<=maxe) return(SEXIT);
     return(CONTNE);
     }
int long rumelhart(int long from_snum,int long to_snum)
     {
     int long i,j,k,m,n,p,offset,index;
     float out;
     char err_file_name[20],sonuc[20];
     strcpy(err_file_name,task_name);
     strcat(err_file_name,".err");
     nsold=0;
     cnt=0;
     result=CONTNE;
     if ((fp3=fopen(err_file_name,"w"))==NULL)
     {
     perror("Cannot open error file");
     exit(0);
     }
     do {
     err_curr=0.0;
     for (i=from_snum;i<to_snum;i++){
     forward(i);
```

```
    for(m=(i*nunit[nhlayer+1]);m<((i+1)*nunit[nhlayer+1]);m++)
    {
                out=*(outptr[nhlayer+1]+(m%nunit[nhlayer+1]));
    *(errptr[nhlayer+1]+(m%nunit[nhlayer+1]))=(target[m]-out)*lamda
            +(1-lamda)*tanh(beta*(target[m]-out));
          }

  for (m=nhlayer+1;m>=1;m--){
  for (n=0; n<nunit[m-1]+1;n++){
          *(errptr[m-1]+n)=0.0;
   for (p=0; p<nunit[m];p++){
        offset=(nunit[m-1]+1)*p+n;
       *(delw[m-1]+offset)=eta*(*(errptr[m]+p))
      *(*(outptr[m-1]+n));

      *(errptr[m-1]+n)+=*(errptr[m]+p)
      *(*(wtptr[m-1]+offset));
      }
      *(errptr[m-1]+n)=*(errptr[m-1]+n)*
(1-*(outptr[m-1]+n)*(*(outptr[m-1]+n)));
}
}
    for (m=1;m<nhlayer+2;m++){
for(n=0;n<nunit[m];n++){
for(p=0;p<nunit[m-1]+1;p++){

    offset=(nunit[m-1]+1)*n+p;
  *(wtptr[m-1]+offset)+=*(delw[m-1]+offset);
   }
   }
   }
ep[i]=0.0;
for(m=(i*nunit[nhlayer+1]);m<((i+1)*nunit[nhlayer+1]);m++){
   ep[i]+=fabs((target[m]-
 *(outptr[nhlayer+1]+(m%nunit[nhlayer+1]))));
}
   err_curr+=ep[i]*ep[i];
   }
   err_curr=0.5*err_curr/ninput;
   lamda=exp(-0.8/(err_curr*err_curr));

  if(kbhit())gitlan();
  fprintf(fp3,"%ld%f\n",cnt,err_curr);
  cnt++;
  result=introspective(from_snum,to_snum);
  }
  while (result==CONTNE);
```

```
  for (i=from_snum; i<to_snum;i++) forward(i);
for(i=0; i<nhlayer+1;i++){
  index=0;
for (j=0;j<nunit[i+1];j++)
  {
   printf("\n\n Weights between unit %ld of layer %ld",  j,i+1);
   printf("and units of layer %ld\n",i);
   for (k=0; k<nunit[i];k++)
    printf ("%f",*(wtptr[i]+index++));
    printf ("\n Threshold of unit %ld of layer%ldis%f", j,i+1,*(wtptr[i]+index++));
    }
    }

    strcpy(sonuc,task_name);
    strcat(sonuc,"y.cns");
      if ((fpf=fopen(sonuc,"w"))==NULL)
    {
    perror("Veri dosyasi açilamiyor");
    exit(0);
      }

        for(i=0;i<ninput;i++){/*\nsample%ldoutput%ld=target%ld=%fi,j%noutattr,j%nout
attr,target[j]*/
        for(j=(i*noutattr);j<((i+1)*noutattr);j++){   Yukarıda yazılan kısmı fprintften
çıkardık
        printf("\n\n  output %ld=%f outpt %ld=%f ", /* set sonuçlarını ekrana bastırma işi*/
        j%noutattr,outpt[j],j%noutattr,*(outptr[nhlayer+1]+(j%noutattr)));

        fprintf(fpf,"%f\n",outpt[j]); }}
            printf("\n\n Total number of iteration is %ld",cnt);
     printf("\n Normalized system error is %f\n\n\n",err_curr);
     fclose(fpf);
     printf("\n Please press a buton for continue");
     getch();
     return(result);
                }
user_session()
     { int long i,j,showdata;              /*  Burada ismi verilen bir */
         char fnam[20],dtype[20];         /*dosyadan belirtilen sayıdaki giriş */
         FILE *fp;                        /* için yine girilen sayıdaki verileri okuyor*/
         printf("\n Start of learning session");
         printf("\n\t Enter the task name : ");
         scanf("%s",task_name);
         printf("\n How many features in input pattern?:");
         scanf("%ld",&ninattr);
         printf("\n How many output units?: ");
         scanf("%ld",&noutattr);
         printf("\n Total number of input samples?: ");
```

```
        scanf("%ld",&ninput);
        /* Burada input[i][j] tarzindaki veri yazmada i 0'dan  ninput-1(giris orneklerinin
toplam sayisi) 'na kadar giris yapilacak*/
        /*Burada input[i][j] tarzindaki veri yazmada j 0'dan  ninattr-1(her numune icin
giris sayisi) 'na kadar giris yapilacak*/

        strcpy(fnam,task_name);
        strcat(fnam,".dat");
        printf("\n Input File name is %s",fnam);
        if ((fp=fopen(fnam,"r"))==NULL)
        {
        printf("\n File %s does not exits", fnam);
        exit(0);
        }

        printf("\n Do you want to look at data just read?");
        printf("\n   Answer yes or no :");
        scanf("%s",dtype);
        showdata=((dtype[0]=='y')||(dtype[0]=='Y'));
                for (i=0;i<ninput;i++){
                  for (j=(i*ninattr);j<((i+1)*ninattr);j++) {
                  fscanf(fp,"%f",&input[j]);
                          if (showdata)printf("%ld  %f\n ",j,input[j]);                    }}

                for (i=0;i<ninput;i++){
                for (j=(i*noutattr);j<((i+1)*noutattr);j++) {

                        fscanf(fp,"%f",&target[j]);
                  if (showdata)printf(" %f  ",target[j]);
                }
                }
                if ((i=fclose(fp))!=0)
        {
        printf("\n File56 cannot be closed %ld",i);
        exit(0);
        }

          return(0);
    }
learning()
    {
    int long result;
    user_session();
    set_up();
    init();
    do {
        initwt();
        result=rumelhart(0,ninput);
```

```
        } while (result==RESTRT);
        if (result==FEXIT)
        {
        printf("\n Max number of iterations reaced,");
        printf("\n but failed to decrease system");
        printf("\n error sufficiently");
        }
        dwrite(task_name);
        wtwrite(task_name);
        return(0);
        }

/* Aşağıdaki fonksiyon önceden belli bir iterasyonla eğitilmiş dosyanın iterasyon sayısını
arttırmak için eklenen kısım */
learning1()
    {
       eski=1;
       int long result;
       int long i,j,showdata;              /*  Burada ismi verilen bir */
        char fnam[20],dtype[20];           /*dosyadan belirtilen sayıdaki giriş */
        FILE *fp;                          /* için yine girilen sayıdaki verileri okuyor*/
        printf("\n Start of learning1 session");
        printf("\n\t Enter the task name : ");
        scanf("%s",task_name);

                    dread(task_name);
                    init();
              strcpy(fnam,task_name);
        strcat(fnam,".dat");
        printf("\n Input File name is %s",fnam);
        if ((fp=fopen(fnam,"r"))==NULL)
        {
        printf("\n File %s does not exits", fnam);
        exit(0);
        }

        printf("\n Do you want to look at data just read?");
        printf("\n   Answer yes or no :");
        scanf("%s",dtype);
        showdata=((dtype[0]=='y')||(dtype[0]=='Y'));
         for (i=0;i<ninput;i++){
                 for (j=(i*ninattr);j<((i+1)*ninattr);j++) {
                 fscanf(fp,"%f",&input[j]);
                 if (showdata)printf("%f  ",input[j]);
                 }}

                for (i=0;i<ninput;i++){
                for (j=(i*noutattr);j<((i+1)*noutattr);j++) {
```

```
                        fscanf(fp,"%f",&ere);
        target[j]=ere/2.0;
                    if (showdata)printf(" %f\n",target[j]);
                   }
          }
          if ((i=fclose(fp))!=0)
         {
          printf("\n File56 cannot be closed %ld",i);
          exit(0);
          }

 printf("\nMax number of iteration  (default=1000)?: ");
 scanf("%ld",&cnt_num);
     printf("\nCreate error file ? If so type 1, or type 0 : ");
  scanf("%ld",&fplot10);clrscr();
         printf("\nPerforms iterations PLEASE DON'T TOUCH!!!");
  nunit[nhlayer+1]=noutattr;
  nunit[0]=ninattr;
             do {

          wtread(task_name);
          result=rumelhart(0,ninput);
          } while (result==RESTRT);
          if (result==FEXIT)
          {
          printf("\n Max number of iterations reaced,");
          printf("\n but failed to decrease system");
          printf("\n error sufficiently");
          }
          dwrite(task_name);
          wtwrite(task_name);
          return(0);
          }

output_generation()
          {
          int long i,m,nsample;
          char ans[10];
          char dfile[20];
          printf("\n Generation of outputs for a new pattern");
          printf("\n\t Present task name is %s", task_name);
          printf("\n\t Work on a different task?");
          printf("\n\t Answer yes or no :");
          scanf("%s",ans);
                  if ((ans[0]=='y')||(ans[0]=='Y'))
          {
           printf("\n\t Type the task name :");
           scanf("%s",task_name);
```

```
        dread(task_name);
        init();
        wtread(task_name);
        }
printf("\n Enter file name for patterns to ");
printf(" be processed: ");
scanf("%s",dfile);
if ((fp1=fopen(dfile,"r"))==NULL)
{
perror("Connat open dfile");
exit(0);
}


                /* printf("\n Enter number of patterns for processing :");
                 scanf("%ld",&nsample);
                 for (i=0;i<nsample;i++)
                        for (m=(i*ninattr); m<((i+1)*ninattr);m++)
                { printf("%ld veri=",((m%ninattr)+1));
                 scanf("%f",&input[m]);  }
                 tek tek veri girme kısmı*/

   /* *.dat veri dosyasından(dosya1) istenilen sayıdaki set girişi okuyup
 sonucu(YSA sonucunu yani) *ht.dat dosyasına kaydediyor
nsample=1 sat□r□ndan for (i=0; i<nsample;i++) satırına kadar devam ediyor.*/
        nsample=1 /*aslında bu gerekmiyor sanki bu test sayısı daha sonra isteniyor*/;
   char dosya1[20],dosya2[20],cont[20];
   float ss;
   printf("veri dosyasının ismini giriniz:");
   scanf ("%s",dosya1);
   strcpy(dosya2,task_name);
   strcat(dosya2,"ht .dat");
   printf("%s\n",dosya2);
   strcat(dosya1,".dat");
   printf("%s\n",dosya1);

if((foutf1=fopen(dosya1,"r"))==NULL)
  {
perror("Connat open dfile");
exit(0);
}
printf("\n Enter number of patterns for processing:\n (test dosyası set sayısı yani) ");
scanf("%ld",&nsample);
/*test datalarının bulunduğu dosyadaki set sayısı*/

 for (i=0;i<nsample;i++){
       for (m=(i*ninattr); m<((i+1)*ninattr);m++)
  {
```

```
 fscanf(foutf1,"%f",&input[m]);

        }               }

 fclose(foutf1);
 if((foutf2=fopen(dosya2,"w+"))==NULL)
                {
 perror("Connat open dfile");
 exit(0);
 }


 for (i=0; i<nsample;i++)
 {
          forward(i);
   for(m=(i*noutattr);m<((i+1)*noutattr);m++){
printf("\nsample%ldoutput%ld=%f",
i,(m%noutattr),ss=*(outptr[nhlayer+1]+(m%noutattr)));
printf("\n");
fprintf(foutf2,"%f\n",   /*Çıkış hatasını hata.dat dosyasına kopyalıyor*/
                *(outptr[nhlayer+1]+(m%noutattr)) );
printf("%f\n",outpt[m],   /*Çıkış hatasını hata.dat dosyasına kopyalıyor*/
                *(outptr[nhlayer+1]+(m%noutattr)) );
          }
     }
          if ((i=fclose(foutf2))!=0)
     printf("\nFile cannot be closedS %ld",i);
                  printf("\nOutput have been generated ");
     return(0);
                  }

void der()
{
 FILE *f;
 if ((f=fopen("veri2.dat","r"))==NULL)
        {
  perror("Cannot open err_file");
  exit(0);
 }
 i=1;
 while(!(feof(f)))
 {
  fscanf(f," %ld %f\n",&cnt1,&cn);
  i++;
 }
  fclose(f);
  fft=i;
  se=550.0;
```

```
  de=fft/se;
  de1=fft/2.0 ;
  de2=fft/4.0  ;
  hih2=floor(de1);
  hih1=floor(de2);

  hih=floor(de);
  if ((f=fopen("veri2.dat","r"))==NULL)
       {
  perror("Cannot open err_file");
  exit(0);
  }
 sd=0;


 for(i=1;i<fft;i++)
       {
  fscanf(f," %ld %f\n",&cnt1,&cn);
  if(enbu<cn)enbu=cn;
  if(enku>cn)enku=cn;
       if(i/(hih+1.0)==floor(i/(hih+1.0))){ sd=sd+1;cn1[sd]=cn*10;}
  }
       itoa(hih1, atama1, 10);
 itoa(hih2, atama2, 10);
      itoa(fft, atama3, 10);

   if ((i=fclose(f))!=0)
  printf("\nFile cannot be closed13 %ld",i);
}

int long  main1()
{   char select[20],cont[20];
    strcpy(task_name,"*********");
do { printf("Be careful Sir! max input #=%d max output #=%d\n max set
#=%d",NMXIATTR,NMXOATTR,NMXINP);
    printf("\n*Select L(earning) or O(utput generatoin)\n or 1(continue from old weights
file)*\n");
    do {

            scanf ("%s",select);
        switch(select[0]){
        case 'o':
        case 'O':
          output_generation();
          break;
          case 'l':
          case 'L':
              learning();
```

```
        break;
           case '1':
           learning1();
           break;
           default:
            printf("\n answer learning(l) or output generation(o)\n or 1(continue from old
weights file)");
   break;
    }
        }
           while ((select[0]!='o')&&(select[0]!='O')
           &&(select[0]!='l')&&(select[0]!='L')&&(select[0]!='1'));
           printf("\nDo you want to continue?");
           scanf("%s",cont);
           }

        while ((cont[0]=='y')||(cont[0]=='Y'));
           printf("\nIt is all finished. ");
           printf("\n Good bye");

                return(0);
}
int long main(void)
{
yer();
main1();
return 0;
}
```

EK 5 AAD Üzerinden Verilerin Alınmasını Sağlayan Program

```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>
#include <time.h>
#include <dos.h>
#include <bios.h>
#include <graphics.h>
#include <conio.h>
#include "dskdef.h"

# define  MAXSCREENLEN 700
# define  period 0.2
 int d=0;

struct parameter {
            unsigned int  pgm_cnt;
            unsigned int  entrypoint;
            unsigned int  inverse;
            unsigned int  com;
            unsigned long speed;
           } prm;
const char* programme ="Digital oscilloscope";
const char* pgmdate   ="15.07.95";
const char* version   ="3.00";
const char* copyright ="Copyright (c) 1995 Texas Instruments";
FILE  *stream,*ste;


unsigned int      pcom;

void WaitFor(unsigned int what)
{
 int j=10000,i=j;

 while(!(inportb(pcom+LSR) & what) && i--);

 if(i<1){
   fprintf(stdout,"Handshake error"
            "\r\nWait loop finished in WaitFor() with start=%d",j);
   exit(1);
 }
 else return;
}
```

```
void InitPort(void)
{
 unsigned int port_no;
 unsigned int  COMADD[]={0x3f8, 0x2f8, 0x3e8, 0x2e8};
 int   BRD=115200l/prm.speed;
 pcom = COMADD[prm.com];
 port_no=pcom;

 asm  mov  DX,word ptr port_no
 asm  add  DX,3
 asm  mov  AL,087h
 asm  out  DX,AL
 asm  sub  DX,3
 asm  mov  AX,word ptr BRD
 asm  out  DX,AL
 asm  add  DX,1
 asm  mov  AL,byte ptr BRD
 asm  xchg AL,AH
 asm  out  DX,AL
 asm  sub  DX,1
 asm  add  DX,3
 asm  mov  AL,07h
 asm  out  DX,AL
 asm  out  DX,AL
 asm  mov  DX, word ptr port_no
 asm  add  DX,5
xempty0:
 asm  in   AL,DX
 asm  and  AL,060h
 asm  cmp  AL,060h
 asm  jne  xempty0

}

void reset50(void)
{
 unsigned int  port_no=pcom;
 if(prm.inverse) {
   asm  mov  DX,word ptr port_no
   asm  add  DX,4
   asm  mov  AL,0xB
   asm  out  DX,AL
   delay(1);
   asm  mov  DX,word ptr port_no
   asm  add  DX,4
   asm  mov  AL,0xA
   asm  out  DX,AL
   delay(1);
```

```
    asm  mov  DX,word ptr port_no
    asm  add  DX,4
    asm  mov  AL,0xB
    asm  out  DX,AL
    delay(1);
  }
  else  {
    asm  mov  DX,word ptr port_no
    asm  add  DX,4
    asm  mov  AL,0xA
    asm  out  DX,AL
    delay(1);
    asm  mov  DX,word ptr port_no
    asm  add  DX,4
    asm  mov  AL,0xB
    asm  out  DX,AL
    delay(1);
    asm  mov  DX,word ptr port_no
    asm  add  DX,4
    asm  mov  AL,0xA
    asm  out  DX,AL
    delay(1);
  }
}

void BaudRateDetect(void)
{
 reset50();
 delay(12);

 if(prm.speed<57600){
    while(!(inportb(pcom+LSR) & XMT_BUF_EMPTY));
    outportb(pcom,0x80);
 }
 else
    outportb(pcom,0x80);
}

unsigned int InitMonitor(void)
{
  int  i=10000;
  int  ans=inportb(pcom+LSR);
  int  byte_return;
  unsigned int send;

  while(!(ans & DATA_READY) && i){
        ans = inportb(pcom+LSR);
        i--;
```

```
 }
 if(i<1)
   return  0;
 if( inportb(pcom)!=KB_ESC)
    return 2;

 delay(1);
 send=0xAA;
 WaitFor(XMT_BUF_EMPTY);
 outportb(pcom,send);
 WaitFor(DATA_READY);
 delay(1);
 if ((byte_return=inportb(pcom))!=send) {
    printf("\nbyte recieve = %x\n",byte_return);
    return 1;
 }
 WaitFor(XMT_BUF_EMPTY);
 outportb(pcom,(send=KB_ESC));
 WaitFor(DATA_READY);
 delay(1);
 if ((byte_return=inportb(pcom))==send)
    return 3;
 return 1;
}

void initializemonitor(void)
{
 unsigned int num;
 unsigned int error;
 int pass=FALSE;


 InitPort();

 pass=FALSE;
 BaudRateDetect();
 delay(2);

 switch(error=InitMonitor())
 {
        case 0:
        case 1:
        case 2:  pass=FALSE;
                 break;
        case 3:  pass=TRUE;
                 break;

 }
```

```
if(!pass){
  clrscr();
  printf("\nCommunication not successful\n");
  switch(error) {
    case 0: printf("\nNo Response.\n");
            break;
    case 1: printf("\nTest Error.\n");
            break;
    case 2: printf("\nNo Escape.\n");
            break;
   default: break;
  }
  exit(1);
}
}

void sendbyte(unsigned int send)
{
 unsigned int receive;
 WaitFor(XMT_BUF_EMPTY);

 outportb(pcom,send);

 WaitFor(DATA_READY);
 delay(1);
 receive=inportb(pcom);
 if (receive!=send)
    fprintf(stdout,"Handshake error"
                "\r\nSent byte in 'sendbyte()' is not correct!\r\n"
                "Sent    : 0x%02x   Received: 0x%02x\r\n",send,receive);
}

unsigned int sendword(unsigned int send)
{
 unsigned int   s1;
 unsigned int   receive;

 WaitFor(XMT_BUF_EMPTY);
 s1 = (send>>8)&0xff;
 outportb(pcom,s1);

 WaitFor(DATA_READY);
 delay(1);
 receive=inportb(pcom);

 WaitFor(XMT_BUF_EMPTY);
```

```
s1 = send&0xff;
outportb(pcom,s1);

WaitFor(DATA_READY);
delay(1);
if((receive=(receive<<8)+inportb(pcom))!=send)
  fprintf(stdout,"Handshake error"
                 "\r\nSent word in sendword() is not correct!\r\n"
                 "Sent    : %04xh   Received: %04xh\r\n",send,receive);

return receive;
}

unsigned int GetDskAdd(char* pbuf)
{
unsigned int x=0,i=4;
char c=*pbuf++;
for(;i;i--,c=*pbuf++)
  {
   x =(x<<4) + c;
   if(isdigit(c))
     x -= '0';
   else
     x -= 'A'-10;
  }
return x;
}

unsigned int GetDskData(char* pbuf,unsigned int *data)
{
unsigned int x,num=0,i;
unsigned char c;

while(*(pbuf++)!='7')
  {
   for(*data=0,i=4;i;i--)
     {
      c = *pbuf++;
      *data<<=4;
      if(isdigit(c))
          *data +=c-'0';
      else
          *data +=c-'A'+10;
     }
    data++;
    num++;
  }
```

```
 return num;
}

void sendcommand(const cmd,unsigned int startaddress,unsigned int length)
{
 sendbyte(cmd);
 sendword(startaddress);
 sendword(length-1);
}

double LoadDsk(void)
{
 char   linebuf[MAXLINE], *pbuf=linebuf;
 unsigned int   line=0;
 unsigned int   data[MAXLINE];
 unsigned int   numdata=0;
 unsigned long  sumdata=0;
 unsigned int   address;
 unsigned int   program;
 double  d;

 clock_t start=clock();
 while(1) {
   pbuf=fgets(linebuf,MAXLINE,stream);

   switch(*pbuf){
        case ':':
        case NULL: fclose(stream);
                   fprintf(stdout,"\nLoading of prog/data finished");
                   d=clock()-start;
                   return d>0 ? d : 0.001;
        case 'K': numdata=0;break;
        case '9': address=GetDskAdd(pbuf+1);
                   switch(*(pbuf+5))
                    {
                     case '7': break;
                     case 'M': program=no; break;
                     case 'B': program=yes; break;
                     default : fprintf(stdout,"\n\nerror 1: 'Wrong Tag "
                                    "in Pos 6'!\n\n\rPlease check whether"
                               " the file is a valid *.dsk file!\n\n\r");
                               return 0;
                    }
                   numdata=GetDskData(pbuf+5,data);
                   sumdata +=numdata;
                   break;
        case '1': address=GetDskAdd(pbuf+1);
                   if(address==0)
```

```
                    break;

                    prm.pgm_cnt=prm.entrypoint=address;
                    break;
          default  : fprintf(stdout,"\n\nerror 2: 'Wrong Tag in Pos 1' in LoadDsk()"
                              "!\n\n\rPlease check whether"
                              " the file is a valid *.dsk file!\n\n\r");
                    return 0;
       }

     if(numdata){
          unsigned int i,cmd=(program==yes) ? LP:LD;

          fprintf(stdout,"\r   %4d %5d %8ld  %s",
                    ++line,numdata,sumdata,program ? "program":"data");
          sendcommand(cmd,address,numdata);
          for(i=0;i<numdata;i++) sendword(data[i]);
     }
  }
}

void ExePgm(void)
{
 sendbyte(XG); sendword(prm.pgm_cnt);
 outportb(pcom,NULL);
 return;
}

unsigned char Bcrcv(int port_no)
{
  char c;
  asm  mov  DX,word ptr port_no
  asm  add  DX,5
  asm  mov  BX,4000
chkst:
  asm  sub  bx,1
  asm  jz   timerr
  asm  in   AL,DX
  asm  and  AL,061h
  asm  cmp  AL,061h
  asm  jne  chkst
  asm  sub  DX,5
  asm  in   AL,DX
  asm  mov  byte ptr c,AL
  return(c);
timerr:
  return(-1);
}
```

```
void checkkey(void);

char sa_kernel_out[]={"osc.dsk"};
int  restart = 0;
clock_t duration;

char  info[300];
char  title[200];

int maxsample = 500;
int threshold = 1;
int sample_rate = 50;
int sample;
int rstart =no;
int srate,osrate;
int first;
int start,ostart;

char buffer[501];
char buffer2[501];
char bit_rate    = 0xFF;
double rate,orate;
float time_base = 1.0;

void setup_vals(void)
{
  setwritemode(0);
  setviewport(0,300,200,309,1);
  clearviewport();
  setviewport(100,0,620,30,1);
  setcolor(14);
  settextstyle(0,0,2);
  outtextxy(80,10,"DSK OSCILLOSCOPE");
  setviewport(0,30,620,330,1);
  setcolor(11);
  settextstyle(0,0,0);
  outtextxy(10,7,  "      5V ");
  outtextxy(10,33, "      4V ");
  outtextxy(10,59, "      3V ");
  outtextxy(10,72, "  A     ");
  outtextxy(10,85, "  M   2V ");
  outtextxy(10,98, "  P     ");
  outtextxy(10,111,"  L   1V ");
  outtextxy(10,124,"  I     ");
  outtextxy(10,137,"  T   0V ");
  outtextxy(10,150,"  U     ");
  outtextxy(10,163,"  D  -1V ");
  outtextxy(10,176,"  E     ");
```

```
 outtextxy(10,189,"     -2V ");
 outtextxy(10,215,"     -3V ");
 outtextxy(10,241,"     -4V ");
 outtextxy(10,267,"     -5V ");

 outtextxy(100,290,"(Q)uit to DOS   ");
}

void start_display(void)
{
 /* int gdriver = EGA, gmode = EGAHI, errorcode, Y, X;
  initgraph(&gdriver, &gmode, "");
  errorcode = graphresult();
  if (errorcode != grOk)
  {
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);
  } */
 /* clearviewport();
  setup_vals();
  setviewport(100,40,600,300,1);
  setcolor(2);
  for(Y=0;Y<=260;Y+=26) line(0,Y,500,Y);
  for(X=0;X<=500;X+=50) line(X,0,X,260);
  setwritemode(1);
  setcolor(15); */
}

int main(int argc, char *argv[])
{  char task_name[20],fnam[20];
  char displaybuff[50];
  int TA;
  int two_block_length,block_length,address;
  int k,f,c, number_overrun,BRD;
  float SCF, FDAC;

  if(argc > 3){
    printf("Please enter the command as 'oscope -cx'\n");
    printf("  where x is the comport that DSK connect to.\n");
    printf("  (The default value is 1 for comport 1)\n");
    exit(0);
  }
  else{
    strlwr(argv[1]);
    if(argv[1]){
        if(strstr(argv[1],"-c"))
```

```
            prm.com = argv[1][2]-'1';
          else{
            printf("Please enter the command as 'oscope -cx'\n");
            printf("  where x is the comport that DSK connect to.\n");
            printf("  (The default is 1 for comport 1)\n");
            exit(0);
             }
               }
      }
  clrscr();
  sprintf(title,"\n%s - Version %s / %s "
                  "\n%s\n",programme,version,pgmdate,copyright);
  printf(title);
  prm.entrypoint=prm.pgm_cnt=0x0a00;
  prm.inverse   =yes;
re_start:
  maxsample = 500;
  threshold = 1;
  rstart =no;
  bit_rate = 0xFF;
  sample_rate = 50;
  prm.speed=57600;
  initializemonitor();

  printf("\nDownloading oscope code...\n");
  stream=fopen(sa_kernel_out,"rb");
  duration = LoadDsk();
  scanf("Kay□t Dosya ~smi:%s",task_name);
  strcpy(fnam,task_name);
  strcat(fnam,".dat");

   ste=fopen(fnam,"w");
  if (duration == 0) exit(0);
  ExePgm();

  prm.speed=115200l;
  delay(50);
  InitPort();
  start_display();
  delay(50);
  first = no;
  start = 0;
  ostart =0;

 /* setcolor(LIGHTRED);
  for(f=start;f<=(start+sample_rate);f++)
  {
    line((f-start)*(rate),130,(f-start)*rate,130-buffer[f]);
```

```
}
setcolor(WHITE);
for(f=start;f<=(start+49);f++)
{
 line((f-start)*10,130-buffer[f],((f+1)-start)*10,130-buffer[f+1]);
}*/
clrscr();
gotoxy(1,24);
cprintf("ESC (€□k□Ÿ)");
gotoxy(17,24);
cprintf("'Y' (Kaydet)");
gotoxy(35,24);
cprintf("'N' (Kayd□ Durdur)");
gotoxy(10,12);
cprintf("Ha...ha... Kopyal□yooom");

for (;;)
{
 // while (kbhit()) checkkey();
  if (rstart == yes)
  {
    rstart = no;
    goto re_start;
  }
clock_t start, end;
 // start = clock();

        asm cli;

  for(f=0;f<=maxsample;f++)
   { buffer[f] = (Bcrcv(pcom))*1.35;// twiddle factor to get correct gain
    if(d==1)fprintf(ste,"%d \n", buffer[f]);
     gotoxy(34,12);
    if(d==1)cprintf("%d",buffer[f]);
    while (kbhit()) checkkey();
    }

    asm sti; //herhangi bir interrup istegini serbes b□rak□r
//    end = clock();
//   fprintf(ste,"%f   \n", (end-start)/CLK_TCK);
/*  first = no;
  ostart = start;
  start = 0;
  for (f=0;f<=maxsample;f++)
  {
   if ( buffer[f] < threshold) first = yes;
   if ((buffer[f] >= threshold)&&(first==yes))
   {
```

```
        first = no;
        start = f;
        break;
    }
  }
if (f>(maxsample-srate)) start =0;
for (f=0;f<maxsample;f++)
{
 buffer[f] = (buffer[f]&bit_rate);
 if (bit_rate ==  -128)
      buffer[f] = buffer[f] +64;
 if (bit_rate ==  -64)
      buffer[f] = buffer[f] +32;
 if (bit_rate ==  -32)
      buffer[f] = buffer[f] +16;
 if (bit_rate ==  -16)
      buffer[f] = buffer[f] +8;
 if (bit_rate ==  -8)
      buffer[f] = buffer[f] +4;
 if (bit_rate ==  -4)
      buffer[f] = buffer[f] +2;
 if (bit_rate ==  -2)
      buffer[f] = buffer[f] +1;
}

srate = (10.0 * time_base)/period;
rate = (double)maxsample/srate;

if (osrate==srate)
{
 setcolor(LIGHTRED);
 for(f=0;f<=srate;f++)
 {
      setcolor(LIGHTRED);
      line(f*rate,130,(f*rate),130-buffer2[ostart+f]);
      line(f*rate,130,(f*rate),130-buffer[start+f]);
      if (f<=srate-1)
      {
       setcolor(WHITE);
       line(f*rate,130-buffer2[ostart+f],(f+1)*rate,130-buffer2[ostart+f+1]);
       line(f*rate,130-buffer[start+f],(f+1)*rate,130-buffer[start+f+1]);
      }
 }
}
else
{
 setcolor(LIGHTRED);
 for(f=ostart;f<=(ostart+osrate);f++)
```

```
    {
        line((f-ostart)*orate,130,(f-ostart)*orate,130-(buffer2[f]));
    }
    setcolor(WHITE);
    for(f=ostart;f<=(ostart+osrate-1);f++)
    {
        line((f-ostart)*orate,130-buffer2[f],((f+1)-ostart)*orate,
                                130-buffer2[f+1]);
    }
    setcolor(LIGHTRED);
    for(f=start;f<=(start+srate);f++)
    {
        line((f-start)*rate,130,(f-start)*rate,130-(buffer[f]));
    }
    setcolor(WHITE);
    for(f=start;f<=(start+srate-1);f++)
    {
        line((f-start)*rate,130-buffer[f],((f+1)-start)*rate,130-buffer[f+1]);
    }
   }

   osrate = srate;
   orate  = rate;
   ostart = start;
   for (f=0;f <maxsample;f++)
    buffer2[f] = buffer[f];
   sprintf(displaybuff,"Timebase %.1f ms/div",time_base);
   setviewport(400,305,600,315,1);
   clearviewport();
   setcolor(11);
   outtextxy(0,0,displaybuff);
   setviewport(100,40,600,300,1);*/
  }
}

void checkkey(void)
{
 int key, f;
 key = bioskey(0);
 if(key==5497)d=1;
 if(key==12654)d=0;
 if(key==283)exit(0);
 }
```

EK 6 DSP Sistem Kontrol Programı

```
*************************************************************************
* BU PROGRAM ALT KONTROL DÖNGÜLERİ YARDIMIYLA MAKSİMUM GÜÇ FAKTÖRÜ YÖNTEMİNİ  KULLANARAK ASENKRON MOTORUN VEKTÖR VE AKIM KONTROLÜNÜ YAPMAKTADIR. *
*                                                                       *
*************************************************************************
*
        .version  50
        .title "MOTOR CONTROL"
        .mmregs
        .global VECTORS, INIT, INIT1, INIT3, DIVISION, NEWDATA, REFSPD
        .ref TBLCOS, TBCEND, TBLSEC, TBSEND, SPDTAB, SPDEND, PREND
        .ref L110A, L110B, L310A, L310B, L605A, L605B, L1605A, L1605B, L7320A, L7320B
        .ref KP, KI, IDSMAX, IQSMAX, IS, REFSPD, BASESPD, WSL, WR, WE, PISIR
*
* SABİTLER VEYA DEĞİŞKENLERİ DEPOLAMAK İÇİN KULLANILAN SABİT HAFIZA HÜCRELERİ *
*
*DEĞİŞKENLER
*
        .sect  "VARIABLES"
*
RSDMEM   .set  00h      ;REFERANS HIZLAR İÇİN HAFIZA
ACTSPD   .set  01h      ;AÇISAL POZİSYON
IQSMEM   .set  02h      ;Q EKSEN AKIMI
IDSMEM   .set  03h      ;D EKSEN AKIMI
ISMEM    .set  04h
ANGK     .set  05h      ;ARCTANGENT DEĞERLERİNİ DEPOLAMA
THETA    .set  06h      ;MOMENT AÇISI
IAREF    .set  07h
ICREF    .set  08h
IBREF    .set  09h
SECMEM   .set  0ah
IAACT    .set  0bh
ICACT    .set  0ch
IBACT    .set  0dh
TRANSEL  .set  0eh
*
* GEÇİCİ DEĞİŞKENLER
*
FRAC     .set  0fh      ;BÖLME İÇİN DOĞRULUK
NUMER    .set  10h      ;HERHANGİ BİR BÖLME İŞLEMİ İÇİN BÖLEN
DENOM    .set  11h      ;HERHANGİ BİR BÖLME İŞLEMİ İÇİN BÖLÜNEN
QUOT     .set  12h      ;HERHANGİ BİR BÖLME İŞLEMİ İÇİN BÖLÜM
ACCH  .set  13h  ;AKÜMÜLATÖRÜN YÜK. AĞIR. BİT'LER. DE GEÇİCİ OLARAK DEPOLAMA
```

```
ACCL .set 14h    ; AKÜMÜLATÖRÜN DÜŞ. AĞIR. BİT'LER. DE GEÇİCİ OLARAK DEPOLAMAK
TEMP0  .set  15h     ;DEĞİŞKENLERİ GEÇİCİ OLARAK DEPOLAMA
TIME   .set  16h
XR0  .set  17h          ;DURUM KAYDEDİCİLER İÇİN MEVCUT DURUMU DEPOLAMA
XR1    .set  18h
ANGK_1  .set  19h
EK     .set  1ah        ;k da HATA DEĞERLERİ
EK_1   .set  1bh
SUMK   .set  1ch     ;İNTEGRAL İŞLEMİNDE KULLANILMAK ÜZERE TOPLAMA
SUMK_1  .set  1dh
TEMPSGN .set  1eh  ;TEST KAYDEDİCİDEKİ BİT'İ İŞARETLE
TEMP1  .set  1fh
TEMP2  .set  20h
TEMP3  .set  21h
TEMP4  .set  22h
*
* PROGRAMLAMA BOYUNCA DEĞİŞEBİLEN SABİTLER
*
* LİNEERLEŞTİRME PARAMETRELERİ
*
        .sect  "LINEAR"
*
L110A   .set   35ch
L110B   .set    30
L310A   .set   218h
L310B   .set   2325
L605A   .set   0d7h
L605B   .set   8980
L1605A  .set    35h
L1605B  .set   15005
L7320A  .set    06h
L7320B  .set   19975
*
* MOTOR PARAMETRELERİ
*
BASESPD  .set    24      ;REV/SC=1450 REV/MIN
STARES   .set    44      ;STATOR DİRENCİ*2^8
RORES    .set    34      ;ROTOR DİRENCİ*2^8
POLE     .set    4
POWBASE  .set    560     ;NOMİNAL GÜÇ
IDSMAX   .set    384     ;SABİT MOMENT D EKSEN AKIMI*2^8
IQSMAX   .set    640     ;SABİT MOMENT Q EKSEN AKIMI*2^8
IS       .set    666     ;SABİT MOMENT STATOR AKIMI*2^8
STAEND   .set    31      ;STATOR ENDÜKTANSI
ROEND    .set    33      ;ROTOR ENDÜKTANSI
MUTEND   .set    814     ;KARŞILIKLI ENDÜKTANS*2^8
MUTENDINV  .set  81      ;KARŞILIKLI ENDÜKTANS'IN TERSİ
```

```
TOR  .set   64              ;ROTOR ZAMAN SABİTİ*2^8
*
* KONTROL SİSTEM PARAMETRELERİ
*
KP      .set    150         ;ORANSAL KAZANÇ
KI      .set    30          ;İNTEGRATÖR KAZANCI
SLOTCLE .set  ss*   ;İKİ ENKODER SLOTU ARASINDAKİ TEMİZLEME (CLEARANCE) SÜRESİ
NUMSLOTS .set  ss-1*  ;ENKODER KULLANILMADIĞI İÇİN BU KISIMLAR *İLE GÖSTERİLİR
*
        .data
*
REFSPD .byte 5,7,10,15,17,20,25,35,40,45,50,60 ;RASGELE SEÇİLEN REFERANS HIZLAR
*                                               ;GERÇEK HIZ=BUNLAR*2*3.14159 IN 3 BIT FR.
*************************************************************************
*
*                   LOOK-UP TABLOSU                     *
*************************************************************************
*
         .sect      "TABLE"
*
*************************************************************************
*
*           PROGRAMDA KULLANILAN KESME VEKTÖRLERİ          *
*************************************************************************
*
        .sect "VECTORS"
RESET   B     INIT
INT1    B     ROTANG
*************************************************************************
*
*     BU BÖLÜM İŞLEMİN KONTROLÜ İÇİN GİRİŞ ŞARTLARI SAĞLAR    *
*************************************************************************
*
        .text
*
MAIN    SPLK    #00h,SUMK_1
        LAR     AR1,#REFSPD
        MAR     *,AR1
        LACC    *,3,AR4
        LAR     AR4,#NUMSLOTS
        SACL    EK_1                ;İLK HATA =REFERANS HIZ
        SACL    RSDMEM
*
        LACL    #00h
        SAMM    PRD         ;CLOKOUT1 PERYODUYLA OPERASYONU YAP
        LACL    #20h
        SAMM    TCR         ;PRD VE RESTART İLE YENİDEN YÜKLE
        SMMR    TIM,TIME    ;TIMER'A GİRİŞ DEĞERİNİ DEPOLA
```

```
SPLK #6624h,TRANSEL ;IGBT 1-2-5'İ TETİKLEYEREK INITIALIZE ET
*************************************************************************
*
*                    ASIL PROGRAM                    *
*************************************************************************
*
AGAIN  CLRC   INTM    ;KESMELERDEKİ MASKELEMEYİ KALDIR
       IDLE           ;BİR POZİSYON KESMESİ GELİNCEYE KADAR BEKLE
       SETC   INTM    ;HESAPLAMALAR YAPILINCAYA KADAR KESMELERİ MASKELE
*************************************************************************
*
*                ROTOR GERÇEK HIZINI BUL                *
*************************************************************************
*
SPEED  LACC   ANGK
*
       SAMM   TREG0    ; *ENCODER ÇIKIŞINI  NORMAL ŞEKLİNE DÖNÜŞTÜR
       MPY    #10Fh    ; #1Fh=360/ss IN 3 BIT KESİRLİ=.0.730
       PAC
       BSAR   6
       SACL   ANGK     ; 3 BIT KESİRLİ İÇİN 12 BIT MANTISSA
*
       LACC   #SPEEDTAB,0
       ADDS   TIME
       TBLR   ACTSPD
*************************************************************************
*
*BU BÖLÜM ALAN ZAYIFLATMA (FIELD WEAKENING) OPERASYONUNU
SAĞLAR *
*      Ids( referans hız>nominal hız>gerçek hız ise)=          *
*      Ids(sabit moment bölgesi)(nominal hız/gerçek hız)       *
*************************************************************************
*
FLDWEAK LACC   RSDMEM
       SUB    #BASESPD,3
       SACB
       SAMM   TREG0
       EXAR
       BCNDD  NORMAL,LT
       LACC   #IDSMAX,3
*
       LACC   #BASESPD,3
       SAMM   TREG0
       MPY    PISIR
       PAC
       BSAR   3
       SACL
       SAMM TREG0
```

```
        MPY     MUTENDINV
        PAC
        BSAR    4
        SACL    IDSMAX
*
        LACC    #0Dh
        SAMM    TREG0
        MPY     IDS
        PAC
        BSAR    4
        SACB
        ADD     TEMP2
        SACL    PISIR
        SACL    TEMP2
        LACL    #0Dh
        SACL    NUMER
        LACL    PISIR
        SACL    DENOM
*
        CALLD   DIVIDE
        SPLK    #03h, FRAC
*
        LACC    QUOT
        SAMM    TREG0
        MPY     IQS
        PAC
        SACL    WSL
        ADD     WR
        SACL    WE
        ADD     TEMP4
        SACL    TEMP3
        ADD     TEMP4
        SACL    THETA
        LACL    TEMP3
        SACL    TEMP4
*
NORMAL  LACB
        SACL    IDSMEM
        SACL    NUMER
*
        LACC    ACTSPD          ;3 BIT KESİRLİ TÜMÜ
        SUB     RSDMEM
        SACL    EK
        ADD     EK_1            ;3 BIT KESİRLİ HATALAR
        SAMM    TREG0
        LACC    SUMK_1          ;3 BIT KESİRLİ İNTEGRALİNİ ÜRET
        MPY     TIME
        APAC                    ;AKÜMÜLASYONU KULLANARAK İNTEGRAL OLUŞTUR
```

```
        BSAR    1           ; (ek+ek_1)*duration/2
        SACL    SUMK
*
        SAMM    TREG0
        MPY     #KI
        PAC
        LT      ACTSPD
        MPY     #KP
        APAC
        SACL    IQSMEM      ; 3 BIT KESİRLİ'DEN  Iqs*2^8'İN ORİJİNAL DEĞERİ
*
* Iqs AKIMINI SINIRLA
*
        BCNDD   NORMALQ,LT
        SUB     #IQSMAX,3   ; 3 BIT KESİRLİ FORMAT'ININ İÇİNE Iqsmax AKIMINI SABİTLE
*
        LACC    #IQSMAX,3
        SACL    IQSMEM
*
        SPLK    #04h,FRAC   ;DEĞİŞEBİLEN DESİMAL NOKTASI BÖLÜMÜ
*
NORMALQ CALLD   DIVIDE
        LACC    IQSMEM
        SACL    DENOM
*
        LACC    QUOT,4      ; 8 BIT KESİRLİ  Ids/Iqs ELDE ET
        SAMM    TREG0
        SACL    QUOT
*************************************************************************
*
*          BU BÖLÜM ARCTAN(Ids/Iqs)  LİNEERLEŞTİRİR          *
*************************************************************************
*
FOR110  LACC    #110
        SUB     QUOT
        BCND    FOR310,LT
        MPY     #L110A
        PAC
        BD      TORANG
        BSAR    4
        ADD     #L110B
*************************************************************************
*
FOR310  LACC    #310
        SUB     QUOT
        BCND    FOR605,LT
        MPY     #L310A
        PAC
```

```
        BSAR   4
        ADD    #L310B,0
        BD     TORANG
*************************************************************************
*
FOR605  LACC   #605
        SUB    QUOT
        BCND   FOR1605,LT
        MPY    #L605A
        PAC
        BSAR   4
        BD     TORANG
         ADD    #L605B,0
*************************************************************************
*
FOR1605 LACC   #1605
        SUB    QUOT
        BCND   FOR7320,LT
        MPY    #L1605A
        PAC
        BSAR   4
        BD     TORANG
         ADD    #L1605B,0
*************************************************************************
*
FOR7320 MPY    #L7320A
        PAC
        BSAR   4
        ADD    #L7320B,0
*************************************************************************
*
*               LINEERLEŞTİRME TAMAMLANDI                *
*************************************************************************
*
TORANG BSAR 8 ;BU BÖLÜM arctan(Ids/Iqs) ÜZERİN. ÖLÇEK. DOLAYI LİNEERLEŞ. İÇİNDİR

        SACL   THETA          ;*MOMENT AÇISI BULUNDU
*
STATCUR ADD    #TBLSEC,0
        TBLR   TEMP1
        LACC   TEMP1
        SAMM   TREG0
        MPY    IQSMEM         ; 3 BIT KESİR. SEC. DEĞERİYLE  Iqs'İ ÇARP
        PAC
        BSAR 8  ;SEC. LOOK TABLOSUNDA YAPILAN ÖLÇEKLEMEDEN DOLAYI
        SACL   ISMEM          ; 3 BIT KESİRLİ STATOR AKIMI BULUNDU
*
CURANG  LACC   ANGK
```

```
        SUB    THETA,3
        BSAR   3
        NOP
        XC     1,LT
         ABS
        SACL   TEMP1          ;A REFERANS AKIMININ FAZ AÇISI
*************************************************************************
*
*          BU BÖLÜM FAZ AKIMLARININ FAZ AÇILARINI BULUR        *
*************************************************************************
*
FORLT90 LACC   TEMP1
        SUB    #90
        BCNDD  FOLT180,GT
         CALLD  CALCUR
          B     CURCONT
*       SST    #0,XR0
*       SST    #1,XR1
*
* SONRA AKIM KONTROL ALTYORDAMI
*
*************************************************************************
*
FOLT180 LACC   TEMP1
        SUB    #180
        BCND   FOLT270,GT
        SACL   TEMP1
        LACC   ISMEM
        NEG
        SACL   ISMEM
        CALLD  CALCUR
          B     CURCONT
*       SST    #0,XR0
*       SST    #1,XR1
*
* SONRA AKIM KONTROL ALTYORDAMI
*
*************************************************************************
*
FOLT270 LACC   TEMP1
        SUB    #270,0
        BCND   FOLT360,GT
        LACC   TEMP1
        SUB    #0180
        SACL   TEMP1
        LACC   ISMEM
        NEG
        SACL   ISMEM
```

```
        CALLD   CALCUR
         B      CURCONT
*       SST    #0,XR0
*       SST    #1,XR1
*
* SONRA AKIM KONTROL ALTYORDAMI
*
*************************************************************************
*
FOLT360 LACC   #360,0
        SUB    TEMP1
        SACL   TEMP1
        CALLD   CALCUR
         B      CURCONT
*       SST    #0,XR0
*       SST    #1,XR1
*
* SONRA AKIM KONTROL ALTYORDAMI
*
*************************************************************************
*
*           VECTÖR KONTROL TAMAMLANDI, SONRA AKIM KONTROL        *
*************************************************************************
*
CURCONT ZAP
        SUB    IAACT
        SUB    ICACT
        SACL   IBACT          ;ÜÇÜNCÜ GERÇEK FAZ AKIMINI OLUŞTUR
*
CHECKA  BIT    IAREF,0
        BCND   NEGREFA,TC     ;IF IAREF<0
POSREFA LACC   IAREF          ;IF IAREF=>0
        SUB    IAACT,3
        BCND   POSBOTA,GT     ; (Iaref)<=Iaact DURUMUNDA KONTROL ET
FIRETR1 APL    #7e66h,TRANSEL
        OPL    #4200h,TRANSEL  ;DİĞER FAZ AKIMLARI
        B      CHECKB
*
POSBOTA LACC   IAREF
        SUB    IAACT,3
        BCND   CHECKB,LT      ; (Iaref)<=Iaact DURUMUNDA
FIRETR4 APL    #3c7fh,TRANSEL
        OPL    #0018h,TRANSEL  ;DİĞER FAZ AKIMLARI
*
CHECKB  BIT    IBREF,0
POSBREF BCND   NEGREFB,TC     ;IF IBREF<0
POSTOPB LACC   IBREF          ;IF IBREF=>0
*
```

```
        SUB     IBACT,3
        BCND    POSBOTB,GT      ; (Ibref)<=Ibact DURUMUNDA
FIRETR3 APL     #7e3ch,TRANSEL
        OPL     #1800h,TRANSEL  ;DİĞER FAZ AKIMLARI
        B       CHECKC
*
POSBOTB LACC    IBREF
        SUB     IBACT,3
        BCND    CHECKC,LT       ; (Ibref)<=Ibact DURUMUNDA
FIRETR6 APL     #667eh,TRANSEL
        OPL     #0042h,TRANSEL  ;DİĞER FAZ AKIMLARI
*
CHECKC  BIT     ICREF,0
POSREFC BCND    NEGREFC,TC      ;IF ICREF<0
POSTOPC LACC    ICREF           ;IF ICREF=>0
*
        SUB     ICACT,3
        BCND    POSBOTC,GT      ; (Icref)<=Icact DURUMUNDA
FIRETR2 APL     #7e5ah,TRANSEL
        OPL     #2400h,TRANSEL  ;DİĞER FAZ AKIMLARI
        B       CHECKA
*
POSBOTC LACC    ICREF
        SUB     ICACT,3
        BCND    CHECKA,LT       ; (Icref)<=Icact DURUMUNDA
FIRETR5 APL     #5a7eh,TRANSEL
        OPL     #0024,TRANSEL   ; DİĞER FAZ AKIMLARI
*       B       CHECKA
*
* AKIM KONTROL TAMAMLANDI
*
        BD      AGAIN
        OUT     TRANSEL,PA6
*
NEGREFA LACC    IAREF
        SUB     IAACT,3
        BCND    FIRETR1,LEQ
NEGBOTA LACC    IAREF
        SUB     IAACT,3
        BCND    FIRETR4,GEQ
        B       CHECKB
*
NEGREFB LACC    IBREF
        SUB     IBACT,3
        BCND    FIRETR3,LEQ
NEGBOTB LACC    IBREF
        SUB     IBACT,3
        BCND    FIRETR6,GEQ
```

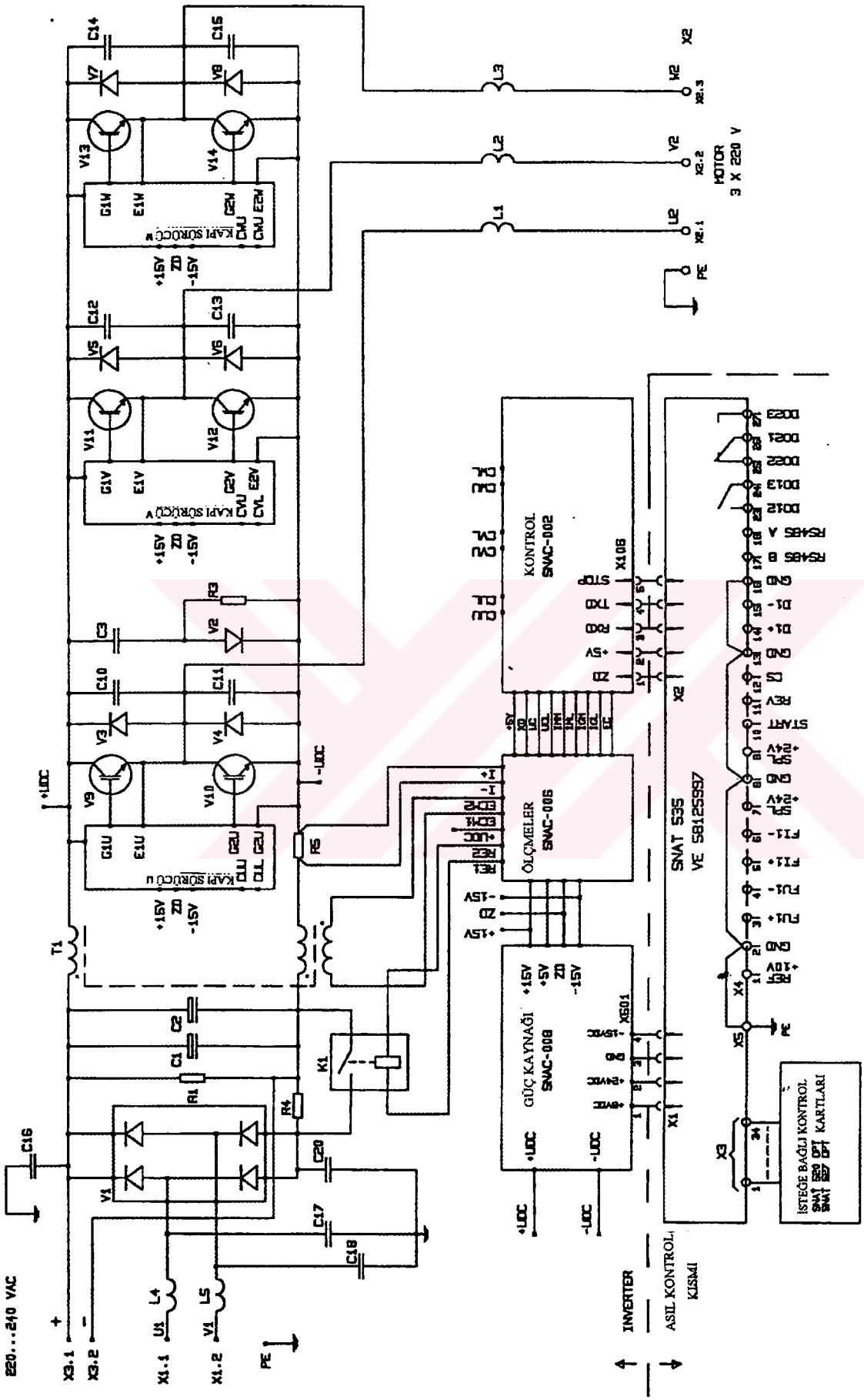
```
        B     CHECKC
*
NEGREFC LACC   ICREF
        SUB    ICACT,3
        BCND   FIRETR2,LEQ
NEGBOTC LACC   ICREF
        SUB    IBACT,3
        BCND   FIRETR5,GEQ
*****************************************************************************
*
*3 BIT KESİRLİ (FRACTION)'NIN DAHA SONRAKİ HESAPLAMALARI İÇİN
VEKTÖR        KONTROLÜNDE        KULLANILAN        VERİYİ        TAŞI
*****************************************************************************
*
INITEND LACC   EK
        SACL   EK_1
        LACC   SUMK
        SACL   SUMK_1
        LACC   EK_1
        BCND AGAIN,NEQ ;act.spd<>ref.spd İSE, EŞİT OLUNCAYA KADAR İŞLEME DE. ET
        MAR    *+
        LACC   *
        SACL   RSDMEM,3
*****************************************************************************
*
* BU ALTYORDAM REFERANS FAZ AKIMLARI  A,B AND C'Yİ HESAPLAR     *
*****************************************************************************
*
CALCUR  LACC   #TBLCOS
        ADD    TEMP1
        TBLR   TEMP0
        LT     ISMEM
        MPY    TEMP0
        PAC
        BSAR 15 ;COSINE DEĞERLERİ ÜZERİNDEN YAPILAN ÖLÇEKLEMEDEN DOLAYI
        SACL   IAREF       ; 3 BIT KESİRLİ FAZ AKIMI A BULUNDU
*
CALCIC  LACC   TEMP1
        SUB    #60
        ABS
        ADD    #TBLCOS,0
        TBLR   TEMP0
        LT     ISMEM
        MPY    TEMP0
        PAC
        BSAR   15
        NEG
        SACL   ICREF          ; 3 BIT KESİRLİ FAZ AKIMI C BULUNDU
```

```
*
CALCIB LACC   #0h
       SUB    IAREF
       SUB    ICREF
       SACL   IBREF        ; 3 BIT KESİRLİ FAZ AKIMI B BULUNDU
*
       RET
*      LST    #0,XR0
*      LST    #1,XR11
*************************************************************************
*
*                  BÖLME                    *
*************************************************************************
*
DIVIDE LT     NUMER
       MPY    DENOM
       SPH    TEMPSGN      ;ÇARPMANIN İŞARETİNİ DEPOLA
*
       LACL   DENOM
       ABS
       SACL   DENOM
       LACL   #15
       ADDS   FRAC
       SACL   TEMP0        ;BÖLÜMÜN DİJİT SAYISI
       LACL   NUMER
*
       RPT    TEMP0
        SUBC  DENOM        ;BÖLMEYİ YAP
*
       SACL   QUOT
       BIT    TEMPSGN,0    ;SONUÇ İŞARETİNİ KONTROL ET
       BCND   NEGAT,TC
        RET
NEGAT  LACL   #0h
       SUB    QUOT
       SACL   QUOT
       RET
```

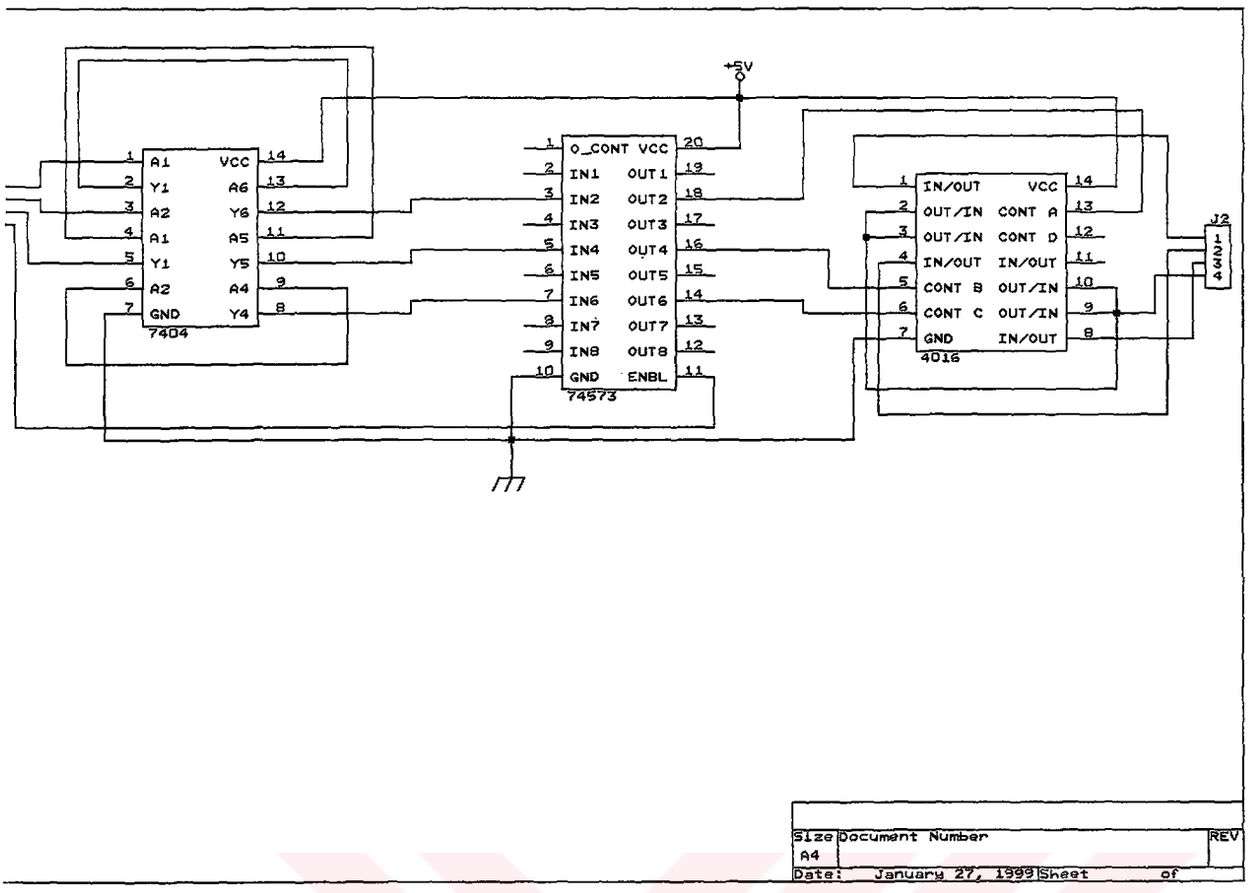
```
*************************************************************************
*                KESME SERVİS İŞLEMLERİ                 *
*************************************************************************
*************************************************************************
*
*                İŞLEMCİ ÇALIŞMAYA HAZIR (INITIALIZATION)                *
*************************************************************************
*
* Initialization :                              *
* Initialization DP                                     *
* Herhangi bir  2 KB program hafıza sınırları boyunca vektör tablosu haritalama (0800h)*
*Veri hafıza alanına D2 biçimle.                        *
*************************************************************************
*
        .text
*
INIT    SPLK    #15fch,TEMP1    ; ARB<-0, CNF<-1, TC<-0, C<-0, SXM<-1, HM<-1
        LST     #1,TEMP1
*
        SPLK    #0830h,PMST     ;S/A RAM->PROG.HAF. (0800h-2C00h)
                                ;ROM->ON-CHIP (0000h-0800h)
                                ;D/A RAM B0->PROG. (FE00h-FFFFh)
                                ; IPTR->1 INT. VECTÖRLER (0800h-0840h)
                                ; AVIS->0 (TAKİP EDİLEN KESME ADRESLERİ
                                ;IACK'YLA İLGİLİ KOD ÇÖZME,
*
        SPLK    #24h,CWSR
*
        SPLK    #01h,IOWSR
*
        SPLK    #0613h,TEMP1
        LST     #0,TEMP1        ;PM<-0, DP<-19, INTM<-1, ARP<-0, OVM<-0
*
        BD      MAIN
        SPLK    #1,IMR          ;ROTOR POZİSYONU İÇİN ENABLE INT1
*************************************************************************
*
*                INITIALIZATION TAMAMLANDI                *
*           Bu kesme servis çalşması  INT1'i kullanır             *
*************************************************************************
*
        IN      ANGK,PA2
        IN      IAACT,PA0       ;KARŞILIK GELEN AKIM DEĞERLERİ
        IN      ICACT,PA1       ; *BU ROTOR POZİSYONU
        BANZ    ROTANG
        LAR     AR4,#NUMSLOTS
        MAR     *,AR1
*
```

```
ROTANG LACL   TIME
       SUBS   TIM
       SACL   TIME
       LACL   #10h
       SAMM   TCR
       LACL   #20h
       SAMM   TCR
       SMMR   TIM,TIME
*
       RETI
*
       .end
```

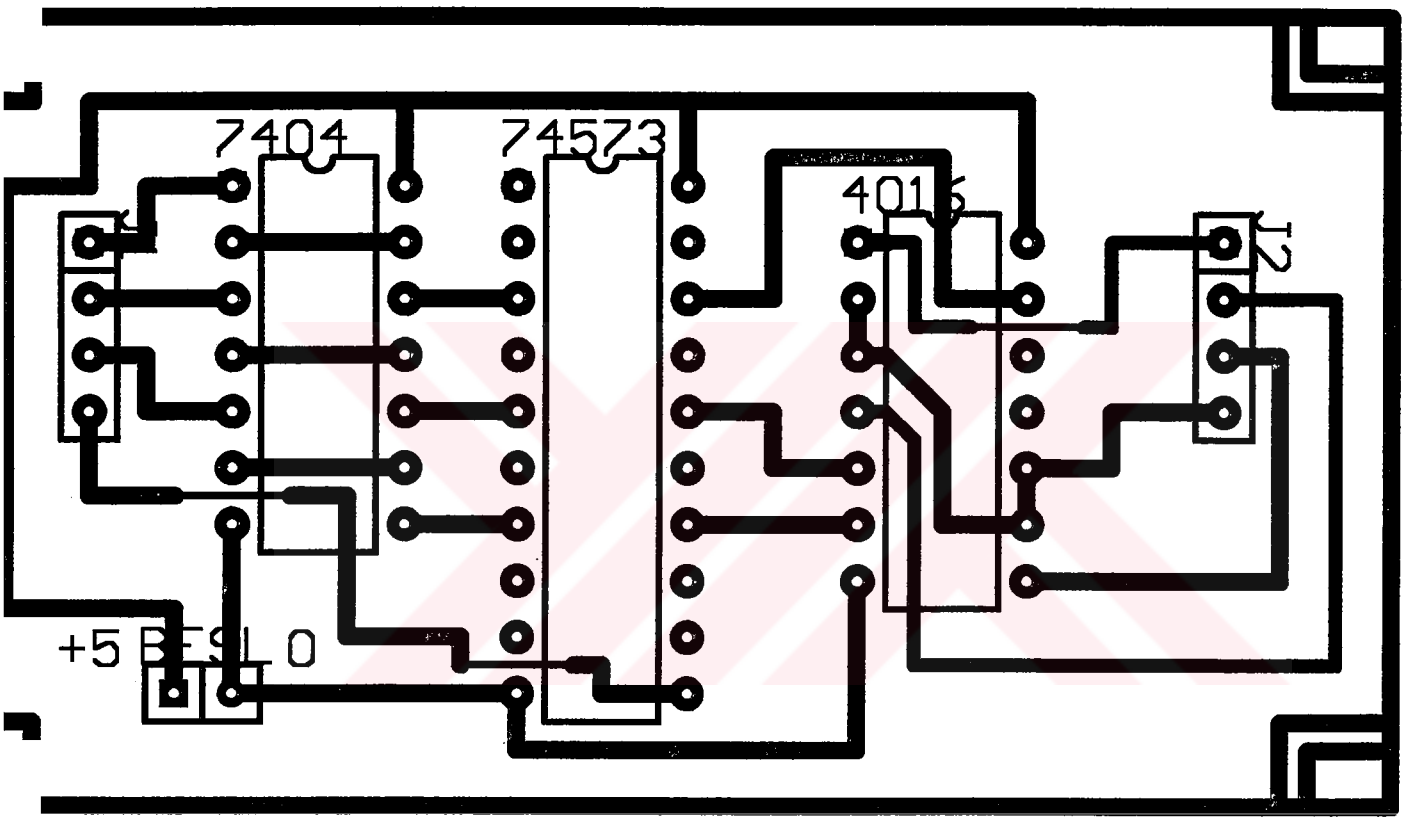
EK 7 İnverter Devre Şeması


220...240 VAC
X3.1
X3.2
X1.1
X1.2
PE
INVERTER
ASIL KONTROL KISMI
SNAT 535
VE SB125997
KONTROL
SNAC-002
ÖLÇMELER
SNAC-005
GÜÇ KAYNAĞI
SNAC-008
MOTOR
3 X 220 V
ISTEĞE BAĞLI KONTROL KARTLARI

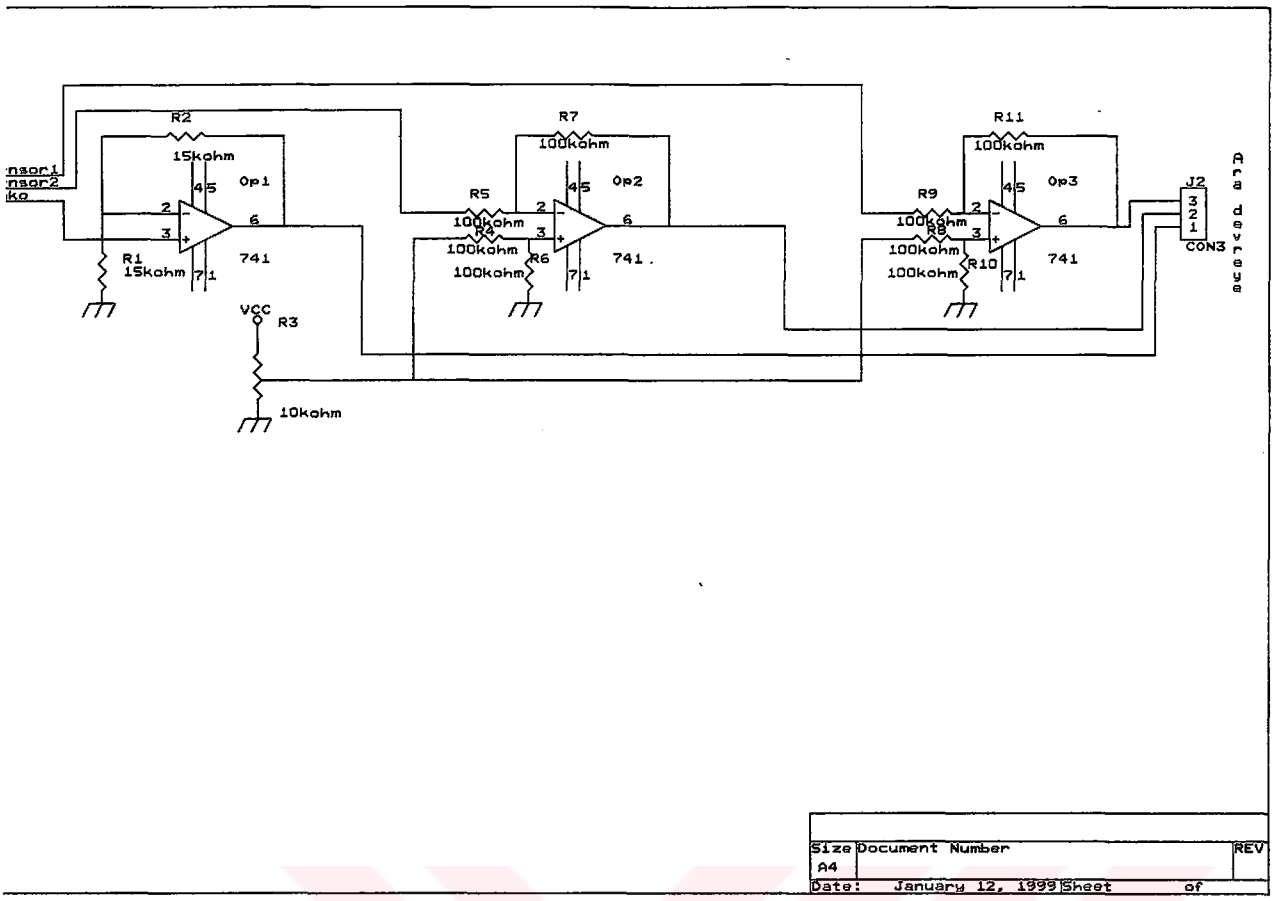
EK 8 Ara Devre 1 (Ölçü Amplifikatörleri)



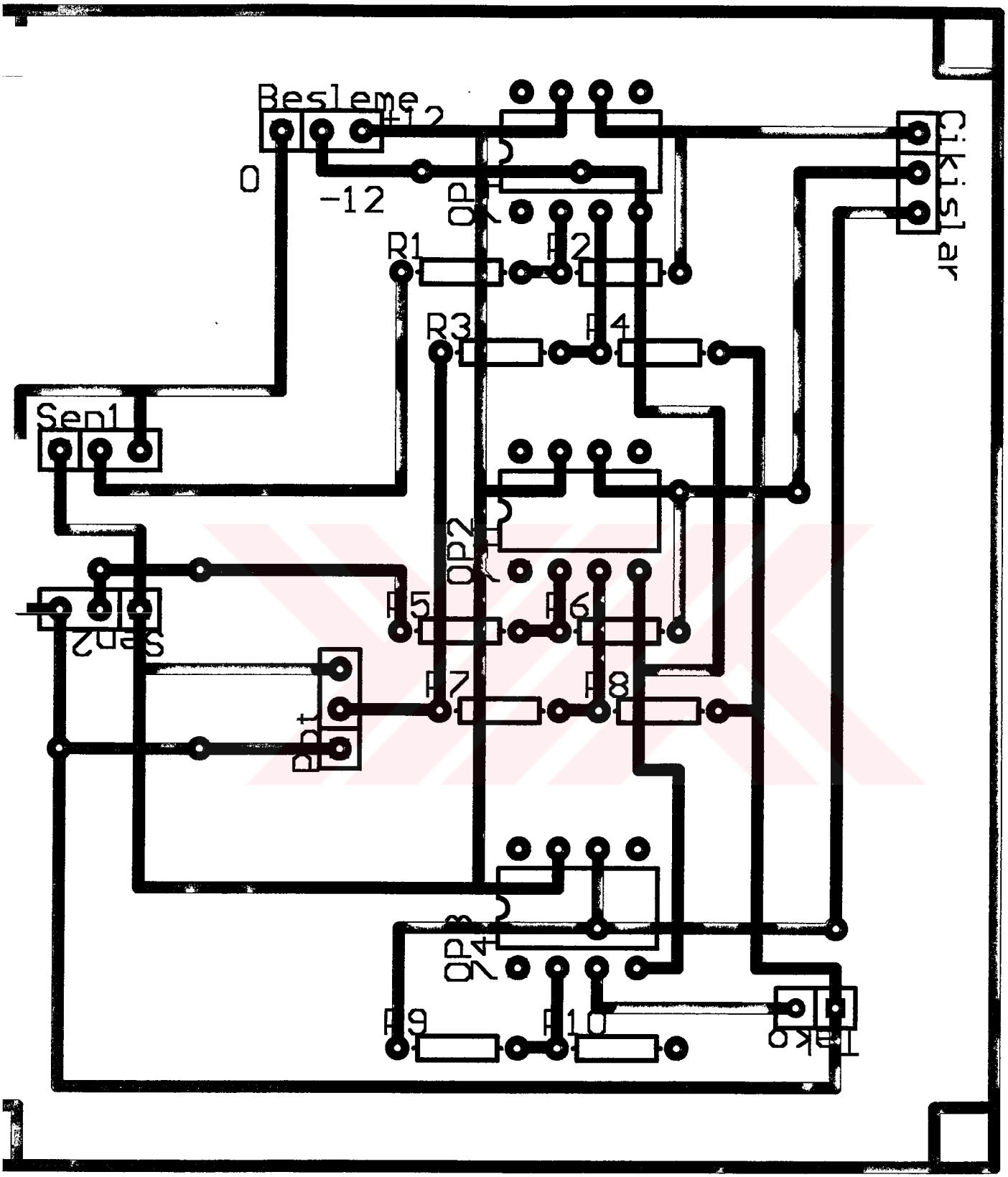
Size	Document Number	REV
A4		
Date: January 27, 1999	Sheet of	


7404
74573
4016
J2
+5
RESET

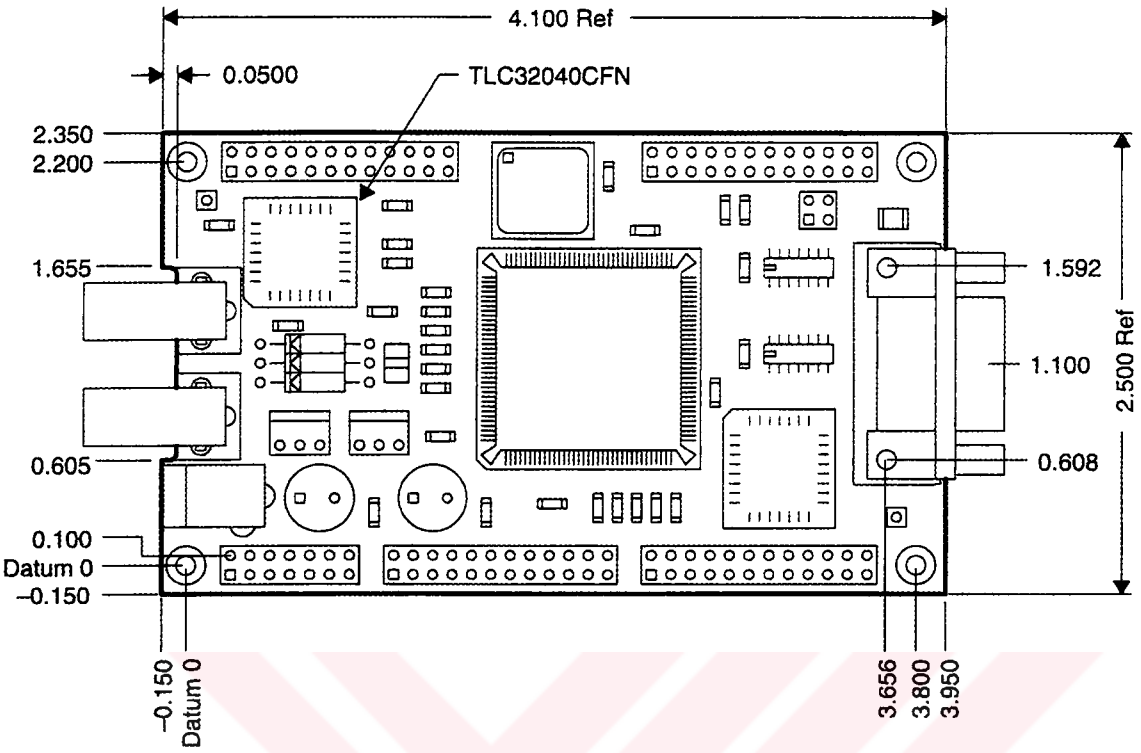
EK 9 Ara Devre 2 (Bekletme Devresi)

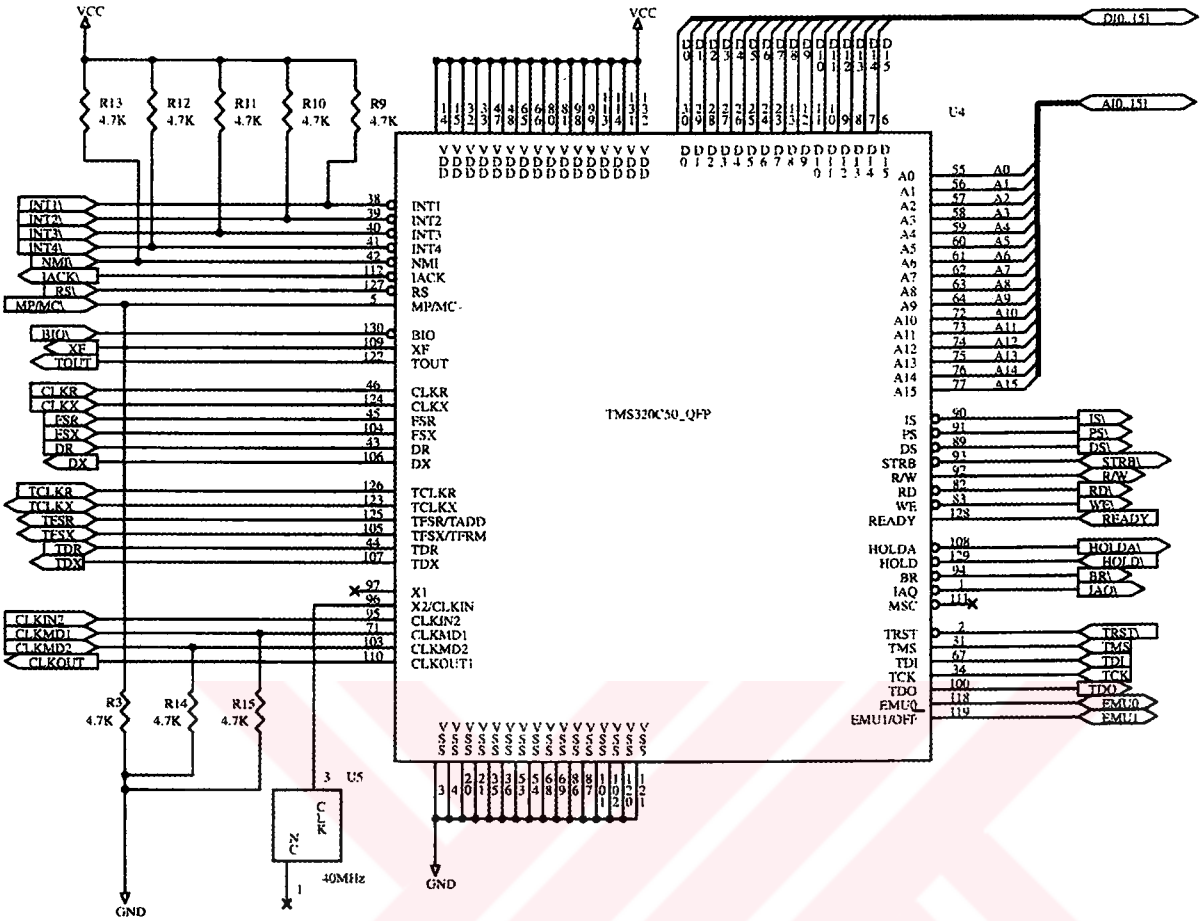


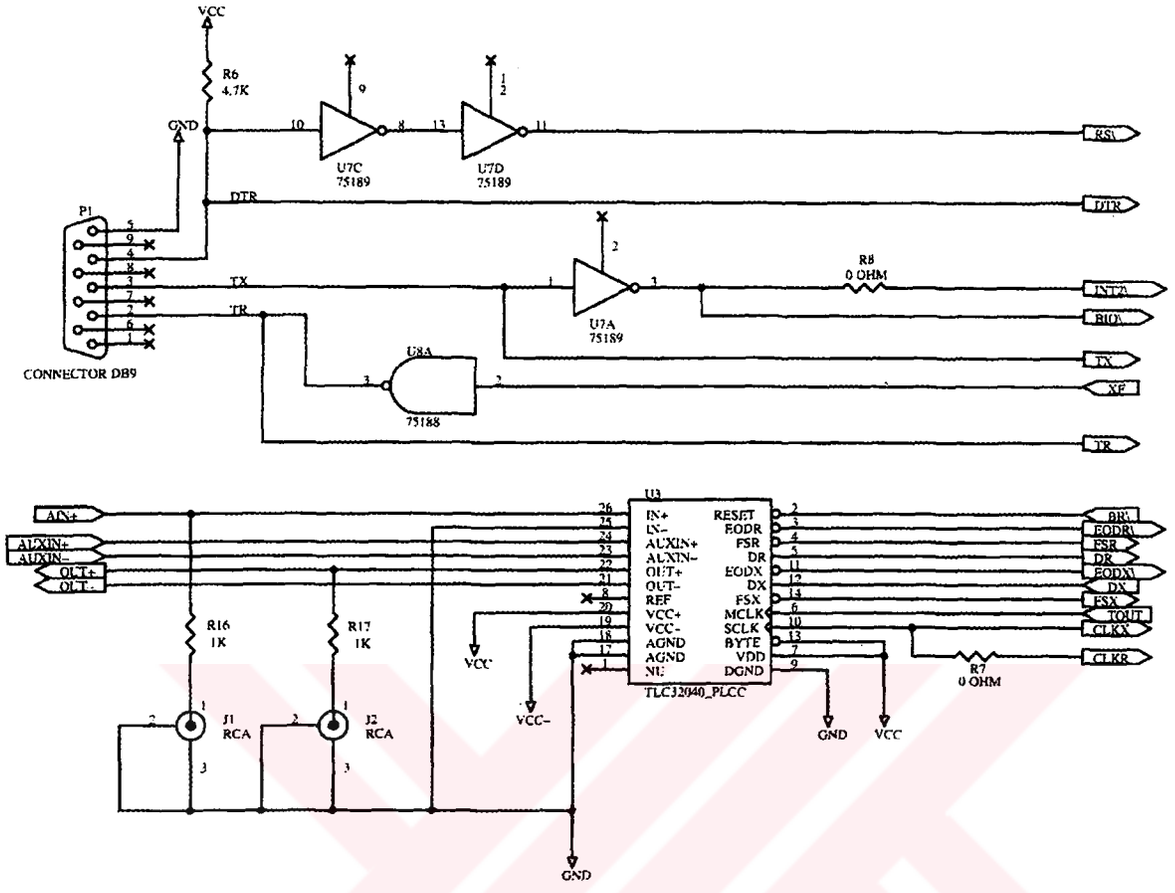
Size	Document Number	REV
A4		
Date: January 12, 1999	Sheet of	

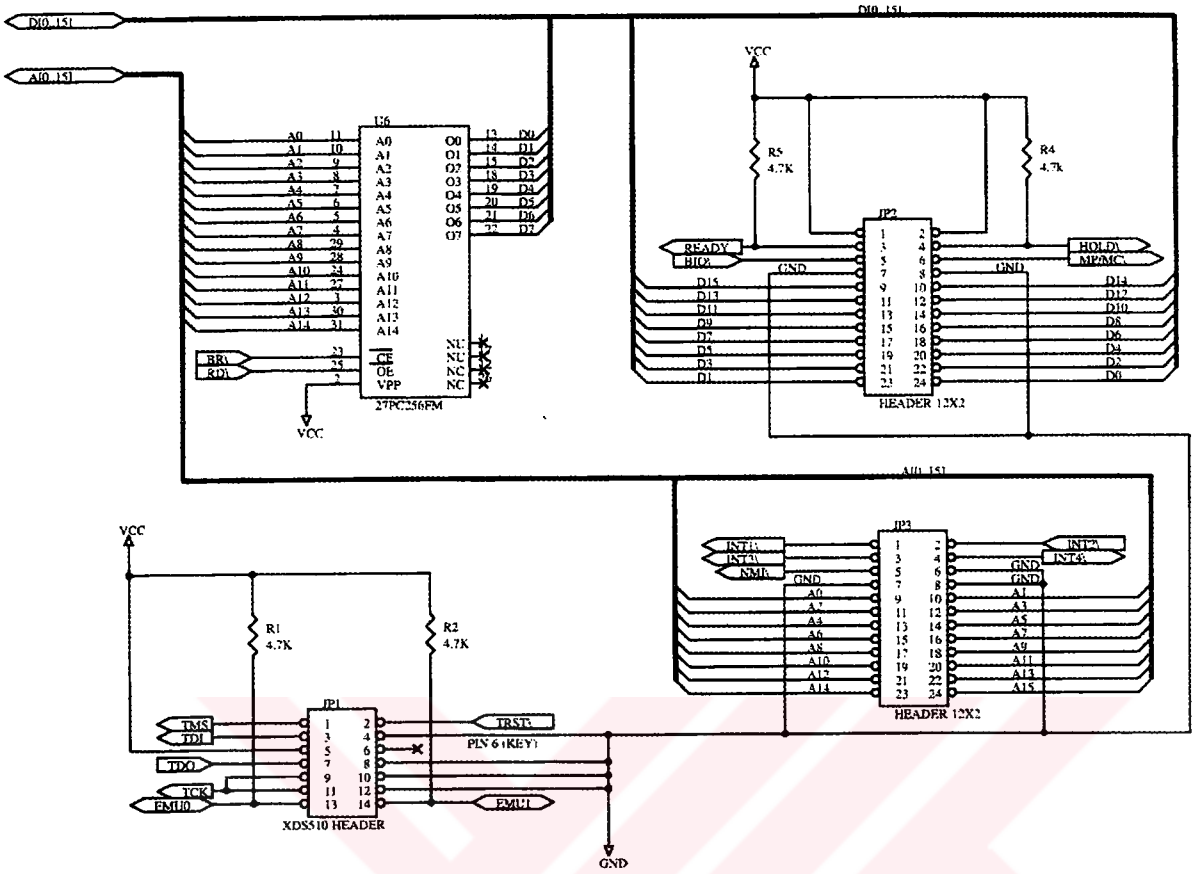

Besleme
+12
0
-12
OP1
R1
R2
R3
R4
Cikis1
Sen1
OP2
R5
R6
R7
R8
Sen2
Buz
OP3
OP4
R9
R10
Cikis2

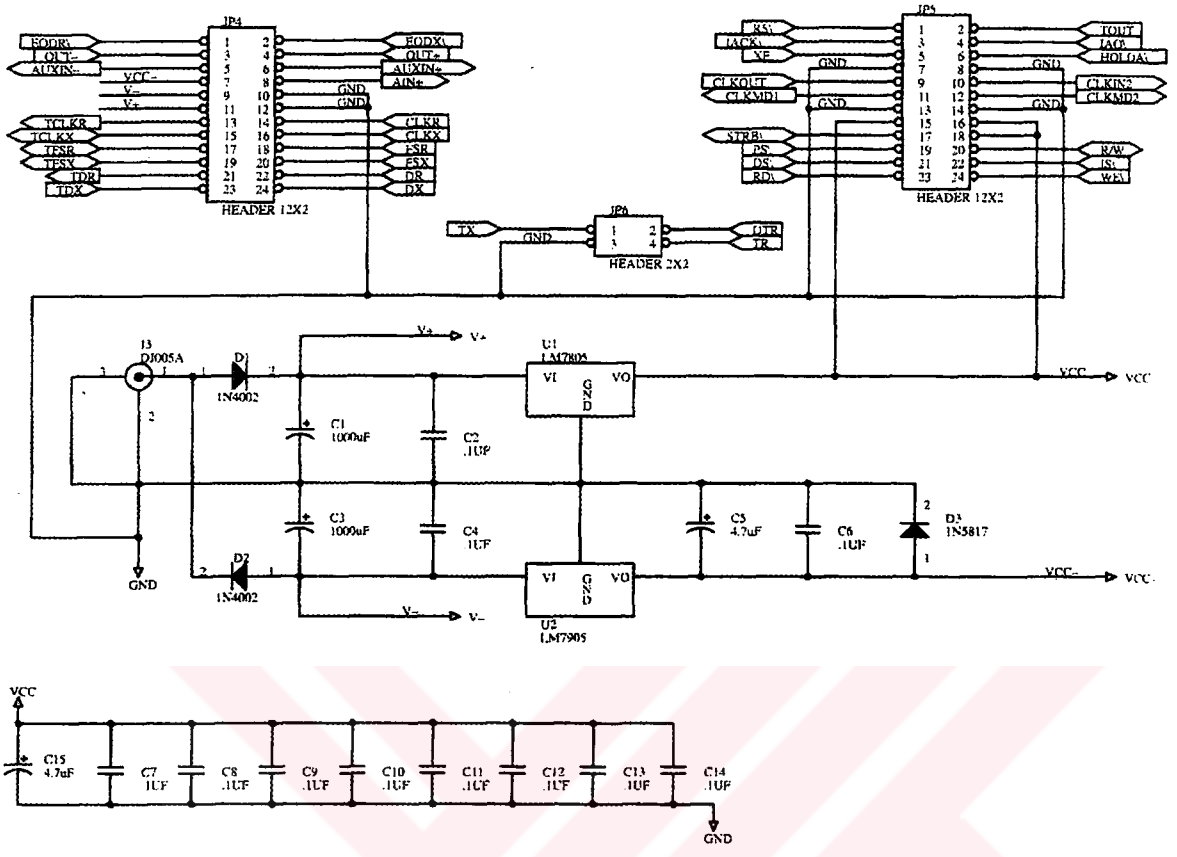
EK 10 DSP Kartı Yerleşim Şemaları


4.100 Ref
0.0500
TLC32040CFN
2.350
2.200
1.655
0.605
0.100
Datum 0
–0.150
1.592
1.100
0.608
2.500 Ref
–0.150
Datum 0
3.656
3.800
3.950


VCC
R13 4.7K
R12 4.7K
R11 4.7K
R10 4.7K
R9 4.7K
INT1\
INT2\
INT3\
INT4\
NMI\
IACK\
RS\
MP/MC\
BIO\
XF
TOUT
CLKR
CLKX
FSR
FSX
DR
DX
TCLKR
TCLKX
TFSR
TFSX
TDR
TDX
CLKIN2
CLKMD1
CLKMD2
CLKOUT
R3 4.7K
R14 4.7K
R15 4.7K
U5
NC
40MHz
GND
TMS320C50_QFP
INT1
INT2
INT3
INT4
NMI
IACK
RS
MP/MC
BIO
XF
TOUT
CLKR
CLKX
FSR
FSX
DR
DX
TCLKR
TCLKX
TFSR/TADD
TFSX/TFRM
TDR
TDX
X1
X2/CLKIN
CLKIN2
CLKMD1
CLKMD2
CLKOUT1
U4
D[0..15]
A[0..15]
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
A12
A13
A14
A15
IS
PS
DS
STRB
R/W
RD
WE
READY
HOLDA
HOLD
BR
IAQ
MSC
TRST
TMS
TDI
TCK
TDO
EMU0
EMU1/OFF
IS\
PS\
DS\
STRB\
R/W
RD\
WE\
READY
HOLDA\
HOLD\
BR\
IAQ\
TRST\
TMS
TDI
TCK
TDO
EMU0
EMU1


VCC
R6
4.7K
GND
U7C
75189
U7D
75189
RS\
DTR
DTR
P1
CONNECTOR DB9
TX
TR
U7A
75189
R8
0 OHM
INT\
BIO\
TX
XF
U8A
75188
TR
U3
AIN+
AUXIN+
AUXIN-
OUT+
OUT-
R16
1K
R17
1K
J1
RCA
J2
RCA
VCC
VCC-
GND
IN+
IN-
AUXIN+
AUXIN-
OUT+
OUT-
REF
VCC+
VCC-
AGND
AGND
NU
RESET
FODR
FSR
DR
FODX
DX
FSX
MCLK
SCLK
BYTE
VDD
DGND
TLC32040_PLCC
R7
0 OHM
GND
VCC
BR\
FODR\
FSR
DR
FODX\
DX
FSX
TOUT
CLKX
CLKR


D[0..15]
A[0..15]
U6
27PC256FM
VCC
R5
4.7K
R4
4.7K
JP2
READY
HI02
HOLD\
MP/MC\
GND
HEADER 12X2
JP3
HEADER 12X2
R1
4.7K
R2
4.7K
JP1
TMS
TDI
TDO
TCK
EMU0
TRST\
PIN 6 (KEY)
EMU1
XDS510 HEADER
GND



ZGEÇMİŞ

Doğum tarihi	12-09-1970	
Doğum yeri	İstanbul	
İlkokul	1976-1981	İstanbul Bayrampaşa Şair Baki İlkokulu
Ortaokul	1981-1984	İstanbul Bayrampaşa Fetihtepe Ortaokulu
Lise	1984-1987	İstanbul Bayrampaşa A. Rıfat Canayakın Lisesi
Lisans	1988-1992	Yıldız Üniversitesi Mühendislik Fak. Elektrik Mühendisliği Bölümü
Yüksek Lisans	1992-1993	1 Yıl İngilizce Hazırlık
Yüksek Lisans	1993-1995(Ocak)	Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü Elektrik Müh. Anabilim Dalı
Doktora	1995-1999	Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü Elektrik Müh. Anabilim Dalı
Çalıştığı kurumlar	1994-1997	Celal Bayar Üniversitesi Mühendislik Fak. Elektrik-Elektronik Mühendisliği Böl. Araştırma Görevlisi
	1997-Devam ediyor	Yıldız Teknik Üniversitesi Elektrik-Elektronik Fakültesi Elektrik Mühendisliği Böl. Araştırma Görevlisi