YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

GÜRÜLTÜ AZALTMADA LMS ADAPTİF SÜZGEÇLERİN FPGA KULLANARAK UYGULANMASI

Elektronik ve Haberleşme Mühendisi Olcay SEVİM

FBE Elektronik ve Haberleşme Mühendisliği Anabilim Dalı Haberleşme Programında Hazırlanan

YÜKSEK LİSANS TEZİ

Tez Danışmanı : Yrd. Doç. Dr. Ünal KÜÇÜK (YTÜ)

İSTANBUL, 2007

İÇİNDEKİLER

Sayfa

SİMGE LİSTESİ

μ	Ağırlık sabiti
$\mu(n)$	Beklenti operatörünün değeri
σ_u^2	Değişinti
∇	Gradient vektörü
p^H	P'nin Hermit operatörü
p^T	P'nin Transpozu
$\wedge$	Tahmin operatörü
p	Tap giriş vektörü ile istenilen cevap arasındaki çapraz korelasyon vektörü
R	Tap giriş vektörü korelasyon matrisi

KISALTMA LİSTESİ

db	Decibel
FPGA	Field Programmable Gate Array
N2LMS	Kendinden uyarlamalı ağırlık sabitli Normalized Least Mean Squares
LMS	Least Mean Squares
NLMS	Normalized Least Mean Squares
VHSIC	Very High Speed Integrated Circuits
VHDL	Very High Speed Integrated Circuits Hardware Description Language

ŞEKİL LİSTESİ

ÇİZELGE LİSTESİ

ÖNSÖZ

Teknoloji tarihi, insanların etkisi altında kalarak sürekli olarak gelişimini sürdürmüştür. Gelişimin olduğu yerde değişimler ve yenilikler söz konusu olmaktadır. Bu değişim altındaki yenilikler bizlere yabancı olsalar da yeniliklerden kaçmak doğru olmayacaktır. Bu düşünceyle genç bir mühendis adayı olarak, tezimi daha önce uğraşmadığım fakat çok yakın zamanda teknolojik sahada önemli bir yere sahip olabilecek araçlarla tamamlamayı uygun buldum. Bu adımdan sonra yaptığım araştırmalarda Türkiye'de, tezimin bir parçası olan FPGA(Field Programmable Gate Array) kullanımının henüz yaygınlaşmamış, kullanımının kısıtlı olduğunu gördüm. Tamamlamış olduğum tezin kariyerimde bana yardımcı olmasını, benden sonraki arkadaşlarıma yardımcı olmasını dilerim.

Bu tez çalışması ana hatlarıyla, ses işaretinden gürültünün arındırılması En Küçük Karesel (LMS, Least Mean Square) ve Normalize En Küçük Karesel (NLMS, Normalized Least Mean Square) algoritmalarını incelemekte ve bunların FPGA üzerinde gerçek zamanlı çalışmasına ilişkin bilgileri sunmaktadır.

Çalışmamda, ilk olarak süzgeçlerden bahsederek kendinden uyarlamalı süzgeçleri açıklamaya çalıştım. Bu çalışmayla farklı süzgeçlerin birbirlerine göre farklılıklarını göstermek istedim. Ayrıca gerçek zamanlı uygulamalara geçiş sürecini ve burada yaşanabilecek sıkıntıları belirtmek istedim. Son olarak; ses işaretinden gürültü işaretini silebilecek algoritmayı FPGA'e göre değiştirerek, gerçek zamanlı olmayan ve sonsuz doğrulukta işlemler ile gerçek zamanlı çalışan ve belirli doğrulukta çalışan işlemleri karşılaştırarak en yakın sonuçlara nasıl ulaşılabileceği hakkında bilgilerimi sundum.

Çalışmamın, sonunda okunmaya değer yardımcı bir kaynak olarak kullanılmasını diliyorum.

Çalışmama yaptıkları katkılarından dolayı öncelikle tüm hayatım boyunca benden yardımı esirgemeyen ve beni sürekli destekleyen çok sevdiğim aileme, hocam Ünal Küçük'e, değerli arkadaşlarım Evren Cesur ve Ulaş Arıca'ya teşekkür ediyorum.

Temmuz 2006, Olcay Sevim

ÖZET

Doğal ortamda insan kulağına gelen ses bilgisi birden fazla kaynaktan çeşitli bileşenlerle birlikte gelmektedir. Kulak, bu toplanmış işareti değerlendirmeye çalışmaktadır. Bu durumda istenmeyen işaretler ses kalitesini düşürmekte hatta bazen duyulan sesin anlaşılmamasına neden olmaktadır. Gürültü birden fazla kaynaktan gelebilir, genliği ve frekansı değişken olabilir. Böyle durumlarda, bu değişimler kendini tekrarlamıyorsa ve önceden bilinmiyorsa önlem almak zorlaşır, anlık çözümler bulmak zorunda kalınır.

Çeşitli gürültü kaynakları için çeşitli süzgeçler kullanılmaktadır. Eğer gürültü bileşenleri belli değerleri alıyorsa ve belli zaman aralıklarında kendini tekrarlıyorsa alınabilecek önlemler hem donanım açısından daha basittir hem de sonuçları oldukça iyidir. Fakat gürültü tamamen öngörülemez ise ve belli bir periyodu yoksa farklı önlemler almak gerekir. Bunun için matematiksel çözümler üretilebilir ancak bunun donanımsal maliyeti çok yüksektir yada imkansızdır. Bu yüzden kendinden uyarlamalı süzgeçler kullanılabilir.

Kendinden uyarlamalı süzgeçler, süzgeç çıkışından aldıkları bilgiyi kendi ağırlıklarını güncellemede kullanarak çıkışı girişe göre düzeltmeye çalışır. Günümüzde oldukça geniş uygulamaları bulunan kendinden uyarlamalı süzgeçlerden biri de LMS tabanlı veya bir ileriki hali NLMS tabanlı süzgeçlerdir. LMS tabanlı süzgeçlerin sonuçları mükemmel değildir fakat kabul edilebilir sonuçlar üretmektedir. LMS tabanlı süzgeçler bunun yanında donanıma uyarlaması bakımından avantajlara sahiptir.

Donanım, algoritmaların yanında ikinci bir problemdir. Bulunan en uygun algoritmanın donanımı imkansız olabilir bu yüzden algoritma ile donanım arasında en iyi ilişkiyi kurabilmek çözümün diğer parçasıdır. Bu çalışmada algoritmadan donanıma geçiş süreci dikkatli bir şekilde anlatılmıştır.

Sunulan bu çalışma: iki giriş işaretini alarak uygun çıkış üretebilen kendinden uyarlamalı, LMS yöntemi ile çalışan süzgeç ve bunun FPGA üzerinde gerçek zamanlı olarak uygulanmasını anlatmaktadır.

Anahtar kelimeler: Uyarlamalı süzgeçler, LMS, FPGA, gerçek zamanlı algoritmalar, ses işleme, gürültü bastırma.

ABSTRACT

Normally, sound contents are compound of various sources that is processed at human ears. Such a condition, ear tries to gather valuable information among others which may resulted in failure. In the case of failure, we conclude that noise is at harmful degree. In addition, when noise comes from different sources with changing contents like frequency and amplitudes with no periodicity, solutions to cancel noise will be a quite hard job and instantenous solutions may be needed.

The varios noise kinds, the various fitler types has been found. If noise contents are always same and known having periodicity, solutions will be easy with regarded success. Sometimes contens come with no information, in such cases high level mathmatical process will be required with highly costs of hardware. Instead, adaptive filters may be chosen in these conditions.

Adaptive filters try to adjust their weights according to their own outputs related with their own inputs. These kind of filters have been used in a quite range of applications. LMS or NLMS oriented adaptive filters is a good example of these filters. LMS oriented filters results are not excellent but accaptable. In addition, LMS oriented filters are one of the easiets filters which is easily implemented on hardwares.

Hardware consideration is a second problem with algorithms. The most performanced algorithm may have impossible hardware solutions. For this reason, finding the right relation between the algorithm and the hardware is a part of solution. In this thesis, the process of algorithm towards hardware solution is explained in details.

This thesis explains: the LMS oriented self-adaptive fitler which takes two input and produces one output. Also, real-time process of fitler is explaind with use of FPGA.

Keywords: Adaptive filters, LMS, FPGA, Real Time algorithms, audio processing, noise cancellation.

1. GİRİŞ

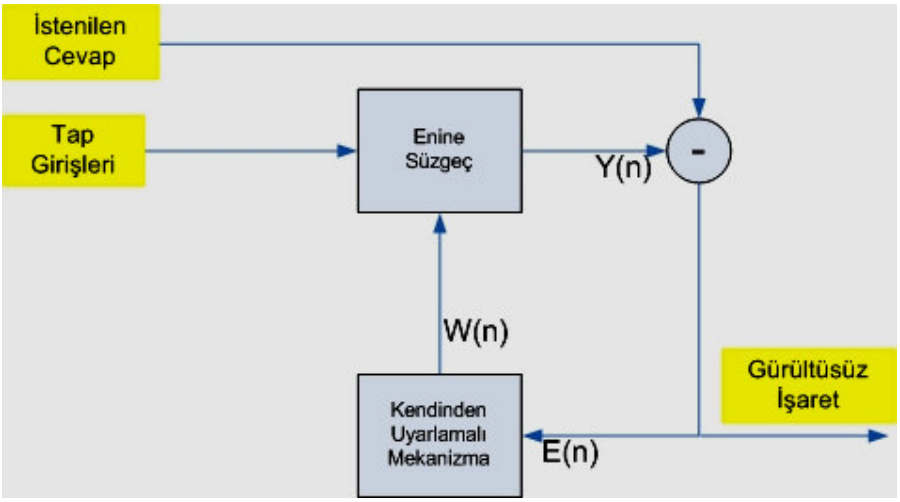
Bu tez ile En Küçük Karesel Ortalama (LMS, Least Mean Square) yöntemi ile uyarlamalı süzgeç kullanılarak arka fondaki gürültü işaretinin bastırılması ve bunun FPGA üzerinde gerçek zamanlı uygulaması gerçekleştirilmiştir.

Arka fondaki gürültüye örnek olarak, bir uçak kokpitindeki uğultu, bir ağır vasıta içerisindeki gürültü yada herhangi bir ortamda bulunan konuşmacının ses işaretine eklenen istenmeyen sesler verilebilir.

Gürültünün azaltılması işlemi süzme işlemidir. Süzgeç terimini uygulamada, ilgilenilen bilginin gürültülü bir ortamdan çekilip çıkarılması için gerekli olan donanım ve yazılımla ifade etmek doğru olur. Örneğin gürültülü ortamda, algılayıcı kullanarak işaretlerin toplanması istenildiğinde algılayıcı ile birlikte gerekli yazılım ve donanım kullanılarak istenilen bilgi algılayıcıdan alınır ve ilave edilen donanım ile işlenerek gürültüden arındırılmış bilgiye ulaşılır. Bu şekilde, uygulamada tipik bir sayısal süzgeç elde edilir.

Böyle bir süzgecin çıkışında süzülmüş işaret girişin doğrusal bir fonksiyonu ise bu süzgece doğrusal süzgeç denir. İstatiksel yaklaşımda doğrusal süzme problemine, belirlenmiş istatiksel parametrelerle(ortalama ve korelasyon fonksiyonları) çözüm üretmek uygun olur. Durağan girişler için genel Wiener süzgeçleri, durağan olmayan girişlerde de Kalman süzgeçleri kullanılabilir.

Wiener süzgeçleri işlem görecek bilginin öncül(Priori) istatiksel bilgilerine ihtiyaç duyar. Bu tip süzgeçler, ancak girişin istatiksel bilgileri ile öncül istatiksel bilgilere göre tasarlanan süzgecin bilgileri birbirine eş olduğunda en iyi performansı verirler. Bu bilgilerin bulunmadığı durumlarda süzgeç olumsuz sonuçlar verir, buna rağmen bu durumu düzeltmek için çeşitli teknikler geliştirilmiştir. Fakat bu teknikleri gerçek zamanlı(Real-Time) sistemlere uygulamak pek kolay olmamaktadır. Bu durumu düzeltmek için daha etkili bir çözüm olan kendinden uyarlamalı süzgeçler kullanılmaktadır. Kendinden uyarlamalı süzgeçler ilgili işaretin karakteristik bilgisinin ortamda bulunmadığı durumlarda yinelemeli(Recursive) algoritmasıyla, kendi kendini ortama göre ayarlayarak istenilen çıkışın oluşturulmasını sağlar. Bu algoritma ilk başlangıç koşullarıyla başlatılarak birkaç denemeden sonra istenilen çıkışı vermeye başlar. Buradan sonuç olarak, süzgeçteki parametrelerin bir denemeden diğer denemeye değişmesiyle, parametrelerin veriye bağımlı olduğu ve böylece süzgeç tarafından kendiliğinden güncellendiği anlaşılır.



Şekil 1.1 Kendinden uyarlamalı süzgeç

Şekil–1 de tipik bir kendinden uyarlamalı süzgeç örneği verilmiştir. Şekilde görüldüğü gibi girişler birden fazla olabilir ve süzgeç ağırlıklarını kendisi günceller. Çıkış bilgisi de ortamdan bağımsız bir şekilde elde edilir.

Her süzgeç tasarımında olduğu gibi kendinden uyarlamalı süzgeçler için de oluşturulan çıkış işaretinde kaliteyi arttırabilmek için süzgecin karmaşıklığını arttırmak gerekir, fakat bu durumda yine donanımsal kısıtlamalar karşımıza çıkar. Bu yüzden performans kriteri, kalite/donanım karmaşıklığı olarak seçilirse en iyi performansı verecek çözümler bu değerlendirmeye göre seçilebilir.

Bu çalışmada, kendinden uyarlamalı süzgeçlerden LMS yöntemi ile çalışan uyarlamalı süzgeci kullanılarak, bu algoritmanın FPGA üzerinde kullanılabilecek bir hale getirilmesi ve elde edilen çıktılarla ses işaretinden gürültünün arındırılması işlenmiştir. İlerleyen bölümlerde sırasıyla, sayısal süzgeçlere genel giriş, kendinden uyarlamalı süzgeçler, bilgisayar ortamından gerçek zamanlı ortama geçiş, FPGA'ler, algoritmanın FPGA aktarılması ve VHDL(Very High Speed Integrated Circuits Hardware Description Language) ile ilgili konularda bilgi verilmektedir.

2. SAYISAL SÜZGEÇLER

2.1 Genel

Uygulamada sayısal süzgeçlerden bahsedebilmek için uygun bir donanıma ve yazılıma sahip olmak gerekir. Bu bileşenler de uygulamaya göre çeşitlilik gösterebilir. Ama genel olarak sayısal süzgeçlerden bahsettiğimizde artık ayrık zaman düzleminde olduğumuz hatırlanmalıdır. Gerçek zamanda çalışabilen sayısal süzgeçlere sahip sistemin genel olarak aşağıdaki özelliklere sahip olması beklenir.

1- Önceden belirlenmiş bir bellek

2- Önceden bilinen işlem miktarı

Bu şekilde bir sistem için uygun yazılım ve donanım ile tipik sayısal süzgeç yapılabilir. Eğer sistemimiz doğrusal ve zamanla değişmeyen(time-invariant) ise giriş çıkış ilişkisi zaman düzleminde konvolüsyon toplamı ile ifade edilebilir(Mitra, 1998):

$$y[n] = \sum_{k=-\infty}^{\infty} h[k]x[n-k] \tag{2.1}$$

Ayrıca sayısal bir süzgeci fark eşitlikleri ile yazmak istersek aşağıdaki formül bulunur,

$$\sum_{k=0}^{N} d_k\, y[n-k] = \sum_{k=0}^{M} p_k\, x[n-k] \tag{2.2}$$

Burada $x[n], y[n]$ sırasıyla giriş ve çıkıştır, d_k, p_k sabitlerdir. Bu denklemde ayrık sistemin derecesi, $N \geq M$ olduğu sürece N'dir. Yukarıdaki denklemde k=0 için $y[n]$ hesaplanabilir.

$$y[n] = -\sum_{k=1}^{N} \frac{d_k}{d_0} y[n-k] + \sum_{k=0}^{M} \frac{p_k}{d_0} x[n-k] \tag{2.3}$$

Bu şekilde tanımlanmış bir sistemin nedensel olduğunu söyleyebiliriz. Doğrusal süzgeçlerde d_k, p_k sabitlerinin seçimi önemlidir. Süzgecin karakteristiğini bu sabitler belirler. Bu sabitlerin belirlenmesinde farklı parametreler vardır. Bunlar

1- Çalışma frekansı, bant genişliği

2- Sistemin önceden hesaplanan beklentisi, değişintisi gibi…

İlk parametre elde edilmek istenin işaretin karakteristiğinin bilinmesi ile ilgilidir, diğer parametre ise içinde bulunulan sistemin modellenmesi ile ilgilidir. Bu model rasgele süreçler ile hesaplanır.

Rasgele süreç (Stochastic process), istatiksel olgunun olasılık kurallarına göre zamanın bir fonksiyonu olarak açıklanmasıdır. Sabit katsayılı süzgeçler için bu süreç önemli iken kendinden uyarlamalı süzgeçler bu bilgileri kullanmaz. Kendinden uyarlamalı süzgeçleri daha iyi anlayabilmek için doğrusal süzgeçleri ve rasgele süreci anlamak yardımcı olacaktır.

3. DOĞRUSAL SÜZGEÇLER VE RASGELE SÜREÇ

3.1 Genel

Rasgele süreç(Stokastik, stochastic process), istatiksel olgunun olasılık kurallarına göre zamanın bir fonksiyonu olarak açıklanmasıdır. İstatiksel olgunun doğasından dolayı, karşılaşılan problemle ilgili daha önceden bir deney yaşanmadıysa bu problemi zaman içinde açıklamak imkansızdır. Bu nedenle önceki deneylerden olasılık kurallarınca bilgi çıkarmak gerekir. Olasılık yoğunluk fonksiyonu bize bir işaret hakkında gelecekteki durumu hakkında bilgi verebilir ancak sadece uygun ölçümlerle yapılmış deneylerde bir rasgele sürece ilişkin olasılık yoğunluk fonksiyonunu hesaplamak mümkün değildir. Bu yüzden sürece ait bazı bilgileri elde etmek gerekir. Bunun için sürecin birinci ve ikinci momenti hesaplanmalıdır.

3.2 Rasgele Süreçlerin Özellikleri

3.2.1 Ortalama değer

$u(n), u(n-1), \ldots . u(n-M+1)$ şeklinde bir rasgele süreci gösteren bir dizi düşünürsek, bu sürecin ortalama değer fonksiyonu denklem 3.1 de gösterilmiştir.

$$\mu(n) = E[u(n)] \tag{3.1}$$

E' beklenti operatörüdür.

3.2.2 Otokorelasyon

$u(n), u(n-1), \ldots . u(n-M+1)$ şeklinde bir rasgele süreci gösteren bir dizi düşünürsek, bu sürecin otokorelasyon fonksiyonu(Öz İlişki) ise aşağıdaki gibidir.

$$r(n, n-k) = E[u(n)u^*(n-k)], \qquad k = 0, \pm 1, \pm 2, \ldots \ldots, \tag{3.2}$$

3.2.3 Çapraz korelasyon(Cross-Correlation)

$u(n), u(n-1), \ldots . u(n-M+1)$ ve $v(n), v(n-1), \ldots . v(n-M+1)$ şeklinde birer rasgele süreci gösteren dizileri düşünürsek, bu dizilerin çapraz korelasyon fonksiyonu aşağıdaki gibidir.

$$r_{u,v}(n, n-k) = E[u(n)v^*(n-k)] = r_{u,v}(k), \; k = 0, \pm 1, \pm 2, \ldots \ldots, \tag{3.3}$$

3.2.4 Otokovaryans

$u(n), u(n-1), \ldots . u(n-M+1)$ şeklinde bir rasgele süreci gösteren bir dizi düşünürsek, bu sürecin otokovaryans fonksiyonu ise aşağıdaki gibidir

$$c(n, n-k) = E\left[(u(n) - \mu(n))(u(n-k) - \mu(n-k))^*\right], \quad k = 0, \pm 1, \pm 2, \ldots\ldots, \tag{3.4}$$

3.1 ve 3.3 denklemlerini göz önüne alırsak ortalama değer, korelasyon ve koveryans arasındaki ilişki aşağıdaki gibi olur

$$c(n, n-k) = r(n, n-k) - \mu(n)\mu^*(n-k) \tag{3.5}$$

Bir sürecin kısmi karakterizasyonu için ortalama değer fonksiyonu ve otokorelasyon yada oto koveryans fonksiyonunu n ve k'nın ilgilenilen çeşitli değerleri için belirlemek gerekir.

Bu durumun bize sunduğu iki avantaj vardır

1- Bu sayede kısmi ölçümler yeterli olur

2- Rasgele süreçlerde doğrusal işlemler için bu durum avantajlar sunar

3.2.5 Özel durumlar

Eğer süreç durağan ise

$$\mu(n) = \mu \tag{3.6}$$

yazılabilir. Ayrıca bu durumda

$$r(n, n-k) = r(k) \tag{3.7}$$

$$c(n, n-k) = c(k) \tag{3.8}$$

yazılabilir. n ve n-k gözlemleme zamanlarıdır.

Eğer k=0 ise, karesel ortalama aşağıdaki gibi olur.

$$r(0) = E\left[|u(n)|^2\right] \tag{3.9}$$

değişinti ise aşağıda gösterilmiştir.

$$c(0) = \sigma_u^2 \tag{3.10}$$

Denklem 3.9 değişintiyi (variance) gösterir. Denklem 3.7 ve 3.8'nin sağlanıyor olması sürecin durağan olduğu anlamına gelmez. Fakat durağan olmayan ama bu iki koşulu sağlayan süreçler için geniş anlamda durağandır deyebiliriz.

Ayrıca değişinti aşağıdaki gibi hesaplanabilir.

$$\sigma_u^2(n) = E\left[\left|u(n)\right|^2\right] - \left|\mu_u(n)\right|^2 \tag{3.11}$$

Eğer bir süreçte aşağıdaki eşitlik bütün n'ler için sağlanıyorsa

$$E\left[\left|u(n)\right|^2\right] < \infty \tag{3.12}$$

Genel anlamda süreç durağandır. Eğer geniş anlamda durağan bir süreçte $\mu = 0$ ise otokorelasyon ve otokovaryans fonksiyonları birbirine eşittir.

Sonuç olarak süreci kısmi olarak tanımlaya bilmek için gerekli birinci moment ortalama değer, ikinci moment ise karesel beklendik değerdir.

3.3 Geniş Anlamda Durağanlık

Geniş anlamda durağanlık önemli bir kavramdır o yüzden burada tekrar değinilmiştir. $u(n), u(n-1), \ldots . u(n-M+1)$ şeklinde bir stokastik süreci gösteren bir dizi düşünürsek, bu sürecin

1- Bir sürecin ortalama değeri $\mu = 0$

2- Bu sürecin değişintisi $\sigma_u^2 = sabit$

3- ve $c(n, n-k) = c(k)$

Koşulları sağlanıyorsa $u(n)$ süreci geniş anlamda durağandır denir. Bu terim yerine zayıf durağan ikinci dereceden durağan terimleri de kullanılabilir.

3.4 Korelasyon Matrisi

$u(n), u(n-1), \ldots . u(n-M+1)$ şeklinde bir rasgele süreci gösteren bir dizi $U(n)$ düşünürsek, M tane $U(n)$ dizisinden oluşan bir gözlem kümesi aşağıdaki gibi yazılır

$$U^T(n) = \left[u(n), u(n-1), \ldots . u(n-M+1)\right] \tag{3.13}$$

Bu sürece ait korelasyon matrisi R ile gösterilir ve aşağıdaki gibi tanımlanır.

$$R = E\left[u(n)u^{H}(n)\right] \tag{3.14}$$

H burada Hermit (Hermitian) operatörüdür. Eşlenik evrik almak için kullanılır.

Eğer denklem 3.11 ve 3.12 i geniş anlamda durağan koşulu için düşünürsek aşağıdaki gibi yazabiliriz.

$$R = \begin{bmatrix} r(0) & r(1)..... & r(M-1) \\ r(-1) & r(0)..... & r(M-2) \\ : & : & : \\ r(-M+1) & r(-M+2).. & r(0) \end{bmatrix} \tag{3.15}$$

Köşegen üzerindeki r(0) değerleri her zaman gerçektir. Geri kalan değerler sanal olabilir.

3.5 Güç Spektral Yoğunluğu

$\{r(k)\}$, $k = 0,\pm1,\pm2,.....,(M-1)$, şeklindeki korelasyon(öz ilişki) dizisi yada MxM şeklindeki R korelasyon matrisi, geniş anlamda durağan rasgele sürecin zaman düzlemindeki tanımıdır. Zaman düzlemindeki $\{r(k)\}$ dizisine Ayrık zaman Fourier transformu uygulanırsa:

$$S(w) = \sum_{k=-(M-1)}^{M-1} r(k)e^{-jwk}\ ,\ -\pi \le w \le \pi \tag{3.16}$$

w' açısal frekans olmak üzere bulunan eşitlik güç spektral yoğunluğu yada sürecin güç spektrumudur. Böylece geniş anlamda durağan bir süreci frekans düzleminde güç spektral yoğunluğu ile açıklamak mümkün olur.

Ayrıca ters ayrık zaman fourier dönüşümünü güç spektral yoğunluğuna uygulayarak $\{r(k)\}$ öz ilişki dizisini tekrar elde edebiliriz.

$$r(k) = \frac{1}{2\pi}\int_{-\pi}^{\pi} s(w)e^{jwk}dw,\ k = 0,\pm1,\pm2,.....,(M-1), \tag{3.17}$$

3.5.1 Güç spektral yoğunluğunun özelliği

1- Durağan ayrık zaman rasgele sürecin güç spektral yoğunluğu periyodiktir.

$$S(w+2k\pi)=S(w),\ T=2\pi \tag{3.18}$$

2- Durağan ayrık zaman rasgele sürecin güç spektral yoğunluğu gerçektir(real).

$$S(w)=r(0)+\sum_{k=1}^{M-1}r(k)e^{-jwk}+\sum_{k=-(M-1)}^{-1}r(k)e^{-jwk} \tag{3.19}$$

$$r(-k)=r^{*}(k) \tag{3.20}$$

ise ve k=-k alarak

$$S(w)=r(0)+2\sum_{k=1}^{M-1}\mathrm{Re}\left[r(k)e^{-jwk}\right], \tag{3.21}$$

3- Eğer süreç gerçek ise $S(-w)=S(w)$ aksi taktirde güç spektral yoğunluğu çift değildir.

4- Durağan ayrık zaman rasgele sürecin güç spektral yoğunluğu pozitif değerler alır.

3.6 Doğrusal Süzgeçler

3.6.1 Genel

Bu başlık altına kadar anlatılanlar, süzgeç teoreminde kullanılan yardımcı araçlardır. Doğrusal süzgeçler, çıkış işaretinin girişe bağımlı olduğu ve girişin doğrusal bir fonksiyonu olduğu süzgeçlerdir.

İstatiksel yaklaşımda doğrusal süzme problemi için istenilen işaretin ve istenmeyen işaretin bazı parametrelerinin(ortalama, korelasyon fonksiyonları..) ve süzgecin ne için çalışacağının bilinmesi gereklidir. Buradaki amaç çıkışta gürültünün etkilerinin azaltılmasıdır. Bu durumda en iyi yaklaşımlardan biri çıkışta oluşan işaret ile olması istenen işaretin farkının karesel ortalamasının azalmasını sağlamak olacaktır. Durağan işaretler için genel çözüm Wiener süzgeçleri olarak bilinir. Bu süzgeçler karesel ortalama dikkate alındığında optimum çözümü üretirler.

Eğer süreç durağan değilse bu durumda daha gelişmiş süzgeç olan Kalman süzgeçleri kullanılabilir. Aslında Wiener süzgeçleri Kalman Süzgeçlerinin özel durumu olarak kabul edilebilir. Fakat Kalman süzgeçleri donanım yönünden oldukça sınırlayıcı özelliklere sahiptir. Kendinden uyarlamalı süzgeçlere geçmeden önce fikir vermesi açısından doğrusal süzgeçler

için Wiener süzgeçlerini incelemek faydalı olacaktır.

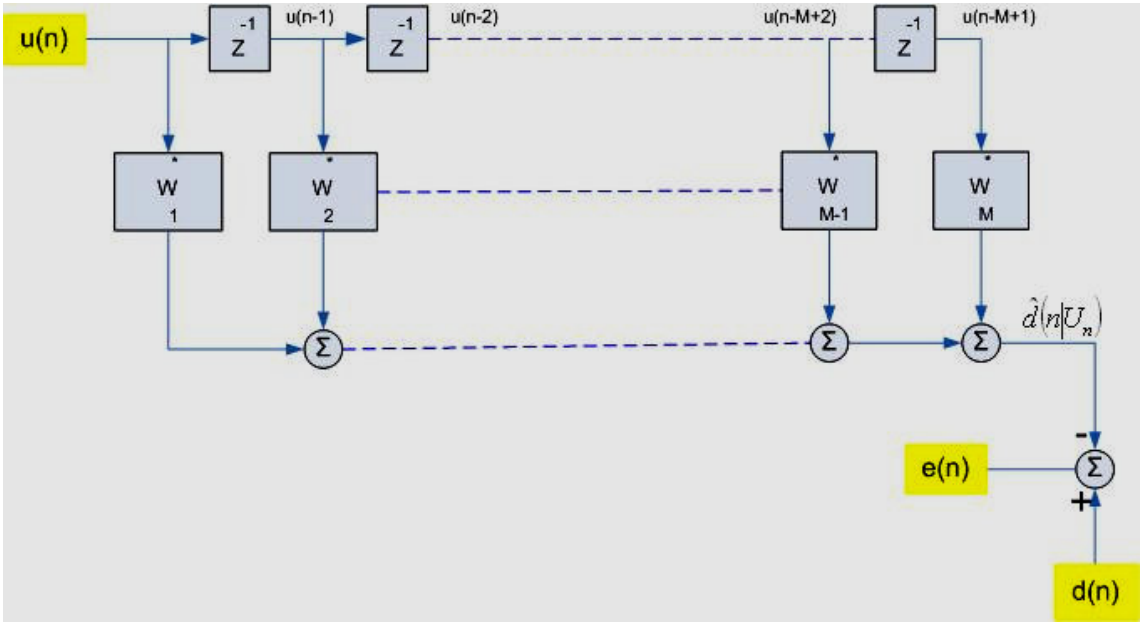
3.6.2 Wiener süzgeçler

Aşağıdaki şekil 3.1 de tasarlanmış enine(Transversal) süzgeci düşünürsek, süzgecin basitçe üç operasyondan oluştuğunu görebiliriz.

1- depolama

2- çarpma

3- toplama



Şekil 3.1 Enine süzgeç

Depolama: Depolama arka arkaya bağlı M-1 bir işaret örneğini saklayabilen gecikme ünitesi olarak düşünülebilir. M-1 ünite M-1 adet tap oluşturur, bu taplar M-1 tane giriş örneğini saklar dolayısıyla, $u(n), u(n-1), ..., u(n-M+1)$ giriş örneğinden u(n) o anki girişi diğerleri de u(n)'in önceki değerlerini göstermek üzere taplarda saklanır.

Çarpma: Çarpma işlemi tap girişleri $u(n), u(n-1), ..., u(n-M+1)$ ile tap ağırlıkları $w_1, w_2, ..., w_M$ arasında içsel çarpım olarak gerçekleştirilir bu da bir dizi çarpıcı kullanımı gerektirir. Çarpım sonucu $w_1^*, w_2^*, ...w_M^*$ içerisine dahil edilmiştir.

Toplama: Buradaki toplayıcıların fonksiyonu çarpıcıların çıkışlarındaki bilgilerin toplanıp sonucun elde edilmesidir.

Enine süzgeçteki gecikme ünitelerinin sayısı süzgecin derecesini gösterir.şekil 3.1 deki süzgecin derecesi M-1 dir. Ayrıca enine süzgecin dürtü(impuls) yanıtı $\{h_k\} = \{w_k\}$, k=1,2,…,M olur.

Burada genel olarak tap girişleri $u(n), u(n-1), ..., u(n-M+1)$'nin gösterdiği süreci geniş anlamda durağan ve sürecin ortalama değerini '0' kabul ederiz.

Şekil 3.1 de filtre çıkışı $\hat{d}(n|U_n)$ ile gösterilmiştir. Bu çıkış aslında olması istenen d(n) işaretinin tahminidir. Bu çıkışın oluşmasında en büyük etkiye tap ağırlıkları sahiptir. Bu yüzden çıkışta oluşan tahmini değerin gerçeğe yakın olması için tap ağırlıklarının iyi ayarlanması gerekir bu işlem de $u(n), u(n-1), ..., u(n-M+1)$ işaretine ilişkin istatiksel bilginin elde edilmiş olması ile alakalıdır. Süzgecin çıkışı tap girişlerinin ve dürtü yanıtının (tap ağırlıklarının) konvolüsyon toplamı şeklinde hesaplanabilir.

$$\hat{d}(n|U_n) = \sum_{k=1}^{M} w_k^* u(n-k+1) \tag{3.22}$$

Bu adımdan sonra süzgecin, istenilen cevap d(n) ile süzgeç çıkışında oluşan $\hat{d}(n|U_n)$ farkının herhangi bir n'zamanında en küçük değeri verecek şekilde tasarlanması gerekir.

$$e(n) = d(n) - \hat{d}(n|U_n) \tag{3.23}$$

Bu formül kestirim hatası olarak adlandırılabilir. Wiener teorisine göre süzgeci en iyi durum için ayarlamak üzere en küçük ortalama karesel hata(minimum mean-square error criterion) kriteri kullanılır. Özel olarak, tap ağırlıkları $w_1, w_2, ..., w_M$, $J(w)$'performans indeksini en küçük değerde tutabilecek şekilde seçilmelidir. Bu indeks, kestirim hatasının karesel ortalamasıdır, yada ortalama karesel hata olarak bilinir.

$$J(w) = E\left[e(n)e^*(n)\right] \tag{3.24}$$

Buradaki w tap ağırlıkları $w_1, w_2, ..., w_M$ dir. Ortalama karesel hata gerçek ve pozitiftir. $J(w)$'i en küçük değerde tutabildiğimiz zaman en iyi yada en uygun doğrusal süzgeci minimum karesel ortalama hatasına göre elde etmiş oluruz.

3.7 Özet

Şimdiye kadar anlatılanlardan anlaşıldığı üzere doğrusal süzgeçlerde en uygun çözümü elde etmek için

1- Ortamdaki gürültünün istatiksel bilgisi

2- Elde edilmek istenen işaretin istatiksel bilgisi

3- Süzgecin hangi amaç için çalışacağı bilgisi

4- Bu bilgiler ışığında tap katsayıların oluşturulması

İzlenecek adımlar ve elde edilmesi gereken bilgilerdir. Bu bilgilerin nasıl kullanılacağı ve bu kat sayıların oluşturulması bu tezin kapsamı içerisinde olmadığından Wiener Süzgeçleri incelemeyi burada sonlandırmak uygun olacaktır. Burada belirtilen, doğrusal bir süzgeç tasarlayabilmek için yukarıda sıralanan dört maddeye uymak gerekliliği ve istatiksel bilgiyi edinmeden yada yanlış edinilmiş bilgilerle başarılı bir süzme işleminin yapılamayacağıdır.

4. KENDİNDEN UYARLAMALI SÜZGEÇLER

4.1 Genel

Önceki bölümde anlatılan Wiener süzgeçleri, işlem görecek bilginin öncül istatiksel bilgilerine ihtiyaç duyar. Bu tip süzgeçler, ancak girişin istatiksel bilgileri ile öncül istatiksel bilgilere göre tasarlanan süzgecin bilgileri birbirine eş olduğunda en iyi performansı verirler. Bu bilgilerin bulunmadığı durumlarda süzgeç olumsuz sonuçlar verir, buna rağmen bu durumu düzeltmek için çeşitli teknikler geliştirilmiştir. Bu tekniklerden birisi kendinden uyarlamalı süzgeçler kullanılmaktır. Kendinden uyarlamalı süzgeçler ilgili işaretin karakteristik bilgisinin ortamda bulunmadığı durumlarda yinelemeli(Recursive) algoritmasıyla, kendi kendini bulunduğu ortama göre ayarlayarak istenilen çıkışın oluşturulmasını sağlar. Bu algoritma ilk başlangıç koşullarıyla başlatılarak birkaç denemeden sonra istenilen çıkışı vermeye başlar.

Kendinden uyarlamalı süzgeçlerde oldukça fazla seçenek vardır. Doğru algoritmanın seçimi önemlidir. Bu seçimi yaparken aşağıdaki faktörleri dikkatle değerlendirmek gerekir.

1- Yakınsama oranı (Rate of Convergence): algoritma için gerekli deneme sayısıyla ifade edilir. Durağan girişler için yakınsama en iyi Wiener çözümüne oldukça yakındır. Hızlı bir yakınsama oranı algoritmanın, durağan çevreye bilinmeyen istatiksellere rağmen hızlı bir şekilde kendini uydurmasına imkan sağlar. Daha fazlası, durağan olmayan çevrelerde algoritma istatiksel değişkenleri izleyebilir.

2- Sağlamlık(Robustness): kötü elde edilmiş giriş verisi üzerinde algoritmanın yeterince başarılı bir şekilde çalışmasıyla ilgilidir.

3- İşlemsel gereksinimler(Computational Requirements): süzgecin içindeki işlemler, bu işlemleri yapabilmek için gerekli bellek, ve bunları birleştirerek programlayabilme bu parametreyi oluşturur.

4- Yapı (Structure): algoritmanın gerçekleştirileceği yapıda başka bir parametredir.

5- Sayısal özellikler(Numerical Properties): Yuvarlama hatası, örnekleme hatası gibi hatalar yüzünden, bir süzgeci sayısal olarak gerçekleştirirken geri beslemeden dolayı çıkışta çok büyük etkiler görülebilir.

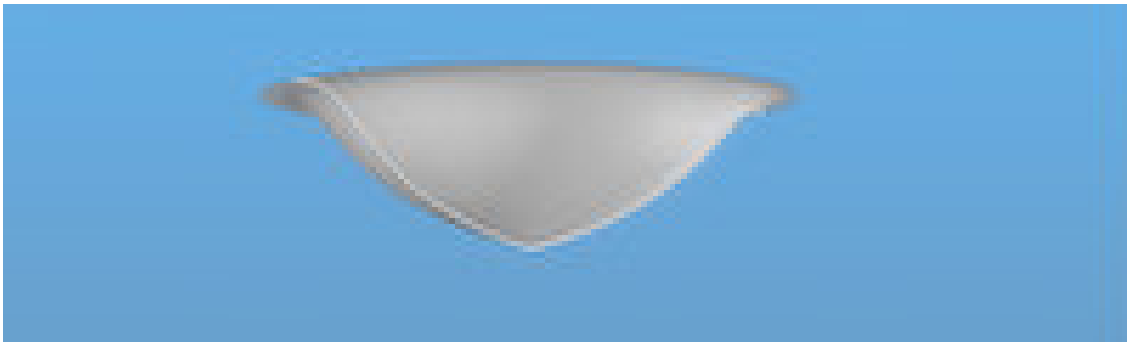
Bu faktörler kendinden uyarlamalı süzgeç tasarlamada parametrelerin ve algoritmaların seçiminde göz önünde bulundurulmalıdır.

4.2 Kendinden Uyarlamalı Süzgeç Tasarlamada Yaklaşımlar

Kendinden uyarlamalı süzgeçlerde sadece bir tip çözümden bahsetmek yanlış olur. Birden fazla kendinden uyarlamalı algoritma ile her biri farklı ancak istenen özellikleri karşılamak kaydıyla değişik kendinden uyarlamalı süzgeçler oluşturulabilir. Farklı kendinden uyarlamalı süzgeçler için gerekli algoritmaları üç ana grup altında toplayabiliriz.

1- Wiener Süzgeç teorisine dayalı yaklaşım: taplı gecikme hattı veya enine süzgeç kullanılır. Böyle bir süzgecin sonlu dürtü (impuls) cevapları bir grup tap ağırlıkları ile tanımlanır. Durağan girişler için ortalama karesel hata enine süzgecin tap ağırlıklarının ikinci dereceden fonksiyonudur.

Ortalama karesel hatanın bilinmeyen tap ağırlıklarına bağlılıkları tek bir minimum noktalı parabole benzetilebilir şekil 4.1. Bu parabole hata başarım yüzeyi denir. Yüzeyin minimum noktasına ait tap ağırlıkları en iyi Wiener çözümünü belirler.



Şekil 4.1 Hata başarım yüzeyi

Kendinden uyarlamalı enine süzgece ait ağırlıkları değiştirmek için iki aşamalı işlem yapılır. İlk aşamada en dik azalış yöntemini(steepest descent) kullanarak en iyi Wiener çözümünü belirleyen matris denklemi düzenlenir. Bu düzenleme için gradient vektörünün kullanımı gerekir. Gradient vektörü iki parametreye bağlıdır. enine süzgecin girişlerinin taplarının korelasyon matrisi ve istenilen cevap ile aynı giriş tapları arasındaki çapraz korelasyon vektörü. İkinci aşama olarak, bu korelasyonlar için anlık değerler kullanarak gradient vektörü için bir kestirim(estimate) yürütülür. Bu algoritma genel olarak en küçük karesel ortalama hata(Least Mean Square, LMS) olarak bilinir. LMS, oldukça iyi bir algoritma olmasına rağmen, yavaş yakınsama oranı algoritmanın istenmeyen özelliğidir.

Durağan olmayan durumlarda, hata başarım yüzeyinin minimumu zamanla sürekli değişir. Bu durumda LMS algoritmasına hata başarım yüzeyinin minimumunu sürekli izleme görevi

eklenir. Bu takip görevi LMS algoritmasının öğrenme oranına kıyasla giriş verisinin yavaşça değiştiği durumlarda gerçekleşir.

2- Kalman Süzgeç teorisine dayalı yaklaşım: kendinden uyarlamalı süzgeci oluşturmak için diğer bir yöntem Kalman süzgeçleridir. Durağan veya durağan olmayan çevrelerin seçimine göre Kalman Süzgeci kendinden uyarlamalı süzgeci oluşturabilir fakat bu tip süzgeçlerin yapısı Wiener Süzgeçlere göre daha karışıktır.

3- En az Kareler yöntemi: Daha önce bahsedilen yöntemler istatiksel kavrama bağlıydı. En az kareler yöntemi ise diğer ikisinden farklı olarak başlangıçtan itibaren tamamen deterministliktir.

4.3 Kendinden Uyarlamalı Süzgeç Uygulama Örneği

Kendinden uyarlamalı süzgeçlerin yapısına geçmeden önce az ileride tasarlanacak olan bu tip süzgeçlerle neler yapılır, nelerde kullanılır sorularına örnekle cevap vermek konuyu anlamada yardımcı olabilir.

Kendinden uyarlamalı süzgeçler bilinmeyen çevrelerde ve girişin zamanla değiştiği durumlarda işaret işleme ve kontrol uygulamaları için kabiliyetleri göz önüne alındığında çok önemli süzgeçlerdir. Kendinden uyarlamalı süzgeçler haberleşmede, radarlarda, sonarlarda, sismolojide ve biyomedikal mühendisliğinde başarıyla kullanılmaktadır.

Bu uygulamalar, doğal olarak birbirinden oldukça farklı olsalar da aslında ortak bir özellikleri vardır. Hata hesaplamada kullanılan giriş vektörü ve istenilen cevap ile bunların geri beslemeyle süzgecin ağırlık değerlerini değiştirmesi. Bu çalışma da kendinden uyarlamalı süzgeçlerin doğasına uygun olan bir amaç doğrultusunda hazırlanmıştır. Çalışma sonucunda oluşacak çıktılar gürültü bastırma adı altında toplanacak uygulamalarda kullanılabilir.

4.3.1 Kendinden uyarlamalı süzgeçlerde gürültü bastırma

Adından da anlaşılacağı gibi kendinden uyarlamalı süzgec ile gürültü bastırma, SNR(Signal to Noise Ratio) işaret gürültü oranını yükseltmek amacıyla işaretten gürültünün çıkarılması için yapılan bir işlemdir.

Gürültü işaretinin alınan işaretten doğrudan çıkartılmaya çalışılması daha kötü sonuçlar doğurur. Alınan işaretin doğrudan süzülmesi, süzgecin hem gürültüyü hemde bilgi işaretini değiştirmesine sebeb olur. Bu işlem sonunda gürültü azalabilir fakat bilgi işareti de

değiştirilmiş olur. Bu yüzden süzme ve çıkarma kendinden uyarlamalı bir işlemle yapılırsa daha iyi sonuçlar elde edilebilir.

Temel olarak kendinden uyarlamalı gürültü bastırıcılar çift girişlidirler ve kapalı geri besleme sistemleri vardır. Bu girişlerden biri ana giriş (Primary) , diğer giriş ise referans(refence) girişidir.

Referans girişte: bilgiyi taşıyan s(n) işareti ve bozucu gürültü işareti $v_0(n)$ bulunur.

$$d(n) = s(n) + v_0(n) \tag{4.1}$$

s(n) ve $v_0(n)$ birbirinden bağımsızdır. Bu durumda:

$$E[s(n)v_0(n-k)] = 0 \text{ her k için} \tag{4.2}$$

Denklem 4.2 s(n) ve $v_0(n)$'in gerçek kabul edildiği durumlarda doğrudur.

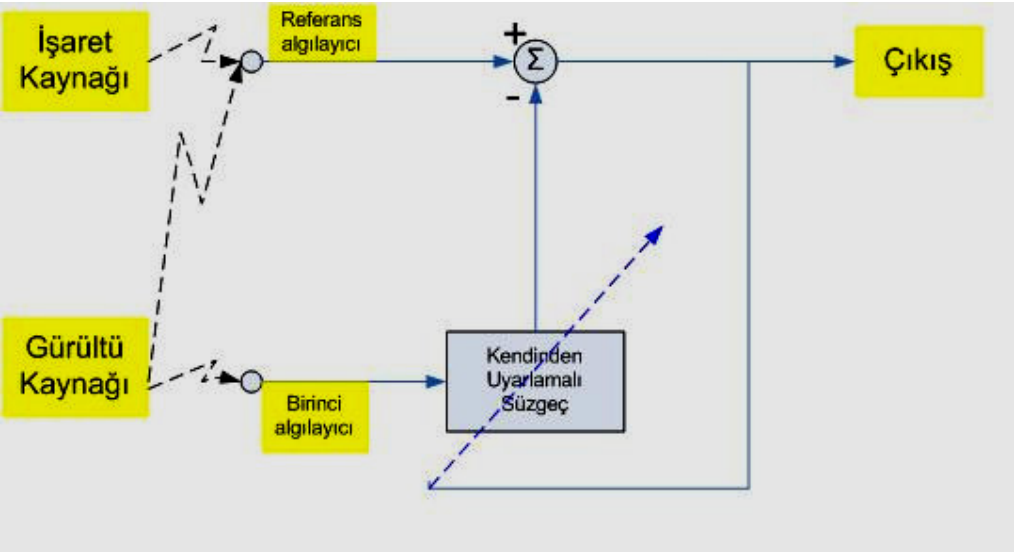
Ana girişte ise s(n)'den bağımsız fakat $v_0(n)$ ilişkili $v_1(n)$ işareti bulunur.

$$E[s(n)v_1(n-k)] = 0 \text{ her k için ve} \tag{4.3}$$

$$E[v_0(n)v_1(n-k)] = p(k) \text{ her k için} \tag{4.4}$$

Burada bütün işaretler gerçektir ve p(k) k. periyot için bilinmeyen çapraz korelasyondur. $v_1(n)$ işareti kendinden uyarlamalı süzgeçte işlenerek çıkış işaretinin oluşturulmasında rol oynar.

Aşağıdaki şekil 4.2'de kendinden uyarlamalı gürültü bastırma kavramı gösterilmiştir. Bu tip süzgeçler günlük hayatta uçak kokpitlerinde haberleşme amaçlı, ağır vasıtalarda haberleşme amaçlı, gürültü oranı yüksek işletmelerde haberleşme amaçlı kullanılabilir. Kısacası bu tip süzgeçleri genel olarak, gürültüden arındırılacak bir işaret olduğu yerde yada süzgeci özelleştirerek daha özel uygulamalarda kullanmak mümkün olmaktadır.



Şekil 4.2 Kendinden uyarlamalı gürültü bastıran süzgeç(Haykin, 1986)

4.4 Kendinden Uyarlamalı Süzgecin Yapısı

Daha önceki bölümlerde bahsedildiği gibi kendinden uyarlamalı süreç iki temel olaya dayalıdır.

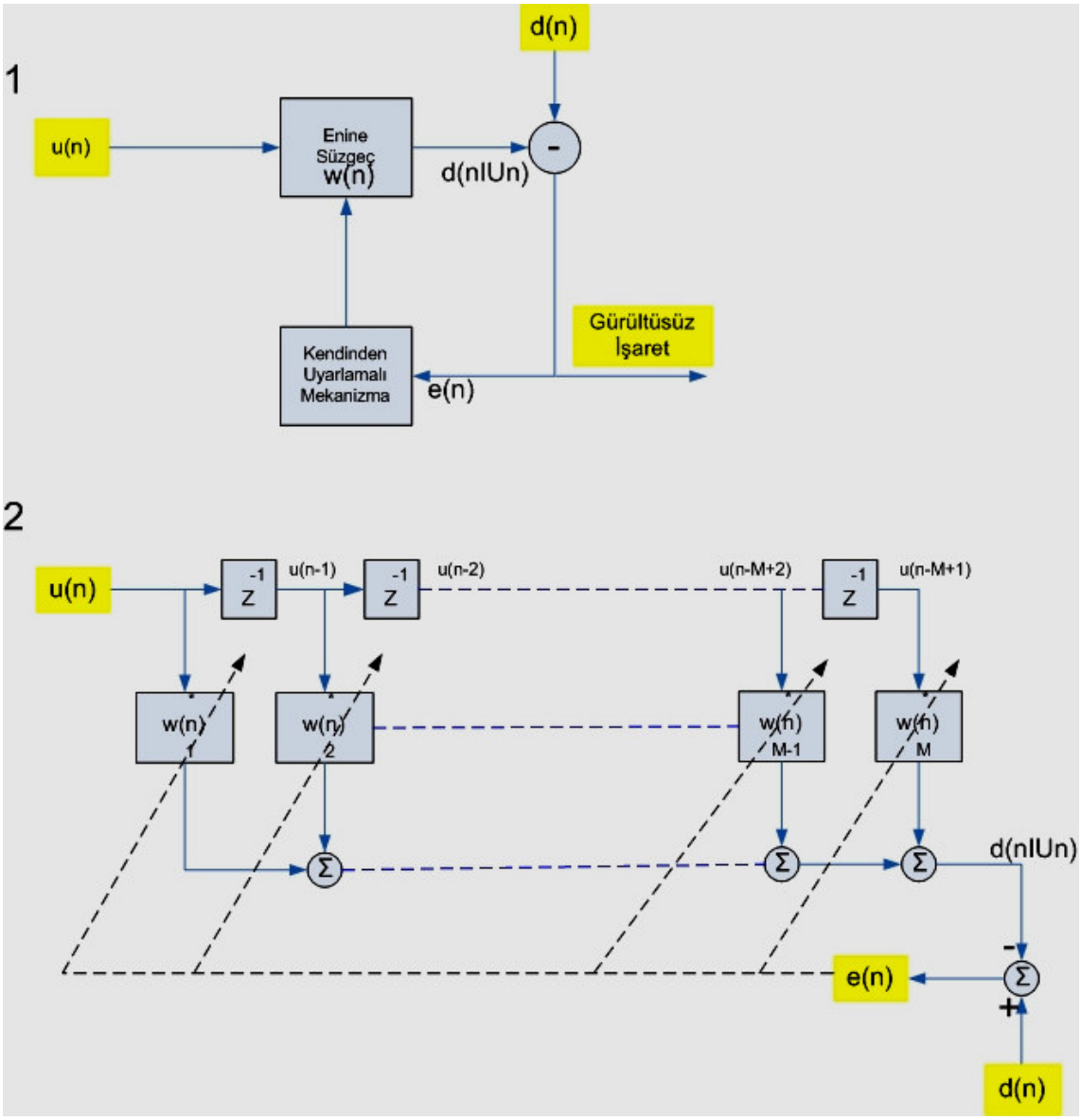
1-süzme işlemi: tap ağırlıkları ile oluşturulan enine süzgecin çıkışının hesaplanmasıyla, bu oluşturulan çıkışla istenilen cevabın karşılaştırılarak hata tahmininin üretilmesi işlemidir.

2-Uyarlamalı işlem: hataya göre tap ağırlıklarının otomatik olarak güncellenmesi işlemidir.

Bu iki temel faktör kendinden uyarlamalı süzgeçler için geçerlidir. Bu çalışmada buradan yola çıkarak LMS algoritması incelenecektir fakat öncesinde, anlamada kolaylık sağlaması açısından ve LMS'i açıklarken bazı özelliklerini kullanmamızın mümkün olduğu en dik azalış(Steepest Descent Algorithm) yöntemi inceleme altına alınacaktır, sonrasında ise LMS anlatılacaktır, fakat bu adımdan önce temelleri oluşturması için bazı terimleri açıklamak gerekir.

Aşağıdaki şekil 4.3 incelenirse kendinden uyarlamalı süzgecin iki parçadan oluştuğu görülür. 1- ağırlıkları güncellenebilir Enine süzgeç. Her hangi bir n anında süzgeç ağırlıkları $w_1(n), w_2(n), ..., w_M(n)$ olmak üzere.

2- ağırlıkları kendinden uyarlamalı bir şekilde güncelleyecek mekanizma



Şekil 4.3 Kendinden uyarlamalı süzgecin yapısı(Haykin, 1996)

Süzme işlemi süresince istenilen cevap (desired response) d(n) normal tap girişleri ile birlikte sisteme uygulanır. Aslında istenilen cevap ağırlıkların güncellenmesinde referans olarak görülebilir. n' zamanında tapların girişindeki vektöre u(n) ve bu vektörün taplara uygulanmasıyla süzgeç çıkışında oluşan işarete $\hat{d}(n|U_n)$ denir. Buradaki U_n, $u(n), u(n-1), \ldots . u(n-M+1)$ ile oluşturulmuştur. Süzgeç çıkışı ile istenilen cevap karşılaştırıldığında kestirim hatası e(n) oluşur.

$$e(n) = d(n) - \hat{d}(n|U)$$

$$= d(n) - w^H(n)u(n) \tag{4.5}$$

$w^H(n)u(n)$, tap ağırlık vektörü w(n) ile giriş vektörü u(n)'in iç çarpımıdır. Tap ağırlık vektörünün açılmış şekli aşağıdaki gibidir.

$$w^T(n) = [w_1(n), w_2(n), ..., w_M(n)] \tag{4.6}$$

Aynı şekilde tap giriş vektörü

$$u^T(n) = [u(n), u(n-1), ..., u(n-M+1)] \tag{4.7}$$

Eğer tap giriş vektörü u(n) ile istenilen cevap d(n) karşılıklı durağan ise, ortalama karesel hata(mean-squared error) J(n) n' zamanında aşağıdaki gibi olur(Haykin, 1986).

$$J(n) = \sigma_d^2 - w^H(n)p - p^H w(n) + w^H(n)Rw(n) \tag{4.8}$$

Burada

σ_d^2 =istenilen cevap d(n) 'in değişintisi

p= tap giriş vektörü u(n) ile istenilen cevap d(n) arasındaki çapraz korelasyon vektörü

R= tap giriş vektörü u(n)'in korelasyon matrisidir.

Denklem 4.8'in nasıl bulunduğunu biraz açmak gerekirse:

$$\hat{d}(n|U_n) = w^H u(n) \tag{4.9}$$

Her iki tarafın Hermit transpozunu alırsak

$$\hat{d}^*(n|U_n) = u^H(n)w \tag{4.10}$$

Elde edilir.

Bu durumda tahmini hatayı aşağıdaki gibi gösterirsek

$$e(n) = d(n) - w^H u(n) \tag{4.11}$$

Kompleks eşleniği ise

$$e^*(n) = d^*(n) - u^H(n)w \tag{4.12}$$

Elde edilir, denklem 4.11 ve 4.12 ‘i denklem 3.24’de yerine yazarsak

$$J(w) = E\left[\left(d(n) - w^H u(n)\right)\left(d^*(n) - u^H(n)w\right)\right] \tag{4.13}$$

w’ tap ağırlıkları sabit sayılar olduğundan

$$J(w) = E\left[d(n)d^*(n)\right] - w^H E\left[u(n)d^*(n)\right] - E\left[d(n)u^H(n)\right]w + w^H E\left[u(n)u^H(n)\right]w \tag{4.14}$$

Elde edilir. u(n) ve d(n) karşılıklı durağan olduğu için aşağıdaki dört durum dikkate alınır:

1- $E\left[d(n)d^*(n)\right]$’in beklentisi d(n) beklentisine eşit olur ve d(n) sıfır ortalamalı olduğu kabul edilirse:

$$\sigma_d^2 = E\left[d(n)d^*(n)\right] \tag{4.15}$$

2- $E\left[u(n)d^*(n)\right]$ beklentisi Mx1 boyutunda tap giriş vektörü u(n) ile istenilen cevap d(n) arasındaki çapraz korelasyon vektörü p ile gösterilirse.

$$\text{p=}\ E\left[u(n)d^*(n)\right] \tag{4.16}$$

açık şekilde yazılırsa

$$p^T = \left[p(0), p(-1),..., p(1-M)\right] \tag{4.17}$$

Ayrıca burada

$$p(1-k) = E\left[u(n-k+1)d^*(n)\right], \text{ k=1,2,…,M} \tag{4.18}$$

3- $E\left[d(n)u^H(n)\right]$ beklentisi p’ çapraz korelasyon vektörünün hermit tersidir

$$p^H = E\left[d(n)u^H(n)\right] \tag{4.19}$$

4- $E\left[u(n)u^H(n)\right]$ beklentisi MxM boyutunda tap giriş vektörü u(n)’in korelasyon matrisidir.

$$R = E\left[u(n)u^H(n)\right] \tag{4.20}$$

Sonuç olarak denklem 4.15, 4.16, 4.19, 4.20 ve 4.14 birleştirilirse denklem 4.8 aşağıdaki gibi elde edilir.

$$J(w) = \sigma_d^2 - w^H p - p^H w + w^H R w \tag{4.21}$$

Buradan özet olarak ortalama karesel hatanın (J(n)) w(n)'e bağlılığını, tek bir minimum noktalı kase şeklinde düşünebiliriz. Bu yüzeye kendinden uyarlamalı süzgecin hata-performans yüzeyi denir. Kendinden uyarlamalı sürecin görevi bu yüzeyin en altını yada minimumunu takip etmektir. Bu minimum noktada tap ağırlık vektörü en uygun değerleri almıştır. Bu değerleri w_0 ile gösteririz. Normal denklemine göre (Haykin, 1986)

$$Rw_0 = p \tag{4.22}$$

elde edilir bu denklemin nereden geldiğini anlamak için bir vektöre göre türev konusuna kısaca değinmek gerekir.

4.5 Bir Vektöre Göre Türev

Bu konuyu açıklarken g'nin Mx1 boyutundaki vektör w'nin bir fonksiyonu olduğunu düşünelim eğer w vektörünü

$$w_k = a_k + jb_k \tag{4.23}$$

Olarak düşünürsek k=1,2,…,M ise g, 2M değişkenli a_k, b_k den oluşan bir fonksiyon olarak düşünülebilir. Bu durumda g'nin w'e göre türevini Mx1 boyutunda bir vektör olarak düşünebiliriz.

$$\frac{dg}{dw} = \begin{bmatrix} \frac{\partial g}{\partial a_1} + j\frac{\partial g}{\partial b_1} \\ \frac{\partial g}{\partial a2} + j\frac{\partial g}{\partial b2} \\ \vdots \\ \frac{\partial g}{\partial a_M} + j\frac{\partial g}{\partial b_M} \end{bmatrix} \tag{4.24}$$

Denklem 4.21 ile tanımlanan ortalama karesel hata J(w) 'i denklem 4.24 kullanarak minimum noktasına getirmenin yollarını aranabilir. Bunu yapmak için anlamada kolaylık sağlaması için üç örnek ile normal denklemine gitmek doğru olur.

1- g fonksiyonu aşağıdaki gibi tanımlansın:

$g = c^H w$, burada c ve w' Mx1 boyutunda vektördür. g'i tekrar yazarsak

$$g = \sum_{k=1}^{M} c_k^* w_k \\ = \sum_{k=1}^{M} c_k^* (a_k + jb_k) \tag{4.25}$$

Bu durumda

$$\frac{\partial g}{a_k} = c_k^*, k = 1,2,..,M \\ \frac{\partial g}{b_k} = jc_k^*, k = 1,2,..,M \tag{4.26}$$

elde edilir. Denklem 4.26'i 4.24'te yerine yazarsak

$$\frac{dg}{dw} = c_1^* + j^2 c_1^* .... \\ = \frac{d\left(c^H w\right)}{dw} = 0 \tag{4.27}$$

elde edilir.

2- g fonksiyonu aşağıdaki gibi tanımlansın:

$g = w^H c$, burada c ve w' Mx1 boyutunda vektördür. g'i tekrar yazarsak:

$$g = \sum_{k=1}^{M} c_k w_k^* \\ = \sum_{k=1}^{M} c_k (a_k - jb_k) \tag{4.28}$$

Bu durumda

$$\frac{\partial g}{a_k} = c_k, k = 1,2,..,M \\ \frac{\partial g}{b_k} = -jc_k, k = 1,2,..,M \tag{4.29}$$

elde edilir. Denklem 4.29'i 4.24'te yerine yazarsak

$$\frac{dg}{dw} = c_1 - j^2 c_1 \ldots\ldots$$
$$= \frac{d\left(w^H c\right)}{dw} = 2c \tag{4.30}$$

elde edilir.

3- g fonksiyonu aşağıdaki gibi tanımlansın:

$g = w^H Q w$, burada w' Mx1 boyutunda vektördür Q MxM matristir.

$c_1 = Q^H w$
$c_1^H = w^H Q$ olsun bu durumda $g = c_1^H w$ olur. Öyleyse c_1 sabit olmak üzere

$$\frac{dg}{dw} = c_1^* + j^2 c_1^* \ldots.$$
$$= \frac{d\left(c_1{}^H w\right)}{dw} = 0 \tag{4.31}$$

bulunur. Aynı şekilde $c_2 = Qw$ olsun bu durumda $g = w^H c_2$ yazılırsa, c_2 sabit olmak üzere:

$$\frac{dg}{dw} = c_1 - j^2 c_1 \ldots\ldots$$
$$= \frac{d\left(w^H c_2\right)}{dw} = 2c_2 \tag{4.32}$$

bulunur. 4.31 ile 4.32' i birleştirerek sonuç olarak

$$\frac{d}{dw}(w^H Q w) = 2Qw \tag{4.33}$$

bulunur. Bu üç örnek yardımıyla, J(w)'nin w'e göre türevini alırsak:

$\frac{d}{dw}(\sigma_d^2) = 0$, σ_d^2=sabit

$\frac{d}{dw}(p^H w)$=0

$\frac{d}{dw}(w^H p)$=2p

$\frac{d}{dw}(w^H R w) = 2Rw$

Sonuçları elde edilir. Bu sayede ortalama karesel hata J(w)'nin w'e göre türevi olan gradient vektörü, ∇, aşağıdaki gibi tanımlanır.

$$\begin{aligned}\nabla &= \frac{dJ(w)}{dw} \\ &= -2p + 2Rw\end{aligned} \tag{4.34}$$

Hata yüzeyinde minimum nokta için gradient vektörünü sıfır alırsak

$$Rw_0 = p \tag{4.35}$$

Bulunur buradaki w_0 en uygun tap ağırlık vektörüdür. Denklem 4.35 normal denklemi olarak bilinir. Denklem 4.35'in her iki tarafını R^{-1} ile çarparsak

$$w_0 = R^{-1}p \tag{4.36}$$

Elde ederiz. Buradan anlaşıldığı gibi en uygun tap ağırlıklarının bulunabilmesi için iki kavramın bilinmesi gerekir. 1. u(n) tap giriş vektörünün korelasyon matrisi R ve 2. tap giriş vektörü u(n) ile istenilen cevap d(n) arasındaki çapraz korelasyon vektörü p.

4.6 Minimum Ortalama Karesel Hata

Denklem 4.21 de tanımlanan ortalama karesel hatanın minimum değerini denklem 4.36 da bulunan eşitlik yardımıyla tekrar yazarsak

$J_{\min} = \sigma_d^2 - {w_0}^H p - p^H w_0 + {w_0}^H R w_0$, ve $Rw_0 = p$ ise

$$\begin{aligned}J_{\min} &= \sigma_d^2 - p^H w_0 \\ &= \sigma_d^2 - p^H R^{-1} p\end{aligned} \tag{4.37}$$

bulunur.

Kendinden uyarlamalı bir süzgeç durağan bir ortamda çalışıyorsa hata yüzeyi sabit bir şekildir. Süzgecin görevi bu yüzeyin yakınlarında çalışmaktır. Eğer ortam durağan değilse, hem yüzeyin şekli hem de minimumu değişebilmektedir. Bu yüzden kendinden uyarlamalı süzgecin görevi, değişken ortamlarda sadece hata yüzeyinin minimumunu bulmak değil ayrıca sürekli takip etmektir.

4.7 En Dik Azalış Yöntemi

Herhangi bir enine süzgeçli kendinden uyarlamalı süzgecin ihtiyacı normal denklemiyle belirtilen eşitliği sağlayacak tap ağırlık vektörünün bulunmasıdır. Bunun bir yöntemi bazı analitik hesaplamaların kullanılmasıdır. Aslında bu yöntem en kısa çözüm olabilir fakat tap ağırlıklarının sayısı fazla olduğu ve gelen verinin oranının yüksek olduğu durumlarda algoritmanın karmaşıklığı artmakta ve bunu uygulamada sorunlarla karşılaşılmaktadır.

Buna alternatif bir çözüm Murray tarafından 1972'de bulunmuştur. En Dik Azalış(Steepest Descent) olarak bilinir. Ortalama karesel hata $J_{\min}$'i bulmak için izlenen yol aşağıdaki gibidir.

1- Başlangıç olarak tap ağırlık vektörü w(0)' a rasgele bir değer verilir, genel olarak ta 0' boş vektörü verilir. Bu sayede hata yüzeyindeki minimum noktanın yerine yönelik bir tahmin yapılmış olur.

2- Bu ilk tahmini kullanarak gradient vektörünü hesaplarız. Bu vektör ortalama karesel hata $J(n)$ 'nin gradient'idir, w'e göre n' zamanında hesaplanmıştır.

3- Tap ağırlıklarının yeni tahminini, bulunan gradient vektörünün ters yönünde önceki tap ağırlıklarını değiştirerek yaparız.

4- İkinci adıma dönerek işlemleri tekrarlarız.

Bu şekilde ağırlık vektörünün gradient vektörüne ters yönde ilerleyerek, hata başarım yüzeyinde minimuma doğru ilerleyerek en dik şekilde en uygun tap ağırlıklarına ulaşılmaya çalışılır.

$\nabla(n)$, n. zamanda gradient vektörünü göstersin. En dik azalış yöntemi ile n+1 anında ağırlık vektörü aşağıdaki gibi hesaplanır:

$$w(n+1) = w(n) + \frac{1}{2}\mu[-\nabla(n)] \tag{4.38}$$

μ pozitif reel sabittir. Daha önce denklem 4.34 ile ortalama karesel hatanın tap ağırlık vektörü w(n)'e göre türevini aşağıdaki gibi bulmuştuk.

$\frac{dJ(n)}{dw(n)} = -2p + 2Rw(n)$, en dik azalış yöntemi için korelasyon matrisi R' ve çapraz korelasyon vektörü p' önceden bilinmelidir. Eğer bunlar önceden bilinen değerler ise denklem

4.34'ü denklem 4.38'te yerine yazarak:

$$w(n+1) = w(n) + \mu\left[p - Rw(n)\right], \text{ n=0,1,2,...,} \tag{4.39}$$

bulunur. Buradaki μ' bir denemeden diğerine tap ağırlık vektöründeki değişimi kontrol eder. Bu yüzden μ' ye basamak boyutu(step size) yada ağırlık sabiti(weighting constant) denir. Denklem 4.39 en dik azalış algoritmasının matematiksel denklemidir.

En dik azalış algoritmasının çalışmasını özetlemek gerekirse:

1- Başlangıç olarak tap ağırlık vektörü w(0)' a rasgele bir değer verilir. Algoritma, hata yüzeyindeki minimum noktanın yerine yönelik bir tahmin yaparak başlatılmış olur.

2- Bu ilk tahmini kullanarak gradient vektörün $J(n)$ hesaplanır.

3- Tap ağırlıklarının yeni tahminini, bulunan gradient vektörünün ters yönünde önceki tap ağırlıklarını değiştirerek yaparız. Denklem 4.39.

4- İkinci adıma dönerek işlemleri tekrarlarız.

En dik azalış yönteminden sonra en uygun tap ağırlıklarını bulmanın başka bir yolu olan en küçük karesel ortalama (Least Mean Square, LMS) algoritmasına geçilebilir.

5. EN KÜÇÜK KARESEL HATA ALGORİTMASI

5.1 Genel

Eğer bir denemeden diğerine $\nabla J(n)$ gradient vektörüne ilişkin tam ölçümler yapılabilseydi, ve μ ağırlık sabiti uygun seçilebilseydi en dik azalış yöntemi en uygun Wiener çözümüne yaklaşabilirdi. Fakat gerçekte bunu yapmak mümkün değildir çünkü tap girişlerinin korelasyon matrisi R ve istenilen cevap ile tap girişlerinin çapraz korelasyon vektörü p önceden bilinmelidir. Sonuç olarak, gradient vektörüne ilişkin doğru tahminler yapmak mümkün olmaz bu yüzden gradient vektörü elimizde olan verilerden hesaplanarak elde edilmelidir. Başka bir deyişle, tap ağırlıkları gelen verilere göre kendiliğinden uyarlanmalıdır. Bu şekilde çalışan bir algoritma en küçük karesel ortalama LMS algoritmasıdır.

LMS algoritmasının en önemli özelliği basitliğidir. LMS, olması gereken korelasyon fonksiyonlarına yada matris çevirimlerine ihtiyaç duymaz.

5.2 LMS Algoritmasının Denklemi

$\nabla(n)$ gradient vektörüne ilişkin tahminler yapmada en çok tercih edilen yöntem daha önceki denklemlerde elde edildiği gibi korelasyon matrisi ve çapraz korelasyon vektörünü kullanmaktı.

$$\nabla(n) = -2p + 2Rw(n) \tag{5.1}$$

R ve p hakkında en basit tahminler yapabilmek için tap girişine ve istenilen cevaba bakılır. Bu sayede anlık tahminler(instantenous estimate) yapılabilir.

$$\hat{R}(n) = u(n)u^H(n) \tag{5.2}$$

$$\hat{p}(n) = u(n)d^*(n) \tag{5.3}$$

Bu durumda anlık gradient vektörünün kestirimi aşağıdaki gibi olur:

$$\hat{\nabla}(n) = -2u(n)d^*(n) + 2u(n)u^H(n)\hat{w}(n) \tag{5.4}$$

5.4 teki denklemi en dik azalan yöntem ile elde edilen 4.38 denklemine yazarsak:

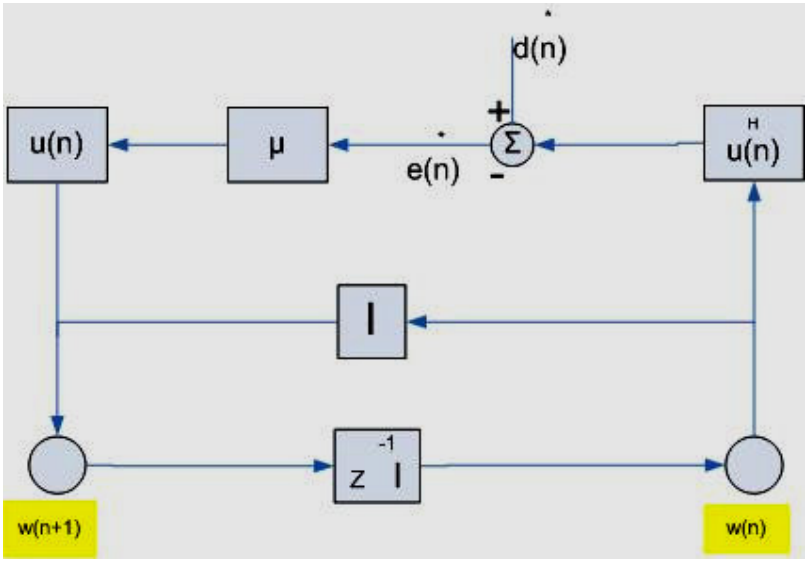
$$\hat{w}(n+1) = \hat{w}(n) + \mu u(n)\left[d^*(n) - u^H(n)\hat{w}(n)\right] \tag{5.5}$$

Hata e(n) istenilen işaret ile süzgeç çıkışındaki işaretin farkları olduğuna göre aşağıdaki denklemleri yazabiliriz.

$$e(n) = d(n) - \hat{w}^H(n)u(n) \tag{5.6}$$

$$\hat{w}(n+1) = \hat{w}(n) + \mu u(n)e^*(n) \tag{5.7}$$

Denklem 5.6 hatayı gösteren denklemdir, tap ağırlık vektörü $\hat{w}(n)$'nin anlık tahminine bağlıdır. Denklem 5.7'nin sağ tarafındaki $\mu u(n)e^*(n)$ terim ise, tap ağırlık vektörü $\hat{w}(n)$'nin anlık tahminine yapılan düzeltmedir. En dik azalış yönteminde olduğu gibi iterasyon işlemi tap ağırlık vektörüne rasgele bir başlangıç değeri verilmesiyle başlar, genel olarak bu değer boş sıfır vektörüdür. Aşağıdaki şekil 5.1 de LMS algoritmasının sinyal akış grafiği görülebilir.



Şekil 5.1 LMS algoritmasının işaret akışı

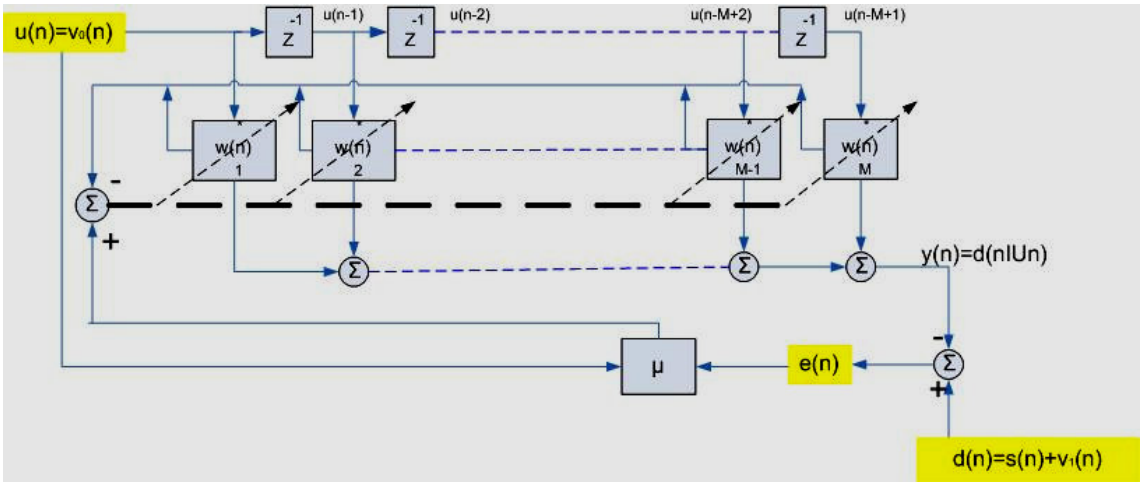
Denklem 5.2 ve 5.3 te verildiği gibi R ve p'nin değerleri anlık tahmini değerler olduğu için ilk bakışta algoritma iyi performans veremeyecekmiş gibi durur fakat algoritma doğasında kendinden uyarlamalı olduğu için algoritma, uyarlama esnasında bu tahminleri orta değerlere çeker bu da performansı iyileştirir.

Burada tanımlanan LMS'nin bulunması için tanımlanan işlemler aslında en dik azalış yöntemine farklı bir yaklaşım olarak görülebilir. En dik azalan yöntemden farklı olarak burada tap giriş vektörünün R korelasyon matrisi ve tap giriş vektörü ve istenilen cevap

arasındaki çapraz korelasyon vektörü p anlık tahminlerden hesaplanmıştır. Literatürde LMS'e yönelik farklı yaklaşımlar da vardır. Bunlara ileride ayrıca değinelecektir.

5.3 LMS Süzgecin Yapısı

LMS algoritması en dik azalış yönteminin bir uyarlaması olup tap giriş vektörünün R korelasyon matrisi ve tap giriş vektörü ve istenilen cevap arasındaki çapraz korelasyon vektörü p anlık tahminlerden hesaplanmıştır. Süzgecin iki girişi ve bir çıkışı vardır Şekil 5.2.



Şekil 5.2 Kendinden uyarlamalı LMS süzgeç

Şekil 5.2 de görüldüğü gibi u(n) enine süzgeç girişi, d(n) istenilen cevap girişi ve süzgeç çıkışı ile istenilen cevabın farkı da işlenmiş işarettir.

LMS ve türevleri için bazı kabullerin sağlanması gereklidir şekil 5.2 deki işaretler için

1- u(n) giriş vektörünün her biri önceki örneklerden istatiksel olarak bağımsızdır:

$$E\left[u(n)u^{H}(k)\right]=0\,,\ \text{k=1,2,3…,n-1} \tag{5.8}$$

2- u(n) giriş vektörünün her biri her istenilen cevap d(n) örneklerinden istatiksel olarak bağımsızdır:

$$E\left[u(n)d^{*}(k)\right]=0\,,\ \text{k=1,2,3…,n-1} \tag{5.9}$$

3- istenilen cevap d(n) örneği aynı zamandaki u(n) örneğine bağlıdır fakat d(n) örneklerinin her biri önceki örneklerden istatiksel olarak bağımsızdır.

Bu kabuller altında algoritmada kullanılan u(n) ortamdaki gürültü, d(n) ise gürültü ve bilgi işaretinin birleşimidir. e(n) ise gürültüden arınmış bilgi işaretidir. İstenilen cevap d(n) aşağıdaki gibi tekrar yazılabilir.

$$d(n) = s(n) + v_1(n) \tag{5.10}$$

s(n) bilgi işareti, $v_1(n)$ gürültü işareti,

$$u(n) = v_0(n) \tag{5.11}$$

ortamdaki gürültü,

$$y(n) = \hat{w}^H u(n) \tag{5.12}$$

enine süzgeç çıkışı,

Yukarıdaki kabullerde yapıldığı gibi d(n)'in u(n)'e bağımlılığı gürültü işaretleri $v_0(n), v_1(n)$'in birbirleri ile bağımlı olmasından kaynaklanır. $v_0(n), v_1(n)$ birbirlerine bağımlı olabilir ancak aynı olmayabilir. Bu durumda enine süzgeç y(n) işaretini $v_1(n)$'e benzetmeye çalışacaktır. Çünkü $y(n), v_1(n)$ birbirine ne kadar yakın olursa aşağıdaki eşitlik sağlanmış olur. Burada y(n) süzgeç çıkışı olduğu için enine süzgecin çalıştığını ve aşağıdaki eşitliklerin sağlandığını düşünelim.

$$\begin{aligned} &y(n) = v_1(n) \\ &e(n) = d(n) - y(n) \\ &e(n) = s(n) + v_1(n) - v_1(n) \\ &e(n) = s(n) \end{aligned} \tag{5.13}$$

Sağlanmış olur. Enine süzgecin görevi e(n) hatasını en küçük değerde tutmak olduğu için en uygun çözüm sağlandığında e(n) hata işareti bilgi işareti s(n) üretmiş olur. LMS'nin amacı J(n)' i en küçük değerde tutmaktır, bu sağlandığı zaman karesel hata da en aza indirilmiş olur. Bunun için ayrıca:

$$\begin{aligned} &E\left[e^2(n)\right] = E\left[(d(n) - y(n))^2\right] \\ &= E\left[d^2(n) + y^2(n) - 2d(n)y(n)\right] \\ &= E\left[(s(n) + v_1(n))^2 + y^2(n) - 2y(n)(s(n) + v_1(n))\right] \\ &= E\left[s(n)^2 + v_1(n)^2 + 2s(n)v_1(n) + y^2(n) - 2y(n)s(n) - 2y(n)v_1(n))\right] \end{aligned} \tag{5.14}$$

Yukarıdaki kabullerde d(n)'in u(n)' e bağımlı olduğu ve u(n)'in d(n)'den bağımsız olduğu belirtilmişti. Zaten gerçekte de s(n)'in y(n) ve $v_1(n)$'den bağımsız olması beklenir. Çünkü bilgi işareti gürültülerden bağımsızdır. Bu yüzden denklem 5.14 te s(n) olan yerler sıfırdır.

$$E\left[e^2(n)\right] = E\left[s(n)^2 + v_1(n)^2 + y^2(n) - 2y(n)v_1(n))\right] \\ = E\left[s(n)^2 + (v_1(n) - y(n))^2\right] \tag{5.15}$$

bulunur. Hatanın minimum olması gerektiğine göre ve bilgi işaretini silemeyeceğimiz için, çünkü süzgecin yapısı buna uygun değil,

$$v_1(n) - y(n) = 0 \tag{5.16}$$

Olmasını bekleriz, bu da

$$v_1(n) = y(n) \\ y(n) = \hat{w}^H u(n) \\ y(n) = \hat{w}^H v_0(n) \\ v_1(n) = \hat{w}^H v_0(n) \tag{5.17}$$

Bulunur bu eşitliğin sağlanması için $v_0(n)$ ile $v_1(n)$ 'in birbiri ile bağıntılı olması gerekir. Böylece Enine süzgeç performansına bağlı olarak 5.17 denklemini sağlamaya çalışır. Sonuç olarak hata en aza indirilmiş olur ve e(n) hata işareti bize gürültüden arındırılmış bilgi işaretini verir.

5.4 LMS Matlab Uygulaması

Denklem 5.6 ve 5.7 de bulunan eşitlikler Matlab'te uygulanırsa:

İki sinüzoidal işaretin toplamı şeklinde bir işaret ve eşit dağılımlı beyaz gürültü şeklinde bir gürültü için, performans kriteri olarak gürültüsüz işaret ile süzülmüş işaretin farkının karesel ortalaması, OKH seçildiğinde(Mean Square Error, MSE) sonuçlar aşağıdaki şekil 5.3 te görüldüğü gibidir.

Şekil 5.3'e bakıldığında:

a) İki sinüzoidal işaretin toplamından oluşan ve genliği (-1,1) aralığında değişen bir işaret, bu işarete tüm bant boyunca eşit dağılım gösteren ve genliği (-0.5,0.5) aralığında değişen beyaz gürültü(u(n) işareti) ve bu orijinal sinyal ile gürültünün

toplamından oluşan istenilen cevabı oluşturan d(n) sinyali,

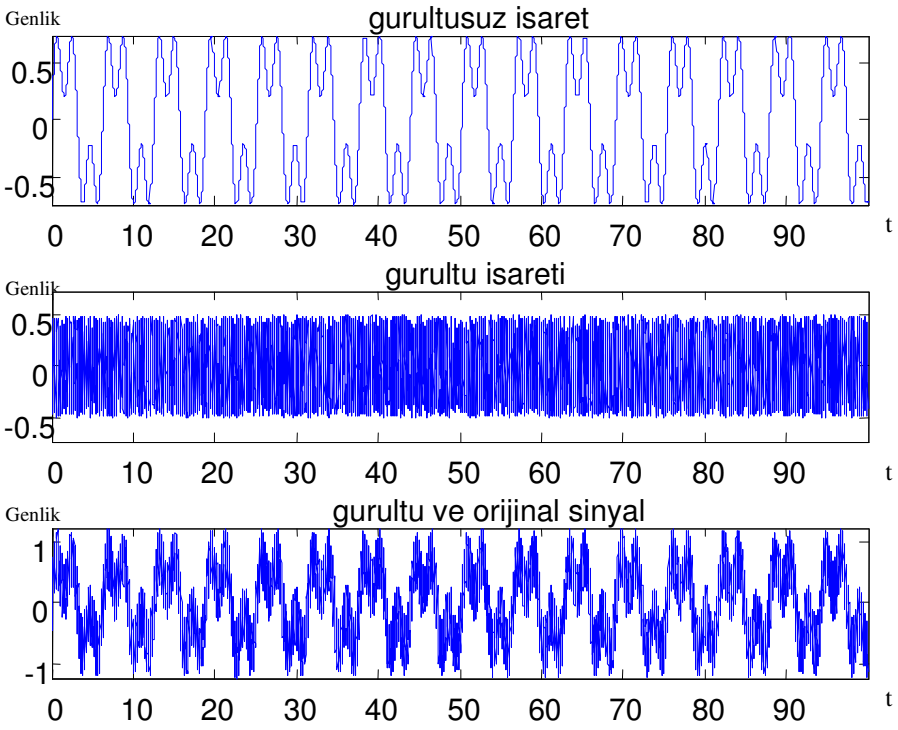
b) İki sinüzoidal işaretin toplamından oluşan işaret ve LMS süzgecin çıkışı olan gürültüden arındırılmış e(n) işareti,

c) LMS süzgecin çıkışı olan gürültüden arındırılmış e(n) işareti ve orijinal sinyal ile gürültünün toplamından oluşan istenilen cevabı oluşturan d(n) sinyali,

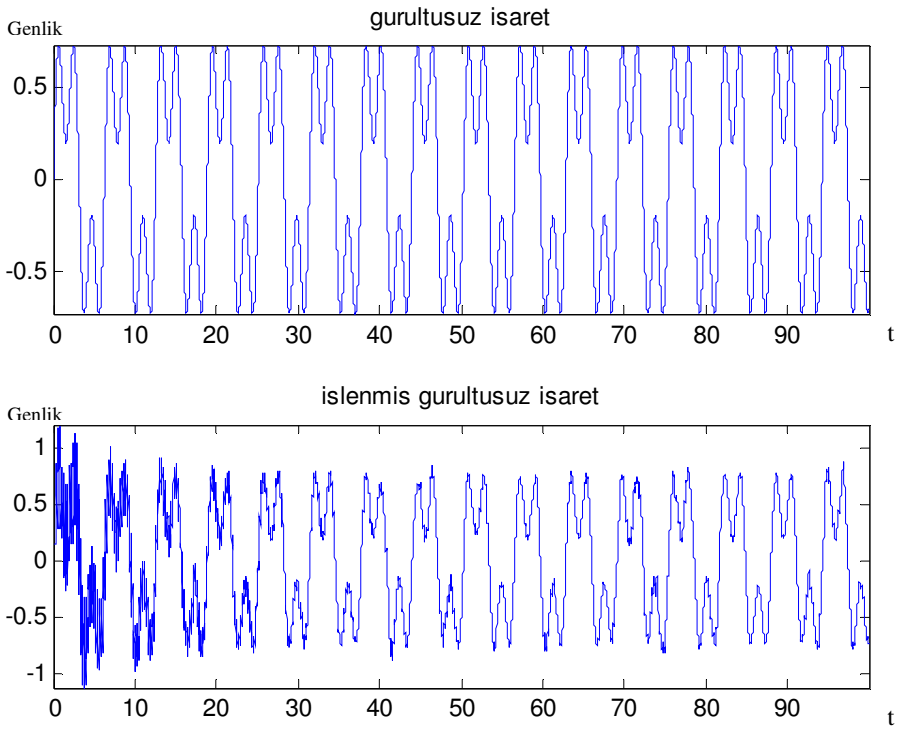
d) Gürültüsüz orijinal işaretin karesi ile LMS çıkışı işaretin karesinin farkının büyüklüğünün ortalama değeri yani ortama karesel hata,

e) Gürültüsüz orijinal işaret ile gürültünün frekans bileşenleri, görülmektedir.

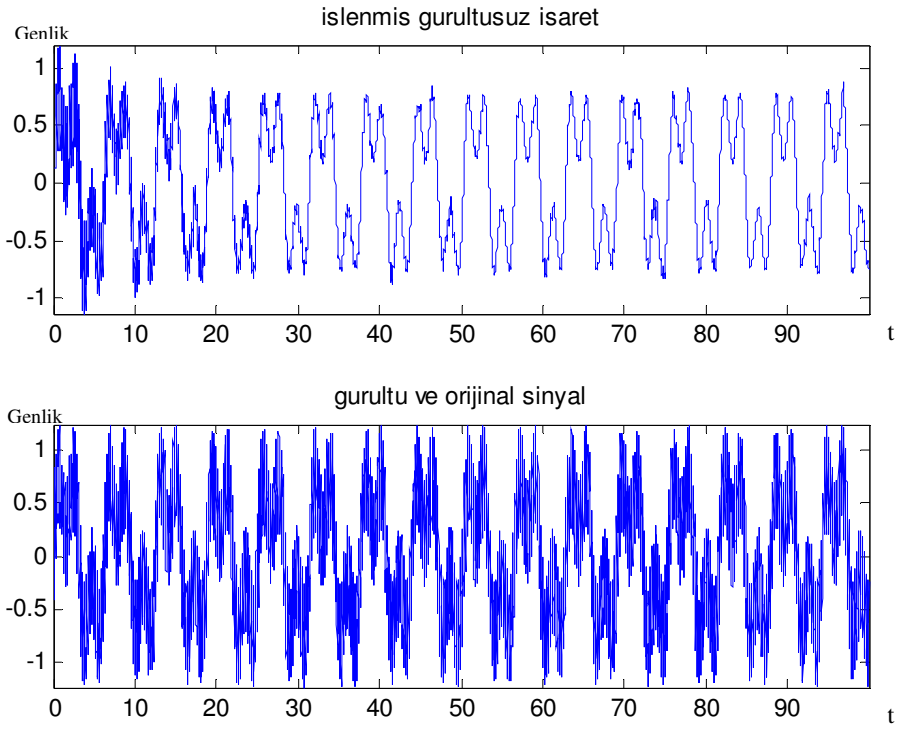
a)



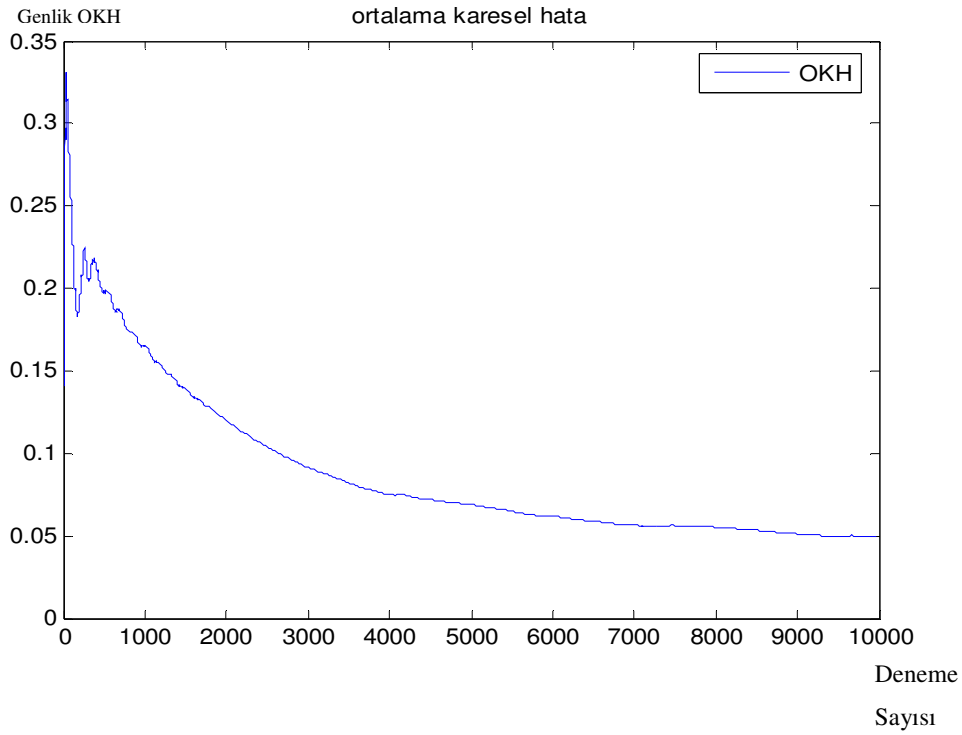
b)


gurultusuz isaret
Genlik
0.5
0
-0.5
0
10
20
30
40
50
60
70
80
90
t
islenmis gurultusuz isaret
Genlik
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
t

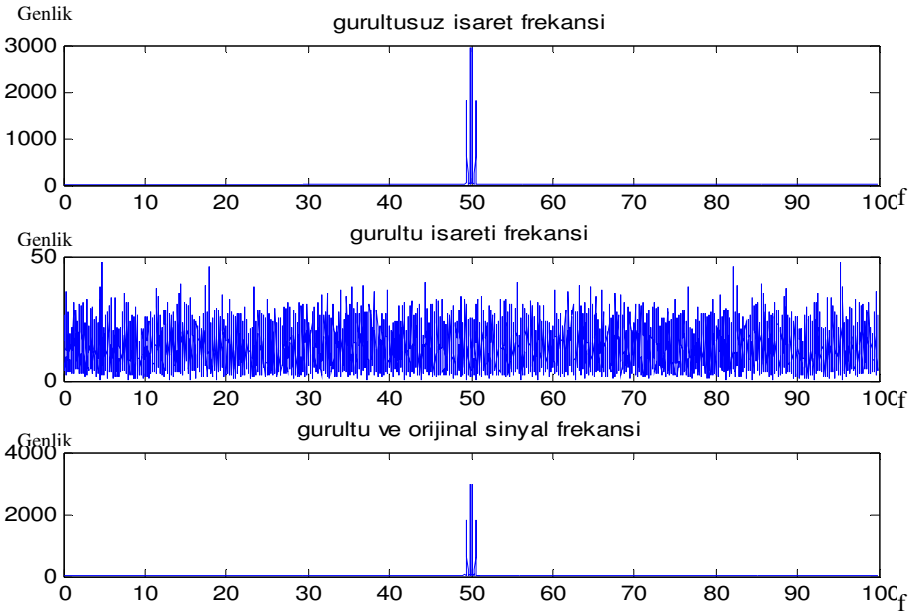
c)


islenmis gurultusuz isaret
Genlik
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
t
gurultu ve orijinal sinyal
Genlik
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
t

d)



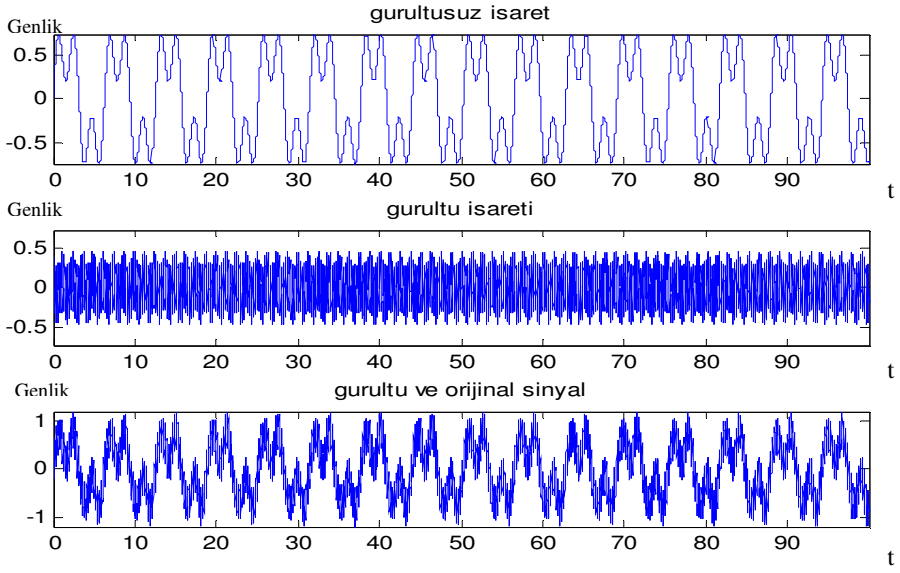
e)



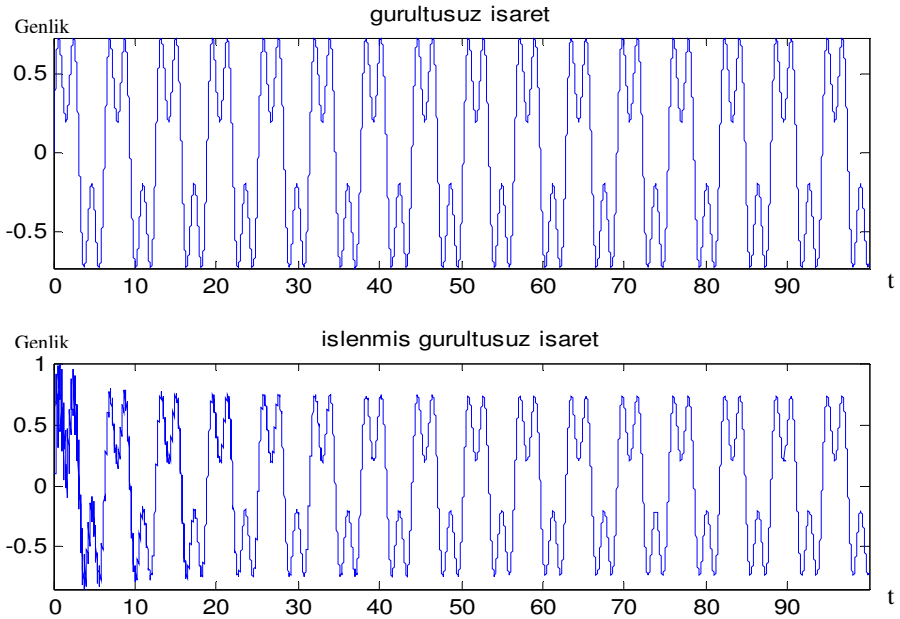
Şekil 5.3 a-Gürültüsüz sinüzoidal giriş, beyaz gürültü u(n) ve istenilen cevap d(n) işaretleri, b-Gürültüsüz sinüzoidal işaret ve işlenmiş LMS çıkışı e(n), c- İşlenmiş LMS çıkışı e(n) ve istenilen cevap d(n), d- Gürültüsüz orijinal işaret ile LMS çıkışı e(n)'in ortalama karesel farkları, , e- Gürültüsüz orijinal işaret ile gürültünün frekans bileşenleri

Şekil 5.3'te görüldüğü gibi OKH, 4000.-5000 denmeler arasında minimum değere ulaşmış kabul edilebilir. Önceki örneğin aynısı bir işaret ve giriş işaretine benzer fakat farklı frekanslara sahip iki sinüzoidal işaretin toplamı şeklinde bir gürültü için, performans kriteri olarak gürültüsüz işaret ile süzülmüş işaretin farkının karesel ortalaması, OKH seçildiğinde(Mean Square Error, MSE) sonuçlar aşağıdaki şekil 5.4 te görüldüğü gibidir.

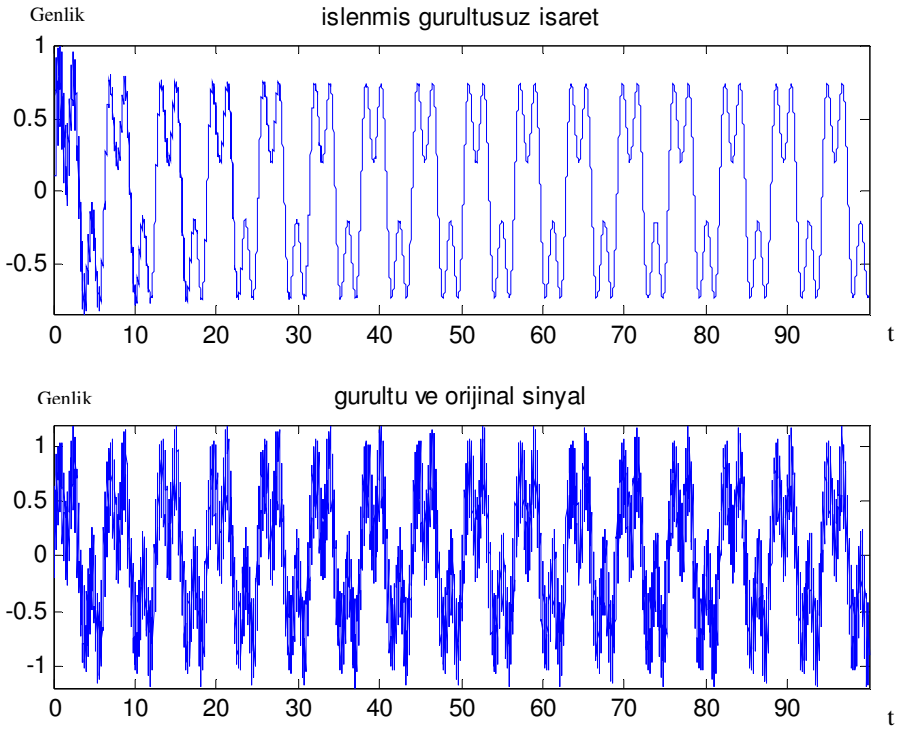
a)



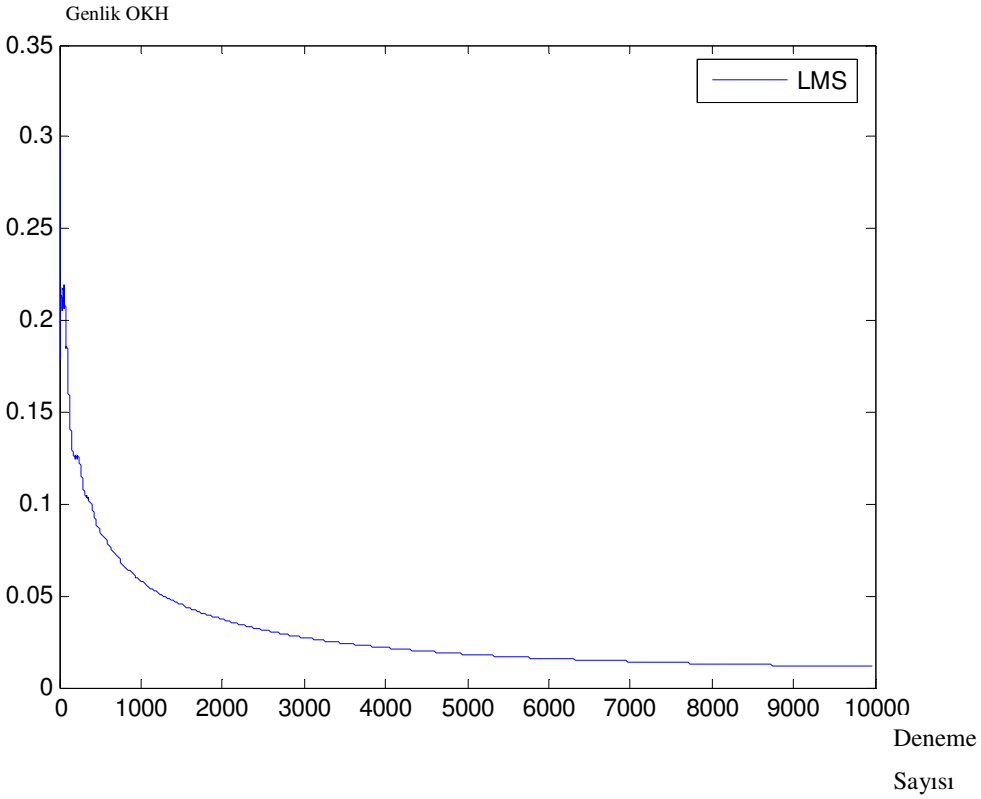
b)



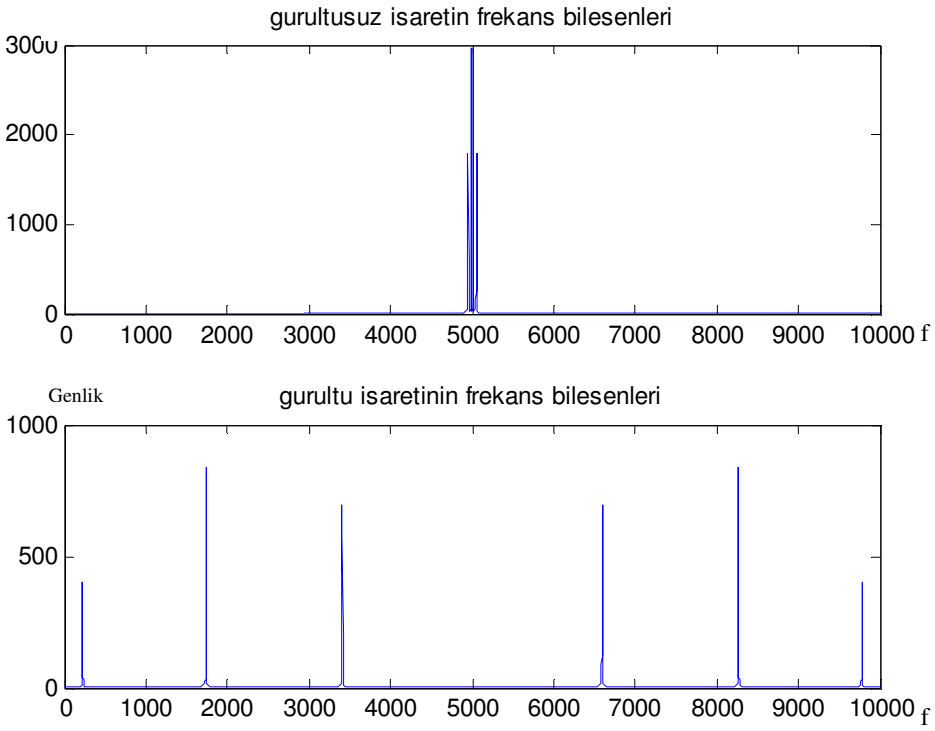
c)


Genlik
islenmis gurultusuz isaret
1
0.5
0
-0.5
0
10
20
30
40
50
60
70
80
90
t
Genlik
gurultu ve orijinal sinyal
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
t

d)


Genlik OKH
LMS
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
Deneme
Sayısı

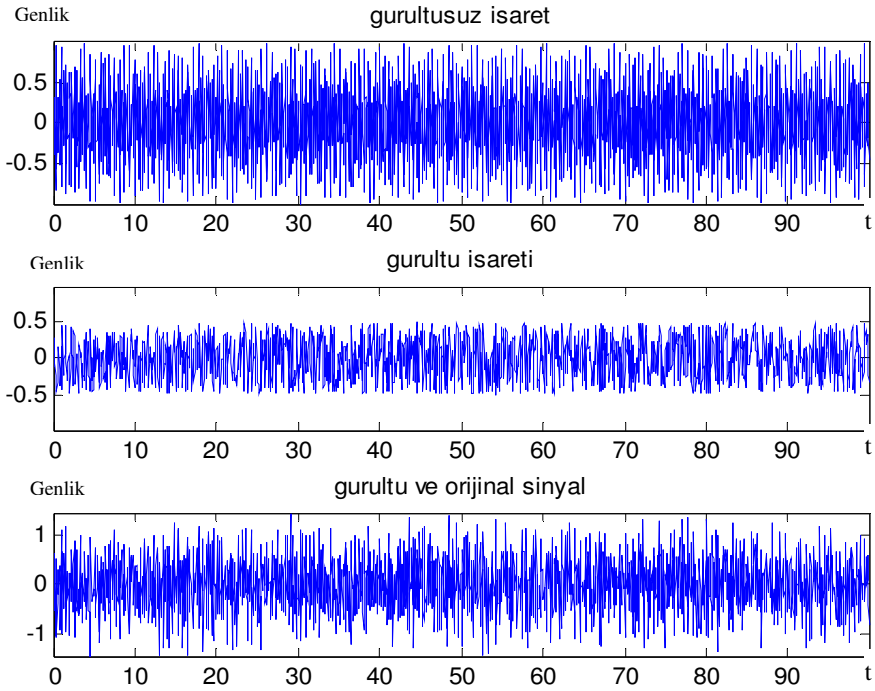
e)



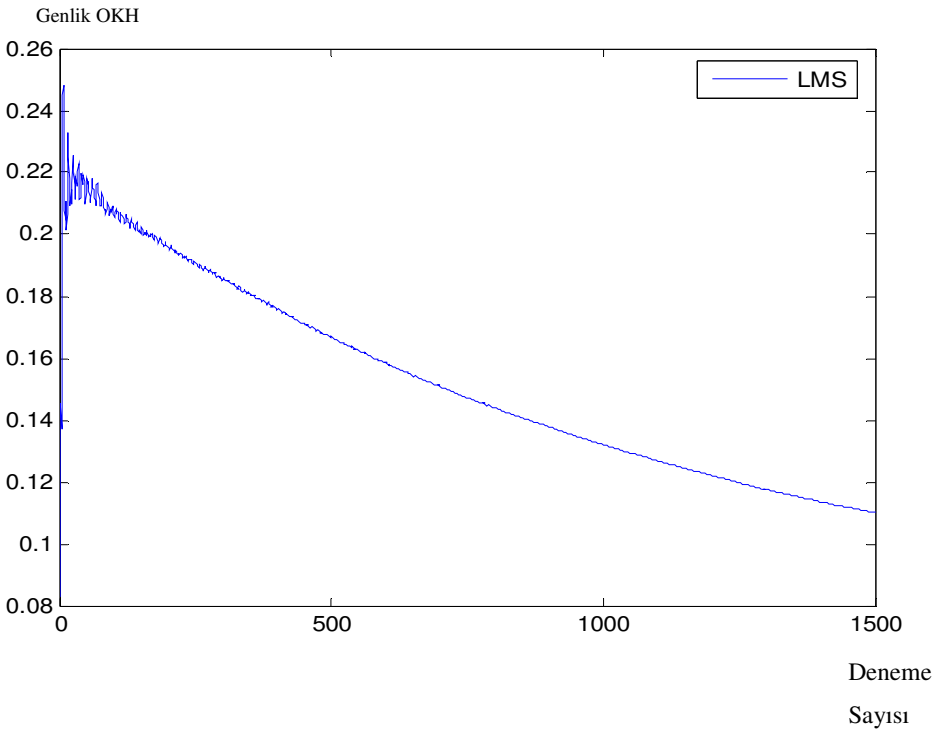
Şekil 5.4 a-Gürültüsüz sinüzoidal giriş, sinüzoidal gürültü u(n) ve istenilen cevap d(n) işaretleri, b- Gürültüsüz sinüzoidal işaret ve işlenmiş LMS çıkışı e(n), c- İşlenmiş LMS çıkışı e(n) ve istenilen cevap d(n), d- Gürültüsüz orijinal işaret ile LMS çıkışı e(n)'in ortalama karesel farkları, e- Gürültüsüz orijinal işaret ile gürültünün frekans bileşenleri

Yukarıda yapılan denemelerde görülebilirlik açısından kolaylık sağlaması için düşük frekanslı işaretler kullanılmıştır. Burada asıl karşılaştırma kriteri, gürültüsüz orijinal işaret ile LMS çıkışı e(n)'in ortalama karesel farkları olacağından; 800 Hz ve 4 KHz den oluşan bilgi işaretine normal dağılımlı beyaz gürültüyü eklediğimizde 200 deneme sonucunda oluşan ortalama OKH (Mean Square Error, MSE) sonuçları aşağıdaki şekil 5.5 te görüldüğü gibidir.

a)



b)



Şekil 5.5 a) 800 Hz ve 4 KHz den oluşan bilgi işareti, normal dağılımlı beyaz gürültü ve toplamı b) 200 deneme sonucunda ortalama OKH

5.5 LMS Türevleri

LMS algoritması diğer algoritmalara göre biraz basit kalmakla birlikte sonuçları kabul edilebilir değerlerdedir. Eğer LMS'nin performansını artırmak istersek algoritmaya eklemeler yapabiliriz. Bu durumda aklımızdan çıkarmamamız gereken algoritma ne kadar zorlaşırsa hesapsal maliyet ve uygulama zorluğu o kadar artar. Ancak bu ilaveler artış, performansta da görüldüğü sürece yapılmalıdır. Donanım konusunu düşünüldüğünde varolan imkanlar dahilinde bu çalışmada sadece LMS uygulayabilmektedir, bu konuya ileride ayrıca yer verilecektir.

5.6 Normalize LMS(NLMS)

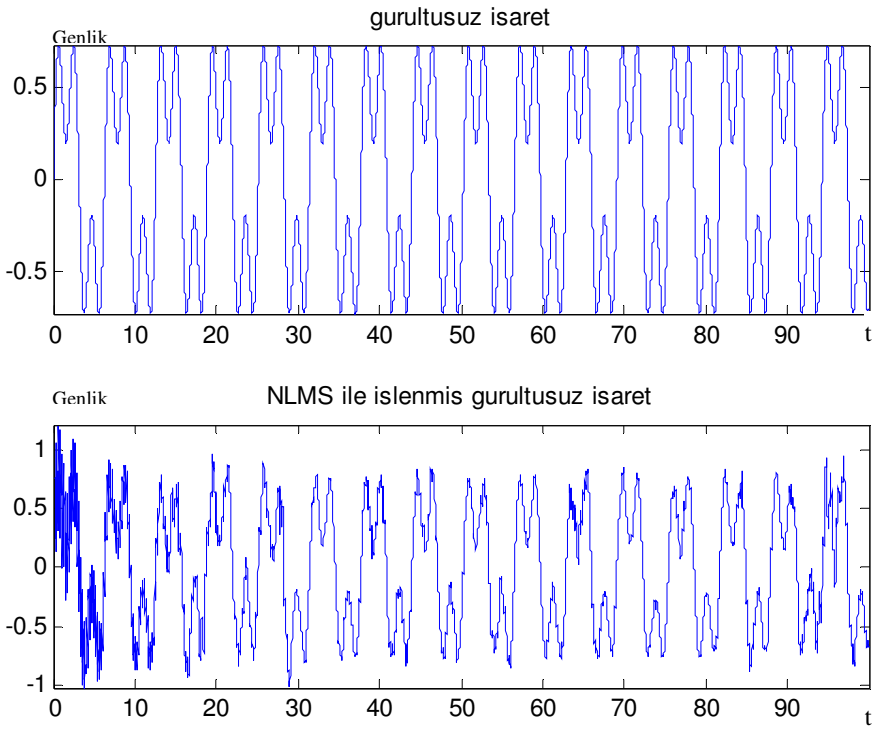
LMS algoritmasında düzeltme faktörü $\mu u(n)e^{*}(n)$ doğrudan tap giriş değerlerine bağlıdır. Bu durumda u(n) büyük değerler aldığında gradient gürültüsü kuvvetlendiricisi(gradient noise amplication) olarak bilinen durumla karşılaşılır. Bu durumdan kurtulmak için NLMS yöntemine başvurulur. Bu yöntem ile tap girişleri, tap girişlerinin toplamının karesi ile normalize edilir(squared Euclidean norm). Böylece tap ağırlık vektörü $\hat{w}(n+1)$'in n+1 zamanında güncellenmesi normalize edilmiş olup daha kontrollü bir güncelleme yapılmış olur. NLMS nin LMS den farkı en uygun çözüme daha hızlı yakınsamasıdır. Aşağıdaki şekil 5.6 te aynı girişler ve parametreler için LMS ve NLMS performans grafikleri gösterilmiştir.

$$\hat{w}(n+1) = \hat{w}(n) + \frac{\mu}{a + u(n) * u^{T}(n)} u(n)e^{*}(n) \tag{5.18}$$

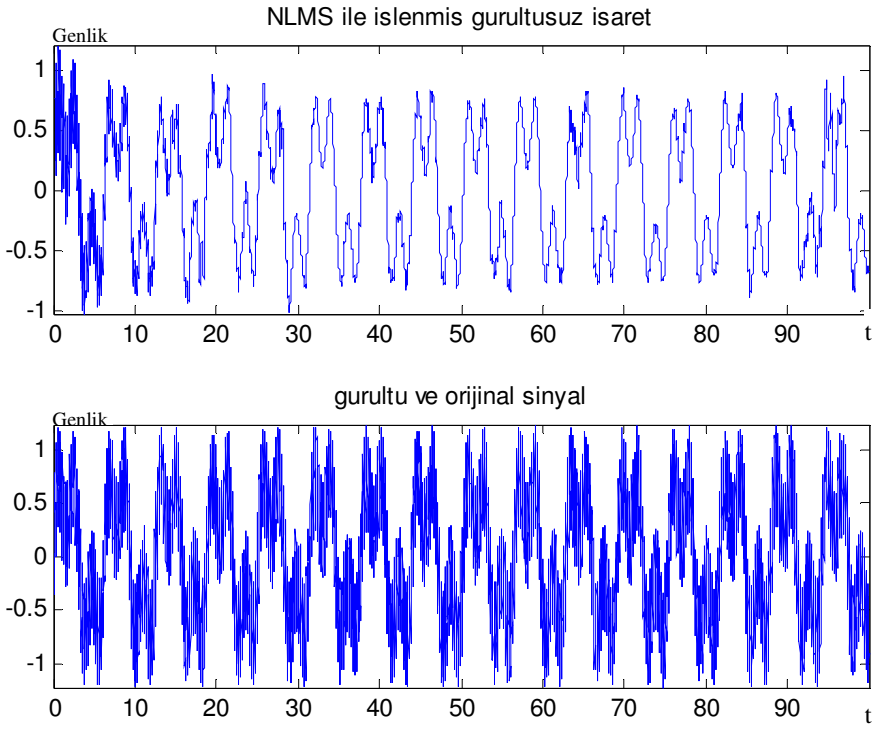
Denklem 5.18 NLMS algoritması için tap ağırlıklarının nasıl güncelleneceğini göstermektedir. a>0 büyük bir değer olup sıfıra bölmeye karşı bir önlemdir. Bu algoritmada

$0<\mu<2$ olması gerekmektedir.

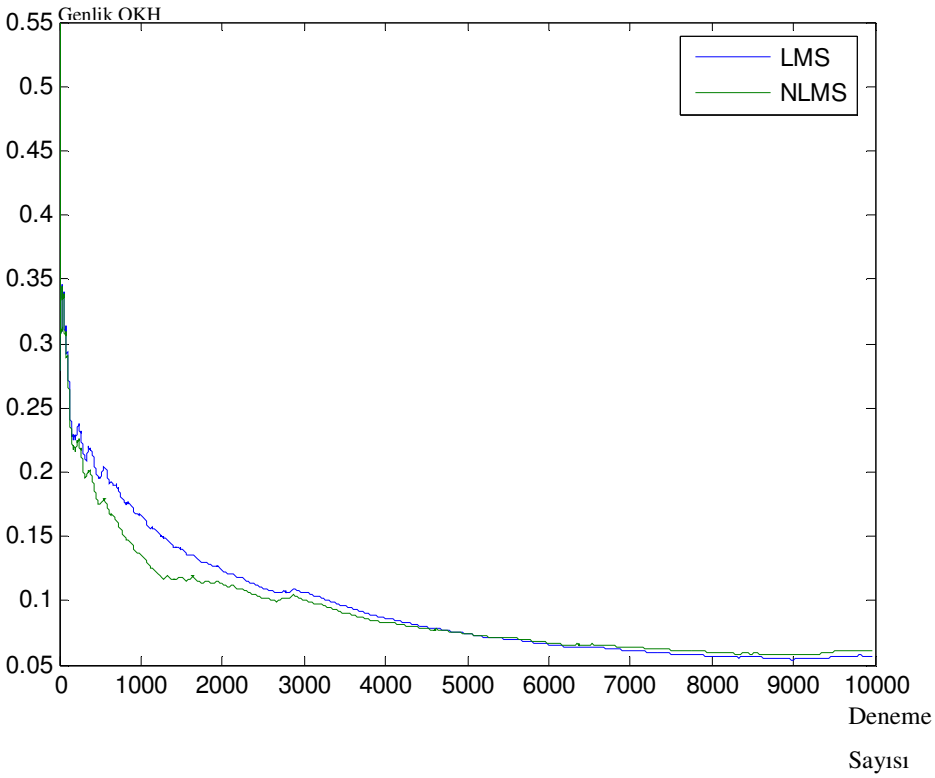
a)


gurultusuz isaret
Genlik
0.5
0
-0.5
0
10
20
30
40
50
60
70
80
90
t
Genlik
NLMS ile islenmis gurultusuz isaret
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
t

b)


NLMS ile islenmis gurultusuz isaret
Genlik
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
t
gurultu ve orijinal sinyal
Genlik
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
t

c)



Şekil 5.6 a-Gürültüsüz sinüzoidal giriş ve işlenmiş NLMS çıkışı e(n), b- İşlenmiş NLMS çıkışı e(n) ve istenilen cevap d(n), c- Gürültüsüz orijinal işaret ile NLMS çıkışı e(n) ve LMS çıkışı e(n)'in ortalama karesel farkları

Şekil 5.6 c) de görülen performans eğrileri gözükmektedir. Görüldüğü gibi NLMS, LMS'e göre daha hızlı en uygun çözüme yaklaşmaktadır. deneme sayısı n sonsuza gittikçe LMS daha başarılı gibi gözükmektedir. Aslında NLMS daha başarılıdır fakat adım ağırlığı, μ parametresinin en uygun seçilmemesi böyle bir sonuç doğurabilir. Buradan anlaşılan süzgecin fiziksel sınırları içerisinde en uygun çözümü yakalamak adım ağırlığı parametresi ile yakından alakalıdır.

5.7 Değişken Adım Ağırlıklı LMS(N^2LMS)

NLMS algoritmasında tap ağırlık vektörü $\hat{w}(n+1)$'in n+1 zamanında güncellenmesi normalize edilmiş olup daha kontrollü bir güncelleme yapılmıştı. Bu normalizasyonu u(n) girişlerinin büyük değerler alabilme olasılığına ilişkin kullanmıştık, burada dikkat edilmesi gereken ikinci bir nokta adım ağırlığı μ 'nün seçimidir. Bu seçim İdeal olarak aşağıda

belirtilen aralıkta olmalıdır(Haykin, 1986).

$0<\mu<\frac{2}{TGG}$, TGG= toplam giriş gücünün toplamı (5.19)

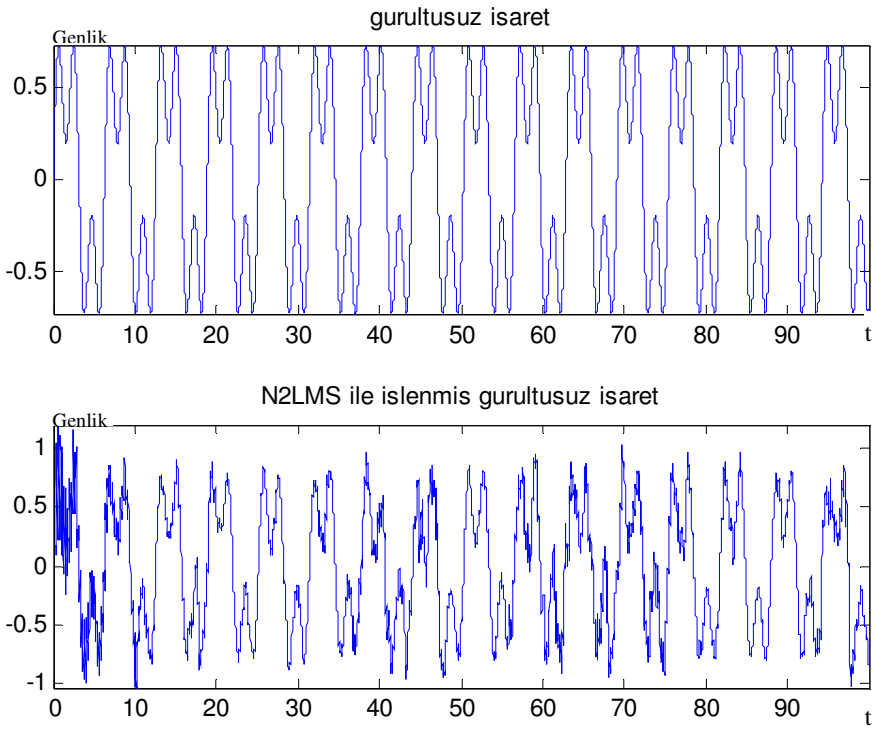
Bu durumda NLMS de olduğu gibi hem u(n)'in normalizasyonunun yapılmış olmasıyla hem de μ'nün u(n) girişlerine ve dolayısıyla n zamanına göre değişken olması performansı değiştirir.

$$\hat{w}(n+1)=\hat{w}(n)+\frac{2S}{a+(u(n)*u^{T}(n))^{2}}u(n)e^{*}(n) \quad (5.20)$$

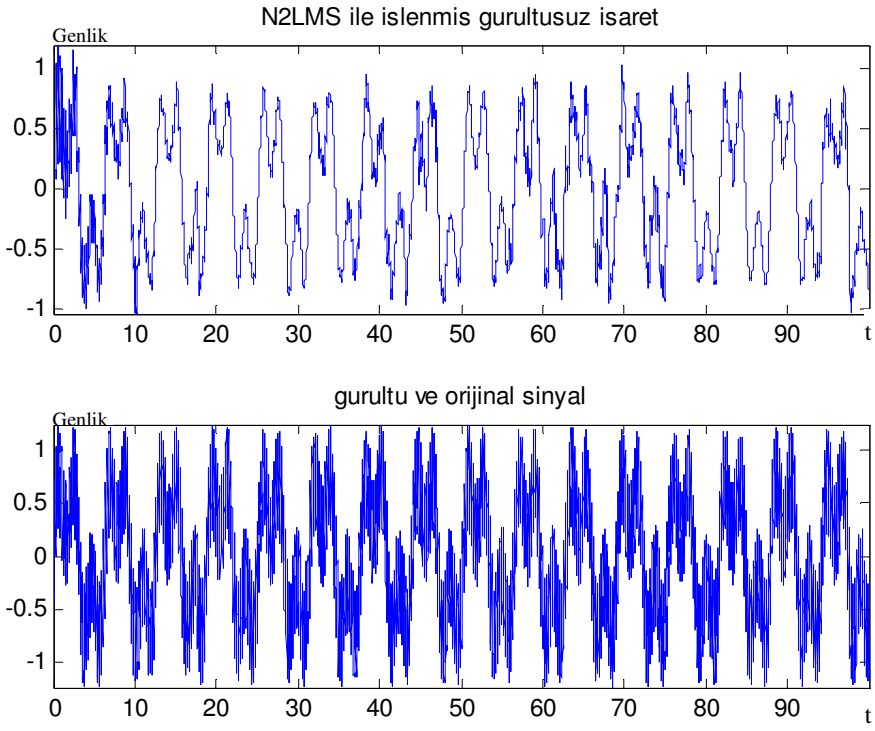
Denklem 5.20 N^2LMS algoritması için tap ağırlıklarının nasıl güncelleneceğini göstermektedir. a>0 büyük bir değer olup sıfıra bölmeye karşı bir önlemdir. S ise 0<2S<1 arasında bir değer olup başka bir ölçeklendirmedir. Maalesef μ' tamamen bağımsız bir hale dönüştürülememiştir. Ancak yine de en uygun çözüme daha hızlı yakınsadığını görebilmekteyiz fakat kendinden uyarlamalı algoritmaların doğasında olan geri besleme sebebiyle sistem ne kadar çok karmaşık olursa n sonsuza giderken süzgecin başarısı da aynı orantıda değişebilmektedir. Aşağıdaki Şekil 5.7 de aynı girişler için LMS, NLMS ve N^2LMS performans grafikleri gösterilmiştir.

Şekil 5.7 de Üç süzgeç performansları incelendiğinde hatayı en hızlı azaltan N^2LMS süzgeç olmuştur ancak n sonsuza giderken NLMS ve LMS süzgeçleri hatayı en aza indirmede daha başarılıdır. Bunun farklı sebepleri olabilir fakat unutulmamalıdır ki burada uğraşılan bütün bantta eşit dağılıma sahip beyaz gürültüdür. Bu tamamen rasgele bir süreç olup süzgeçlerin sınırlarını zorlamaktadır. Giriş olarak alınan şekiller de incelendiğinde SNR oranları 0 dB' e kadar gidebilmektedir. Buradaki amaç en kötü durumlarda süzgeçleri test etmektir. Ayrıca daha öncede belirtildiği gibi kendinden uyarlamalı süzgeçler geri beslemeye sahip olduğundan n sonsuza giderken önceden öngörülemeyen hatalar oluşabilmektedir. Bu yüzden ileride de bahsedileceği gibi performans olarak kabul edilebilir değerler içinde kaldığı için ve uygulamada diğerlerine göre kolaylıklara sahip olduğu için donanım uygulamasında LMS süzgeci FPGA üzerinde koşturulacaktır.

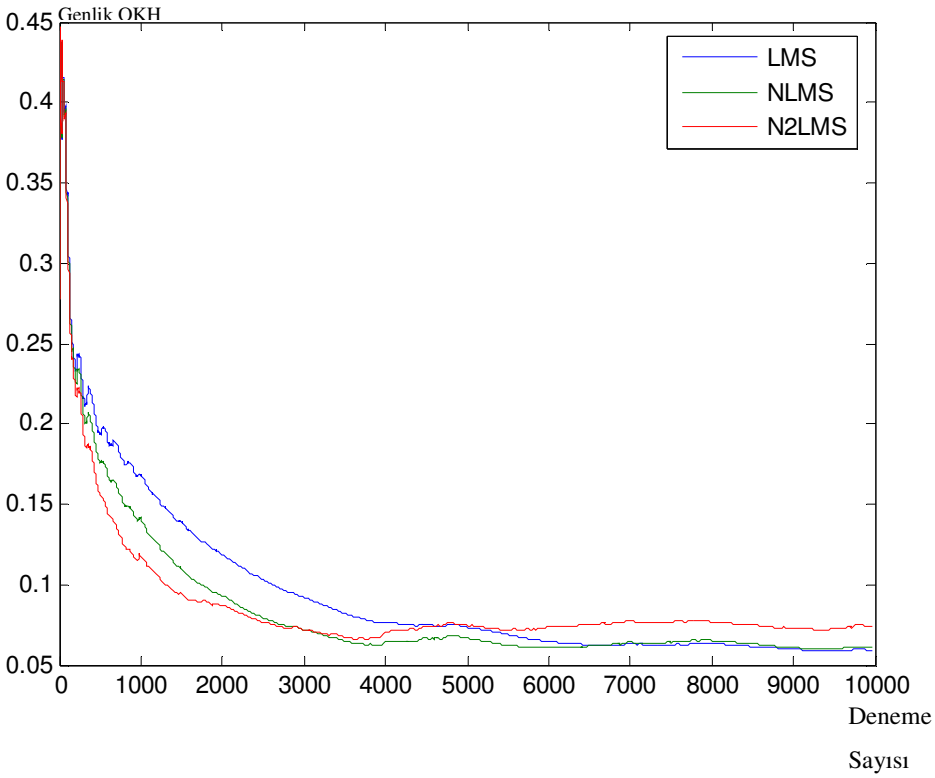
a)


gurultusuz isaret
Genlik
0.5
0
-0.5
0
10
20
30
40
50
60
70
80
90
t
N2LMS ile islenmis gurultusuz isaret
Genlik
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
t

b)


N2LMS ile islenmis gurultusuz isaret
Genlik
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
t
gurultu ve orijinal sinyal
Genlik
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90
t

c)



Şekil 5.7 a-Gürültüsüz sinüzoidal giriş ve işlenmiş N^2 LMS çıkışı e(n), b- İşlenmiş N^2 LMS çıkışı e(n) ve istenilen cevap d(n), c- Gürültüsüz orijinal işaret ile N^2 LMS çıkışı e(n), NLMS çıkışı e(n) ve LMS çıkışı e(n)'in ortalama karesel farkları

5.8 LMS Algoritma ve Performans Özeti

LMS algoritmasında sonucu belirleyen $\hat{w}(n+1)$'in tahmini değeri

1- Giriş işaretinin önceki değerleri, u(n),u(n-1),….,u(0)

2- İstenilen cevabın önceki değerleri, d(n),d(n-1),….,d(0)

3- tap ağırlık vektörünün başlangıç değeri $\hat{w}(0)$.

değerlerine bağlıdır. LMS algoritmasının performansı ise $\hat{w}(n+1)$'in en uygun Weiner çözümüne yaklaşması ile alakalıdır. Algoritmanın sınırları olduğu aşikârdır ancak bu performansı düşürmemek için de dikkat edilmesi gereken parametreler vardır.

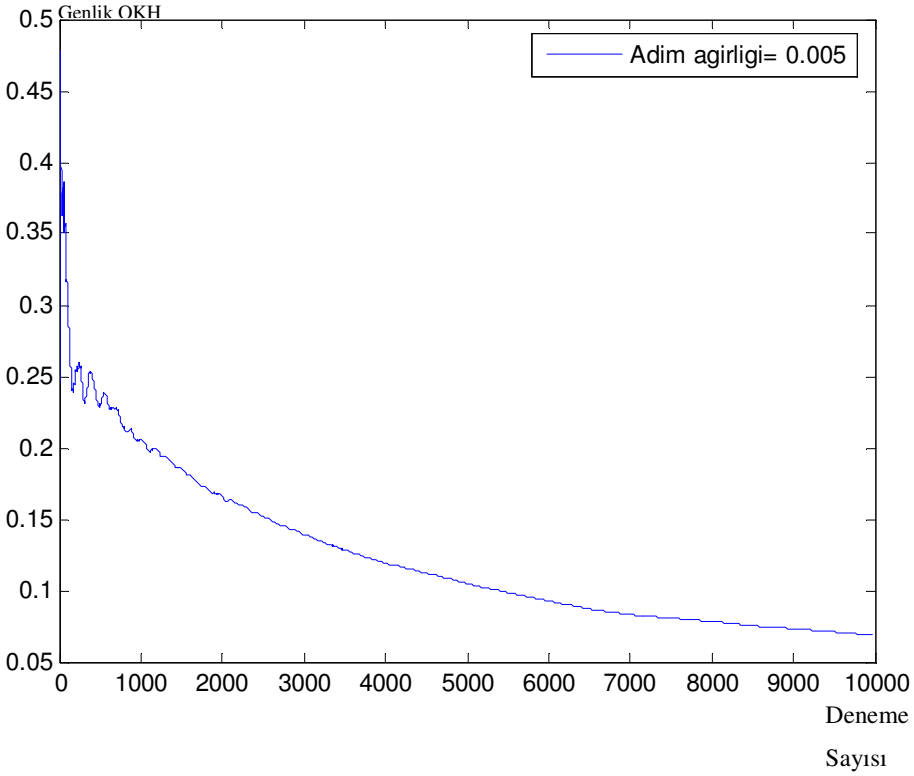
μ' adım ağırlığının seçimi çok önemlidir. İdeal olarak adım ağırlığı aşağıda belirtilen aralıkta olmalıdır(Haykin, 1986).

$0<\mu<\frac{2}{TGG}$, TGG= toplam giriş gücünün toplamı (5.21)

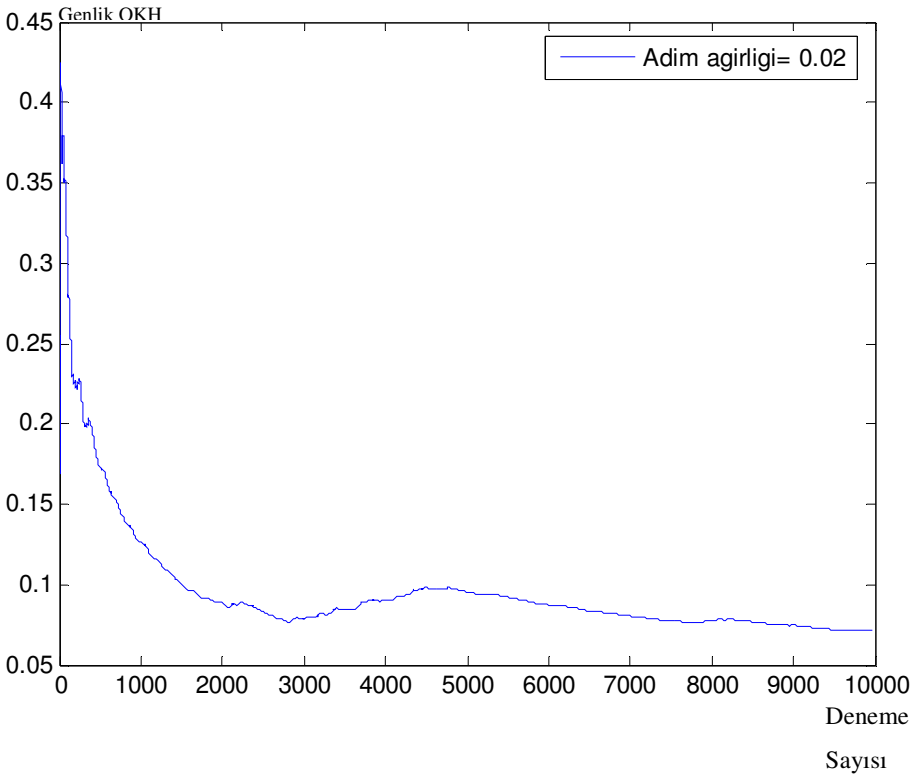
TGG pratikte M taplı süzgecin giriş değerlerinin ortalama karesel değerlerinin toplamıdır. Şekil 5.8 de μ=0.02, 0.1, 1 için sonuçlar karşılaştırılmıştır.

Şekil 5.8 dikkatle incelenirse süzgecin performansı doğrudan μ'e bağlıdır. Ancak yapılan çalışmalar sonucunda bilinmektedir ki uygun aralıkta küçük μ değerleri için süzgeç yavaş yakınsar, uygun aralıkta büyük μ değerleri için süzgeç daha hızlı yakınsar. Uygun değerler dışında örneğin μ'nün büyük değerleri için süzgeç sabit bir değere yakınsayamaz minimum hata yüzeyinde minimum noktaya yaklaşamaz.

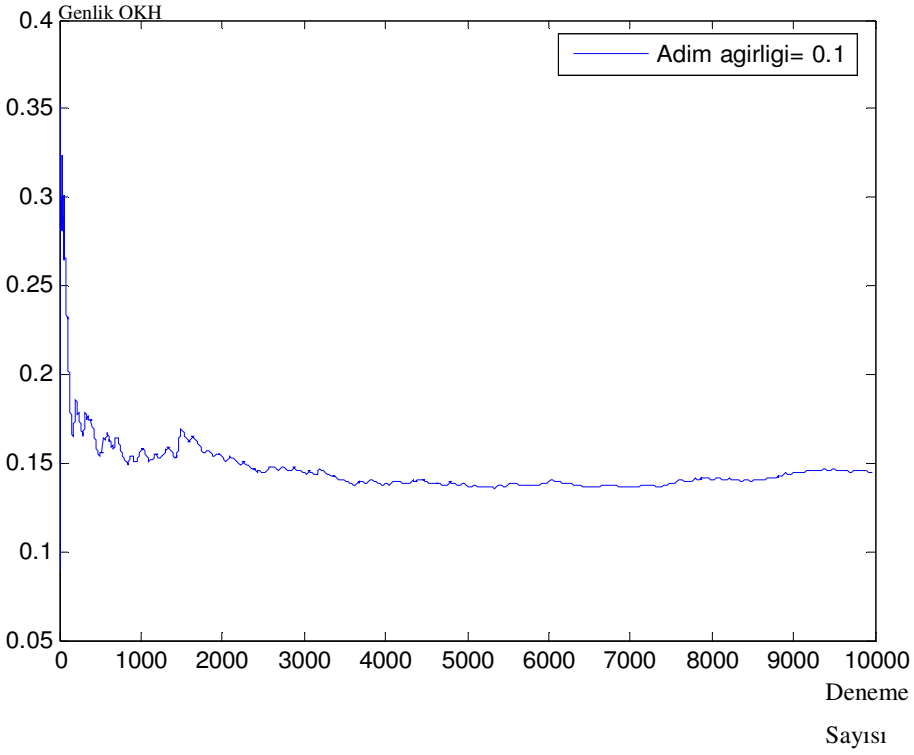
a)



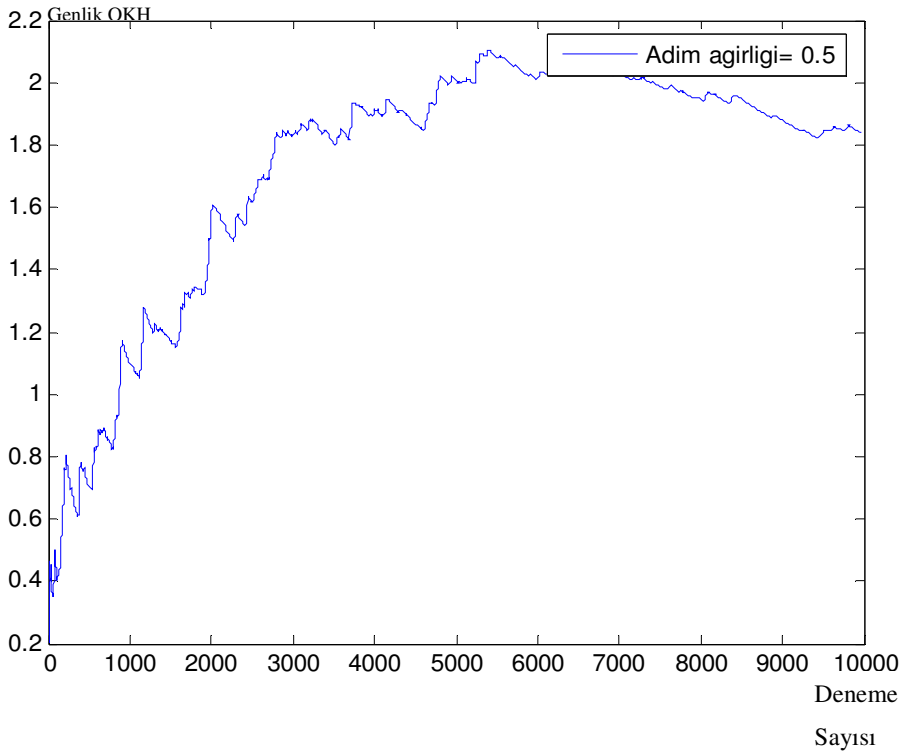
b)


Genlik OKH
Adim agirligi= 0.02
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
Deneme
Sayısı

c)


Genlik OKH
Adim agirligi= 0.1
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
Deneme
Sayısı

d)



Şekil 5.8 a- μ =0.005, b- μ =0.02, c- μ =0.1, d- μ =0.5

Şimdiye kadar anlatılanların özeti olarak: bir LMS kendinden uyarlamalı süzgeç ile çalışacağımız zaman veri olarak elimizde bulunanlar:

u(n) : Mx1, n zamanında tap giriş vektörü

d(n) : n zamanında istenilen cevap

Bizden istenenler:

n+1 zamanında $\hat{w}(n+1)$ 'in tahmini değeri.

Hesaplamalar:

$$e(n) = d(n) - \hat{w}^H(n)u(n)$$

$$\hat{w}(n+1) = \hat{w}(n) + \mu u(n)e^*(n)$$

Sonuç olarak üretilen cevap: $e(n) = d(n) - \hat{w}^H(n)u(n)$

6. ALAN PROGRAMLANABİLİR KAPI DİZİLERİ (FPGA)

6.1 Genel

FPGA'ler(Field Programmable Gate Arrays) genel olarak kıymık(chip) tasarımında öncü tasarımda kullanılırlar. Yapılan tasarımlar FPGA üzerinde test edilip ilk ürün (Prototype) oluşturulur. FPGA'ler mantık kapılarından oluşur, içine gömülecek devreler birden farklı yöntemlerle hazırlanır ve FPGA'e gömülür. Konunun devam eden bölümünde neden FPGA seçildi, avantajları nelerdir, FPGA ile çalışma süreci nasıldır, bu çalışmada izlenen yol ile algoritmanın FPGA'e nasıl gömüldüğü anlatılacaktır.

6.2 FPGA'in Diğer Ortamlara Göre Avantajları

FPGA'lerin en büyük avantajları yeniden programlanabilir olmasıdır. Bu özelliği FPGA'lerin kıymık tasarımında yada sayısal sistem tasarımında AR-GE aşamasında tercih edilmesinin başlıca sebebidir. FPGA'ler sayısal devreler olup analog-sayısal karışık devreler de mevcuttur.

FPGA'lerin diğer bir avantajı çok esnek bir yapıya sahip olabilmesidir. Bugünlerde bir FPGA devresi 100~1000 kadar giriş çıkış bacağına sahip olabilmekte ve bu bacaklar bir den fazla standardı(Low Voltage Differantial Siganalling, LVDS; Transistor Transfer Level, TTL gibi) destekleyip istenildiği gibi kullanılabilmektedir. Örnek olarak bir FPGA devresi ile haberleşme hatlarında sistemler arasında geçişleri kontrol eden bir devre yapılabilir. Bir televizyon içerisinde görüntüyü işleyip paneli sürebilen devre yapmak mümkün olduğu gibi kendi başına USB ve Ethernet kontrolü yapabilen sistemin üzerindeki RAM ile ara yüz görevini üstlenip elde edilen veriyi de bir algoritmayla işlemek tek bir FPGA ile yapılabilmektedir.

FPGA ile yapılan bazı işlerin DSP'ler ile yapılması mümkündür ancak FPGA'ler yine de avantajını korumaktadır çünkü FPGA'lerin en büyük özelliği paralel çalışabilme özelliği DSP'lerde mevcut değildir. Paralel çalışma FPGA'in özgün niteliklerinden biri olup bir algoritmada FPGA'in kaynaklarına bağlı olarak birden fazla işlemin aynı anda yapılması mümkündür. FPGA'ler sadece işlem yapmak için değil işlemleri kapalı devre gibi, kendi içerisinde sahip olduğu bellek birimleri sayesinde dışarıdan belleğe ihtiyaç duymadan kullanılabilir. Örnek olarak, görüntü işleme algoritması için videonun birkaç satırının saklanması gerekmekteyse FPGA ile bu işlemi yaparken aynı zamanda da algoritma için

gerekli işlemleri aynı anda dışarıdan başka bir yardım gerekmeden, adres yönetimi dahil olmak üzere bir FPGA ile yapılabilir.

FPGA'lerin başka bir özelliği de, DSPlere göre oldukça esnek bir yapılarının olmasıdır. Örneğin DSP'lerde çarpıcıların bit genişliği belli iken bu FPGA'de kullanıcının ayarlaya bileceği bir durumdur. FPGA'de ayrıca çarpıcı sayısı, toplayıcıların sayısı, ve bit genişliği kullanıcının kontrolü altındadır.

6.3 FPGA ile çalışma

Bu çalışmada izlenmesi gereken süreç genel olarak algoritma geliştirme ve bunları FPGA ile uygulamada izlenebilecek bir yöntem olarak kabul edilebilir.

FPGA ile uygulama geliştirmek için öncelikle bir algoritma olması gerekmektedir. Bundan sonraki adım bu FPGA devresinin sistemin neresinde görev alacağı ve başka hangi işlemlerin FPGA'e ait olacağının belirlenmesidir. Daha sonra bu algoritmanın işaret akış diyagramı yapılır. Burada ortaya çıkan ürün bize algoritmanın zorluğu ve kritik işlemleri hakkında bilgi verir. Böylece doğru devre seçimi ve sistem mimarisi belirlenir. Daha sonra ise işaret akış diyagramı incelenerek paralel veya saatli çalışacak yerler, pipelened çalışacak yerler belirlenir ve ardından işaret zaman grafiği oluşturulur. Bütün bunlar sistemi oluşturmak adına önceden yapılmalıdır. Sistemi devreye gömmeden önce yapılması gereken son bir iş daha vardır. Gömülecek olan algoritmanın sabit noktalı(Fixed-Point) halinin çıkarılması gerekmektedir. Bu işlemi yaparken gerekirse algoritmanın performansı tekrar kontrol edilmelidir ve gerektiği takdirde algoritmada değişikliklere gidilmelidir.

Bu adımlardan sonra sistemi devreye gömebilmek için VHDL yada Verilog ile FPGA'e uygun sentezlenebilir kodlar oluşturulur. Burada anlatılanlar bölüm 7 de ayrıntılı olarak ele alınacaktır.

Uygun VHDL kodu üretildikten sonra devre üreticilerinin ya da üçüncü parti şirketlerin sağladığı sentezleyici programlar ile kodlar sentezlenebilir. Bu işlem sonucunda kodun devrenin ne kadar kaynağına ihtiyaç duyduğu ve performansının ne olacağı hakkında bilgiler elde edilmiş olur. Sentezlenebilir koda sahip olduktan sonra emin olmak için yine devre üreticilerinin ya da üçüncü parti şirketlerin sağladığı programlar yardımıyla bilgisayar ortamında canlandırma (simullation) yapılır burada sistemin doğru çalışıp çalışmadığı izlenebilir. Buraya kadar her şey yolundaysa Place and Route adı verilen süreç izlenir ve artık devreye gömebileceğimiz bit dosyası hazırdır.

Bu çalışmada, Synplify Pro, ISE Foundation ve Mentor ModelSIM araçları kullanılmıştır. Synplify Pro ve ISE Foundation sentez ve Place and Route için, Mentor ModelSIM ise canlandırma için kullanılmıştır. Bu programlar sonucu oluşturulan raporlar bölüm 8 de açıklanarak verilecektir.

6.4 Tez Çalışmasında Kullanılan FPGA Devresi

Bu çalışmada, algoritmayı gerçek zamanlı uygulayabilmek için Xilinx firmasının Spartan3 ailesinden XC3S200 FPGA'i kullanılmıştır. Bu FPGA'in özellikleri aşağıdaki çizelge 6.1'de verilmiştir.

Çizelge 6.1 XC3S200'ün özellikleri

XC3S200 Özellikleri	Miktar
Sistem Kapısı(System Gates)	200K
Mantık Hücreleri(Logic Cells)	4,320
18x18 Çarpıcılar(18x18 Multipliers)	12
Blok RAM(Block RAM Bits)	216K
Dağıtılmış RAM(Distributed RAM Bits)	30K
Sayısal Saat Üreticileri(DCMs)	4
I/O standartları/ En fazla I/O (I/O Standards/Max IO)	24/173

Sistem kapıları FPGA'in içindeki en küçük birimlerdir. Mantık hücrelerinin oluşturulmasında kullanılır. Bu mantık hücreleri üreticinin kendi teknolojisi olup başka markada farklı bir FPGA yapısı oluşturulabilir.

Blok RAM FPGA'in içinde RAM olarak kullanılabilecek kaynaklardır. Bunlar hali hazırda mevcuttur. Dağıtılmış RAM ise Mantık hücreleri kullanılarak tasarıma göre oluşturulabilir.

Sayısal saat üreticileri(DCM) üreticinin kaynak olarak sunduğu bir imkandır. DCM'ler doğru ayarlandığında bir saat kullanılarak birden farklı saat hızlarına sahip olmak mümkün olur. Örnek verecek olursak 50 Mhz bir saati DCM'ler sayesinde 25 Mhz yada 150 Mhz gibi hızlarda kullanabiliriz.

I/O standartları FPGA devresine giriş ve çıkış olabilecek standartlardır bu standartlardan bazıları LVTTL, SSTTL, LVDS, RSDS gibi…

7. VHDL VE VHDL ÇIKTISININ SİMULASYONU

7.1 Genel

VHDL'in İngilizce açılımı: "Very High Speed Integrated Circuits (VHSIC) Hardware Description Language" dır. Bu Türkçe'ye hızlı donanım tanımlama dili şeklinde çevirilebilir. VHDL yada benzeri Verilog FPGA'e gömülebilecek kodu üretmek için kullanılabilir. Diğer diller ile karşılaştırıldığında VHDL, donanıma en yakın dildir diyebiliriz.

Bu bölümde, VHDL'in temel yapısı, VHDL ile FPGA'in özelliklerini nasıl kontrol edebileceği anlatıldıktan sonra bu çalışma tamamlanırken Matlab'ten VHDL ve gerçek zamana geçiş işlemlerinin nasıl kontrol edildiği bunun için gerekli simülasyon ortamı anlatılarak uygulama bölümüne geçilecektir.

7.2 VHDL'in Temel Yapısı

VHDL diğer programlama dillerine benzemekle birlikte çok farklı özel halleri mevcuttur. VHDL'de sinyal tanımlamak değişken tanımlamaya benzer ancak nerede ve nasıl kullanacağına dikkat etmek gerekir. Ayrıca VHDL doğrudan çalışabilen sistemler olduğu için giriş çıkış ara yüzünü tanımlamak için kendi türleri vardır. VHDL bit seviyesinde işlem yapabildiği için istediğimiz şekilde kaynakların sınırlarını aşmadan tanımlamalar yapabiliriz bunlara örnekler verecek olursak.

```
entity lms_core is
tasarımın ismi lms_core bildirimi
  Generic(
    datawidth          : integer := 14;
    tap                : integer := 36
    );
Parametre edilebilir ara yüz
  Port (
     clk           : in std_logic;
     rst_n         : in  std_logic ;
     primary       : in std_logic_vector(datawidth-1 downto 0);
     desired       : in std_logic_vector(datawidth-1 downto 0);
     state_vec     : in std_logic_vector(5 downto 0);
     output        : out std_logic_vector(datawidth-1 downto 0)
  );
```

Çeşitli büyüklüklerden giriş çıkışlar

```
    signal error32      : std_logic_vector(31 downto 0);
    signal error        : std_logic_vector(17 downto 0);

    signal state        : integer range 0 to 63;
```

32, 18 bit ve 0 dan 63'e kadar değerleri alabilen tam sayı işaretlerinin tanımlanması

```
process(clk, rst_n)
saat ve reset'e göre bir işlem
            begin
                if (rst_n='0') then
reset ile belirtilen işaretlerin başlangıç değerlerini alması
                    weight_s(i)                 <= (others=>'0');
                    weight(i)                   <= (others=>'0');
                elsif (clk='1' and clk'event) then
saatin yükselen kenarında kütük ile işlemlerin yürütülmesi
                    if (state=9) then
                        weight_s(i)   <= sum (23 downto 12);
                    elsif (state=10) then
                        weight(i)     <= weight(i);
                    end if;
                end if;
             end process;
```

VHDL dilinde "process" altında yapılan işlemler saat kullanıldığı zaman kütüklü işlemlerdir ve buradaki işlemler sıralı olarak yapılmaktadır. Ancak aynı işlemi birden fazla yerde yaptırmak mümkün olduğundan paralel çalışma sağlanmış olur. Ayrıca işlemlerin kütüklü olması pipelined işlemlerin gerçekleşmesine olanak sağlamaktadır.

7.3 VHDL ve FPGA

VHDL diğer diller ile karşılaştırıldığında oldukça katı kuralları olan bir dildir. Bunun faydası programcının FPGA'e yükleyeceği kodun, her parçasını birebir karşılaştırılabilir yapması gerekmektedir. Bunun yanında sorun yaratmayacak fakat kural gereği hata sayılan durumlar da dezavantajlı bir durumdur. Genel olarak VHDL ile FPGA'i kontrol edebilmek için kuralları tam olarak bilmek gerekmektedir bu da FPGA'i tam olarak istenildiği gibi kullanılması anlamına gelmektedir.

FPGA'de tanıtılan "process" ile sıralı(sequential) işlemler yapılır. Bu sayede hem pipelined

hem de paralel işlemler mümkün olmaktadır. Böylece gerçek zamanlı uygulamalarda mükemmel bir avantaj kazanmış oluruz. Pipelined özellik sayesinde ilk başta oluşan bir gecikmenin ardından her bir saat çevriminde çıkış alabilmek mümkün olmaktadır. Bu da FPGA'in sınırları içinde kalma kaydıyla uygulanan saatin hızına bağlıdır. Sadece saati değiştirmek ile çıkış hızını ayarlayabiliriz.

Paralel özellik sayesinde ise ağır işlem yükü olan yerlerde aynı "process" ile aynı işi yapan birden fazla motor (engine) tanımlanabilir. Böylece işlemler paylaştırılmış olur. Örnek verecek olursak işaret işlemede bu yöntem sıklıkla kullanılır. LMS algoritmasında 12 taplı bir süzgeci düşünürsek: eğer DSP (sayısal işaret işlemcisi) ile işlem yapıyor olsaydık. Her bir çarpma için 3-4 saat çevrimi ve her bir çarpmayı toplamak için 12 saat çevrimi gerekmektedir. Toplamda 48-60 saat çevriminde bir "t" anı için süzgeç çıkışını almış oluruz. bu durumda DSP'nin çalışma saati süzgeç girişindeki işaretin örnekleme hızının 48-60 katı olmalıdır.

FPGA'de ise 12 çarpma aynı anda yapılır ve toplamayı da paralel yaparak toplamda 5 saat çevriminde çıkış elde edilir. DSP'den farklı olarak bu 5 saat çevrimi gecikme "latency" olarak adlandırılır. Bu gecikme bir kez yaşanır ve ardından süzgeç girişindeki hızda çıkışlar elde edilir. Bu bize süzgeç girişindeki bir saat ile aynı hızda çıkış verebileceğimiz anlamına gelir.

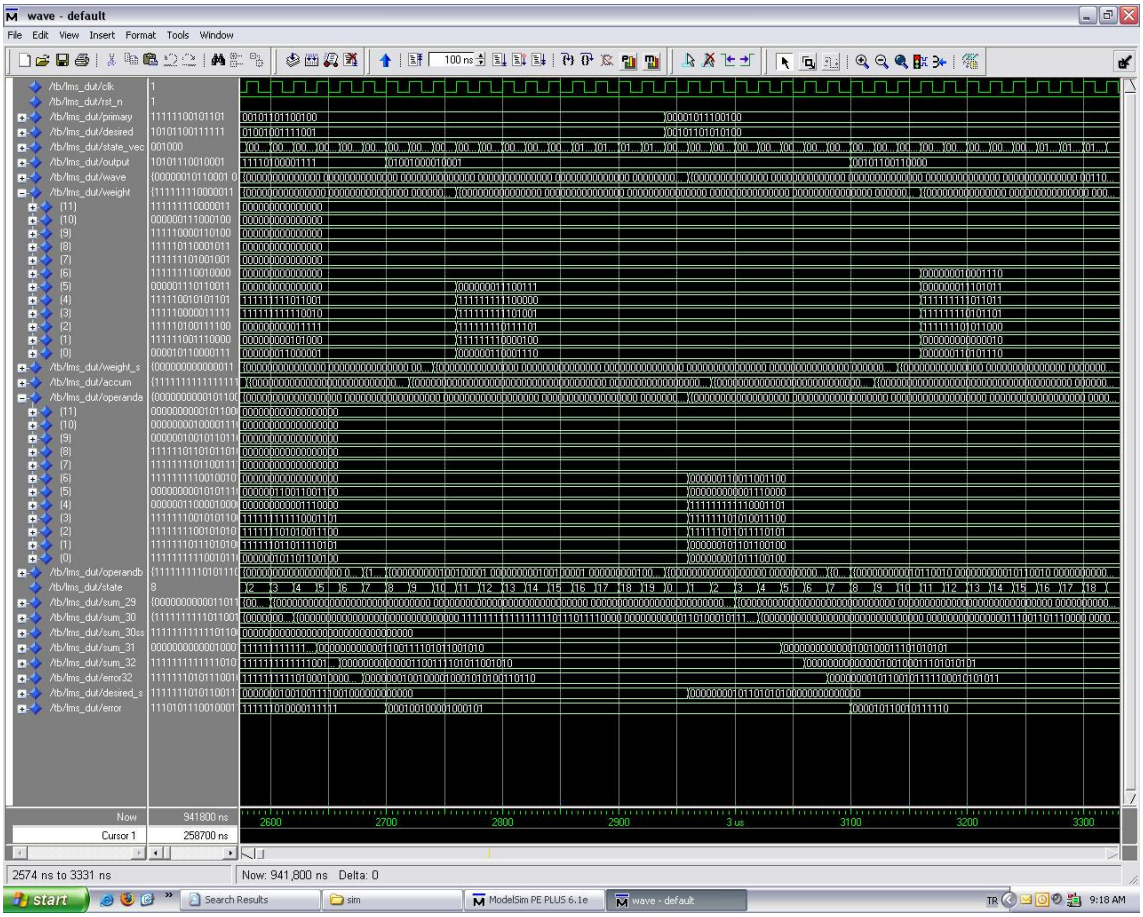
Yukarıda anlatılanlar hem FPGA'in pipelined özelliğini hem de paralel çalışabilme özelliğini göstermektedir. FPGA'in bunun yanında başka özellikleri vardır. Saate ihtiyaç duymadan beklediği işaretlerde değişme oldukça işlem yapabilme özelliği FPGA'nin doğasında olup gerekli olduğu yerde başka bir şeye ihtiyaç duymadan kullanabildiğimiz avantajlardan bir başkasıdır. Bunun yanında FPGA devrelerinin içerisinde bulunan blok RAM'ler ve saat yöneticileri dışarıdan ek bir kaynağa ihtiyaç duymadan çeşitli uygulamalara cevap verebilen özelliklerdir.

7.4 VHDL Simülasyonu

C yada Matlab kodundan VHDL koduna geçildiğinde algoritmada değişiklikler kaçınılmazdır. Bu sebeple mutlaka test yapılmalıdır. Bunun içinde çeşitli üreticilerin simülasyon programları mevcuttur. Bu çalışmada Mentor Graphics'in ModelSim'ini kullanılmıştır. ModelSim'in "textio" özelliği sayesinde Matlab'te oluşturulan sabit nokta girdileri ModelSim'de işlenerek FPGA de çalışıyormuşçasına çıktılar test edilmiştir.

Yukarıda bahsedilen sabit nokta model ModelSim'e yüklenerek, saat ve reset işaretleri atandıktan sonra Matlab girdileri okunur. Bu girdiler gerçek zamanlıymış gibi ModelSim

içerisinde işlenir ve sonra gerçek zamanlıymış gibi çıktılar yazılır. Bu işlemi yapan kod parçacığı test programı, tb(Test Bench) olarak adlandırılır. TB'e bağlı kodlara ise test altındaki tasarım anlamına gelen dut(Design Under Test) denir. TB ile gerçek zaman modeli oluşturulur. Daha önce sabit nokta modeli oluşturulan Matlab kodunun kullandığı giriş verileri ve Matlab'in ürettiği karşılatırma amaçlı çıkış verileri tb tarafından sisteme alınır. Bu veriler 14 bitlik verilerdir. Gerçek zamanlı çalışacak VHDL kodu bu veriler ile test edilir. VHDL sonucunda oluşan 14 bitlik veriler ile Matlab çıktıları burada karşılaştırılır. İstenirse daha sonra bu çıktılar Matlab'te de kontrol edilir. Şekil 7.1 de ModelSim'de simülasyon esnasında bir görüntü örneklenmiştir.



Şekil 7.1 ModelSim'den örnek

Görülen pencerede ModelSim'de tasarımda tanıtılan sinyaller ModelSim'in Wave'ine yüklenmiştir. Matlab'ten alınan veriler her bir saat çevriminde alınarak gercek zamanlı veri geliyormuş gibi tasarıma yüklenir. Tasarlanan devreye gelen işaretler Wave aracılığı saate göre aldığı değerler incelenebilir ve çıkan sonuçlar tekrar Matlab'te karşılaştırılır.

8. LMS ALGORİTMASININ UYGULANMASI

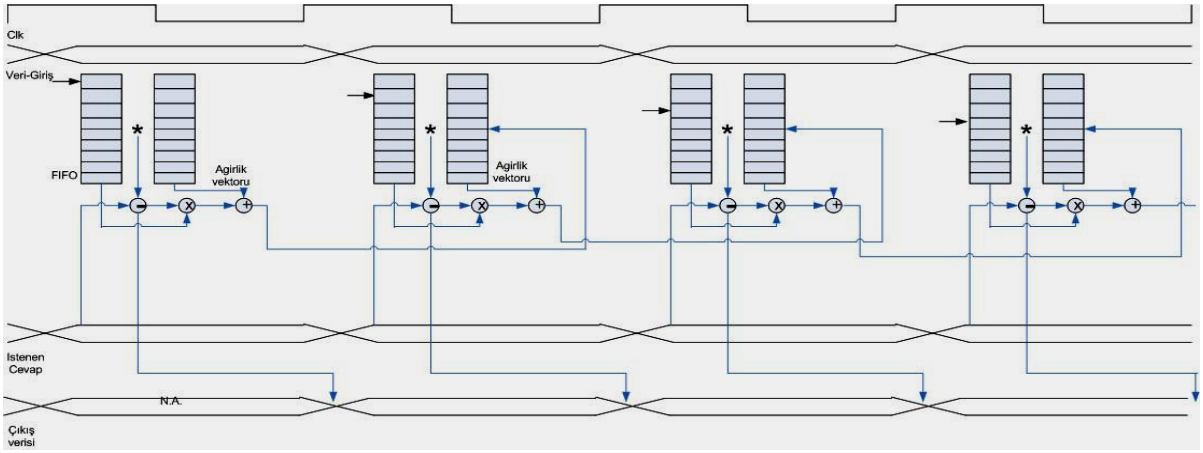
8.1 Genel

LMS algoritmasından daha önce bahsedilmişti, bu bölümde LMS algoritması gerçek zamanlı çalışmasının nasıl yapıldığı, donanımın getirdiği kısıtlamalardan dolayı uygulanan Sabit-Nokta modeli ve FPGA içindeki işaret akışı anlatılacaktır.

8.2 Gerçek Zamanda LMS

LMS'in gerçek zamanda çalışmasını ele aldığımız zaman algoritmanın yaptığı işi bozmadan işleyişini değiştirmek gerekmektedir. Gerçek zamanda veriler sürekli gelmektedir. Öyle bir sistem oluşturulmalıdır ki veri geldikçe sistem çıkış oluşturabilmelidir. LMS tabanlı kendinden uyarlamalı süzgeçte bu zorluğa ilave olarak doğasındaki kendinde uyarlı olma durumu sistemin yapısını daha da zorlaştırmaktadır. Çünkü bir girişten bir çıkışa olan gecikme bir saat çevrimini(clock cycle) geçmemelidir. Kendinden uyarlı sistem bu çıkışı kullanacağı için yeni veriler geldiğinde çıkışlar hazır olmazsa sistem çöker.

Bu yüzden algoritmanın gerçek zamanlı işaret zaman akışı şekil 8.1 deki gibi olur.



Şekil 8.1 Gerçek zamanlı LMS işaret-zaman akışı

Şekil 8.1'de işlemler basitleştirilerek verilmiştir. Algoritmanın FPGA içerisindeki sinyal akışı bölüm 8.4 te anlatılacaktır. Şekil 8.1 incelenirse, Clk olarak gösterilen çalışma saati örnekleme oranı olarak alınabilir. Bu saatin her yükselen kenarında yeni veriler 14 bit olarak gelmektedir. Bir sonraki yükselen kenara kadar algoritma sonucu üretmiş, yeni ağırlık değerlerini güncellemiş olmalıdır. Bunun sebebi yeni verilerin bu yeni ağırlıkları kullanacak

olmasıdır. Aslında bu türden bağımlılıklar FPGA'in çalışmasına uygun olmamasına rağmen FPGA'in getirdiği paralellik ve esnek yapısı sayesinde birden fazla saat tanımlayıp, sonradan tanımlanan saat hızlarının düşük tutularak çıkış üretilebilmektedir.

Algoritmanın bir çıkış vermesi için tap sayısının iki katı kadar toplama, tap sayısının üç katı kadar çarpma ve bir adet çıkarma işlemi gerekmektedir. Eğer DSP kullanılsaydı, DSP'nin bir adet toplayıcıya ve bir adet çarpıcıya sahip olduğunu ve her işlemi bir saat çevriminde yaptığını göz önüne alarak, her bir veri için bu işlemlerin en az tap sayısının beş katı saat çevriminde tamamlanırdı. Ancak FPGA'in getirdiği paralellik sayesinde sadece başlangıçta bir gecikme(Latency) ardından her bir saat çevriminde örnekleme hızında çıkış alabilmek mümkün olmaktadır. Buradaki avantaj örnekleme dışındaki saat hızı DSP'e göre daha yavaştır. Ayrıca kullanılan FPGA devresi daha büyük olsaydı ikinci bir saate gerek duymadan sadece örnekleme hızında saat ile çalışılabilen bir sistem tasarlanabilir. FPGA'ler ile DSP'lere göre düşük saat hızlarında aynı işlerin yapılmasından dolayı sistem tasarımında avantajlar elde edilir.

8.3 Sabit Nokta Geçişi

Matlab'te veya C gibi ortamlarda geliştirilen algoritmalar olduğu gibi FPGA'de gerçekleştirilemez. Bunun sebebi ortamın sahip olduğu kaynakların kısıtlı olmasıdır. Bu yüzden yapılması gereken algoritmanın çalışmasını bozmadan göz ardı edilebilir kayıplarla sonsuz çözünürlükten sabit noktalı çözünürlüğe geçmektir. Bu durumda kesme ve yuvarlama (Truncation, Saturation) yapmak gerekir. Eğer bu adımlar dikkatli yapılmazsa sistemdeki performans çok kötu olabilir ve LMS tabanlı kendinden uyarlamalı süzgeç uygulamasında sistem doğasındaki geri beslemeden dolayı hata kontrol dışına çıkabilir. Bu gibi durumlarda izlenecek iki yol vardır.

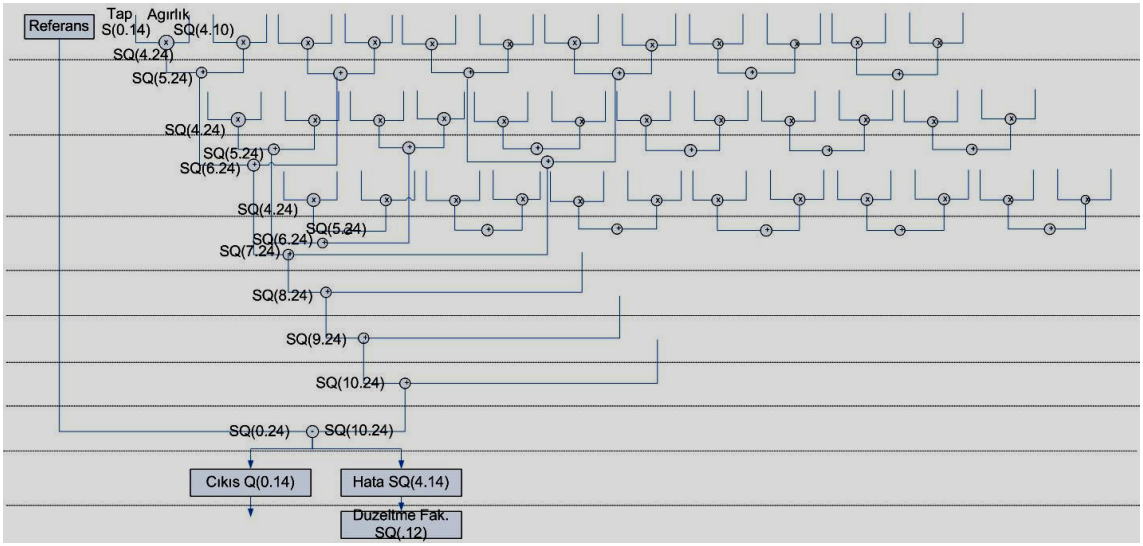
Birinci yolda sistem mimarisi (system architecture) oturtulduktan sonra aritmetik hesapla kesme ve yuvarlama yapılır sonra algoritma çeşitli metodlar ile test edilebilir. Ama buradaki sakınca, algoritmanın zaten gerçek zamanlı sisteme geçişinde ve buradaki güncellemeden dolayı değişmiş olması, hata durumunda hatanın tespitini zorlaştırır. Bu durumda ikinci yol tercih edilebilir.

İkinci yolda ise öncelikle Matlab yada C gibi ortamlarda algoritma Sabit noktalı modele çevrilebilir. Önce performans orada test edilir daha sonra FPGA'e geçilir. Bu çalışmada ikinci yol seçilmiştir.

8.4 FPGA İşaret Akışı

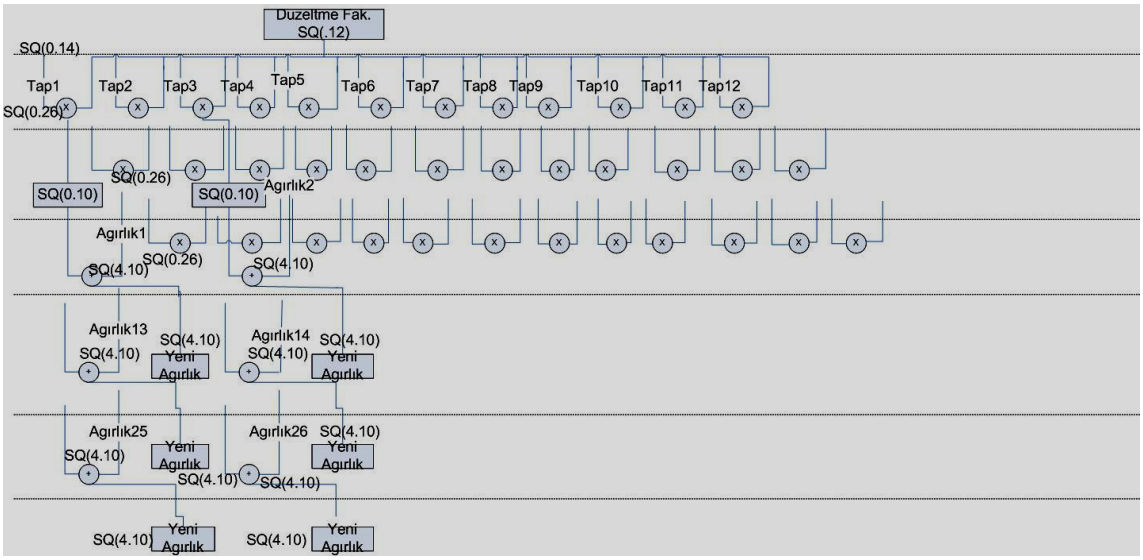
Daha önce de belirtildiği gibi herhangi bir algoritmayı doğrudan FPGA'e gömmek pek kolay olmaz. Bu yüzden yukarıdaki adımları tamamladıktan sonra FPGA'e geçilebilir. Şekil 8.2 ve Şekil 8.3'te sabit noktalı gerçek zamanlı LMS algoritmasının sinyal akışı izlenebilir.

Şekil 8.2 ve Şekil 8.3 dikkatle incelenirse ilk olarak sesin örneklenmesinde sonsuz çözünürlükten 14 bite geçildiği görülür. Daha sonra ağırlık vektörü Q(4.10) formatında alınmıştır. Bu iki sinyale işaret biti de eklendiğinde ses ve ağırlık vektörleri sırasıyla SQ(.14) ve SQ(4.10) olmuştur. "S" işaretli veri, sadece "Q" işaretsiz veri anlamında kullanılmıştır. Daha sonra bu iki değer çarpıldığında sonuç SQ(4.24) olmuştur bu değerler 36 taplı bir süzgeç için toplandığında süzgecin çıkışı SQ(10.24) olmaktadır. Şüphesiz ki bu işlemler sonsuz çözünürlükte olsaydı daha doğru değerler çıkardı ancak buradaki değerler insanın hissedemeyeceği ya da performansı etkilemeyen kayıplarla başarıyı yakalamaya yetmektedir.



Şekil 8.2 Gerçek zamanlı LMS algoritmasının FPGA üzerinde işaret akışı-1

FPGA her ne kadar esnek olsada sınırsız kaynakları bulunmaz. Kullanılan devrede çarpıcı sayısı 12'idr. FPGA devresinde aynı anda 12'den fazla çarpma işlemi yapılırsa çarpıcılar mantık kapılarıyla gerçekleştirilecektir. Bu sonuç olarak yavaşlamaya ve alan tüketimine neden olur. Bu yüzden algoritmayı durumlara(State) ayırarak aynı kaynağı farklı zamanlarda farklı işaretlere tahsis etmek gerekir. Bu işlem kaynak paylaşımıdır (Resource Sharing). Kaynak paylaştırma, eğer aynı zamanda o kaynak farklı işaretler tarafından kullanılmıyorsa işlem basittir. Ancak aynı anda aynı kaynağa ihtiyaç varsa bunun kontrolü durumlar aracılığıyla oluşturulup paylaşım kontrol altına alınmalıdır.



Şekil 8.3 Gerçek zamanlı LMS algoritmasının FPGA üzerinde işaret akışı-2

Bu çalışmada 36 taplı filtre tasarlanmıştır. FPGA'deki çarpıcı sayısı 12 olduğu için bir anda yapılabilecek çarpım sayısını 12 ile sınırlamak mantıklı olur. Durum makinesi çarpıcılara göre ayarlanabilir.

Durum 1. veriler alınır ve depo biriminde saklanır(FIFO Buffer). Bu depo 36 tane 15 bitlik veriyi saklayabilmektedir.

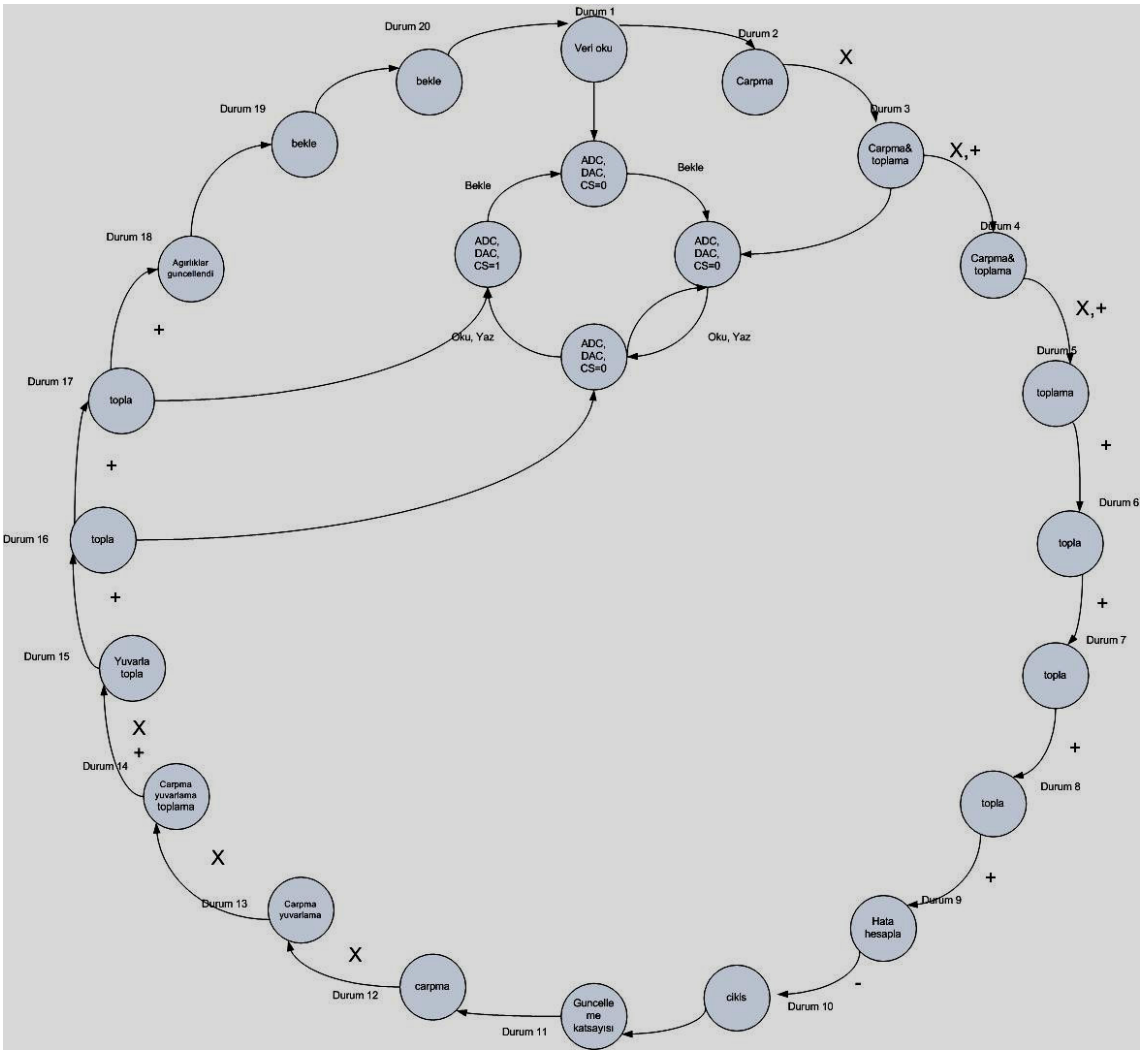
Durum 2. 36 adet veriden 12 tanesi alınır ve aynı yapıdaki ağırlık katsayılarının tutulduğu depodan 12 tane ağırlık ile ilk çarpma yapılır. Bu çarpma SQ(4.24) formatındadır.

Durum 3. diğer 12 adet veri bunlara uygun ağırlıklar ile çarpılır. Önceki durumda oluşan 12 çarpma 30 bitlik 6 adet toplayıcıda bir biri ile toplanır.

Durum 4. kalan 12 adet veri uygun ağırlıklar ile çarpılır. Önceki adımda oluşturulan çarpımlar, önceki adımda belirlenen toplayıcılarda toplanır ayrıca önceki adımda oluşan toplamlar 31 bitlik 3 adet toplayıcıda birbirleri ile toplanır.

Durum 5. toplama işlemleri 30, 31 ve 32 bitlik toplayıcılarda piped olarak devam eder. Bu işlemler sırasında bazen verileri kütüklerde tutmak gerekebilir(Registered), bunun sebebi gerçek zamanda işaretleri sürekli olarak değişmesidir. Durum 9 a kadar toplamalar devam eder ve Durum 9 da enine süzgecin çıkış SQ(10.24) formatında hazırdır.

Durum 9. burada LMS yönteminde hata, LMS yöntemi ile çalışan süzgeçte ise çıkış bulunur.bulunan değer SQ(10.24) formatındadır.



Şekil 8.4 Durum makinesi

Durum 10 ve 11 de gerekli kırpmalar ve doymalar ile süzgeç çıkışı hazır hale gelir.

Durum 12. daha önce kullanılan çarpıcılara oluşan SQ(.12) formatındaki hata verilerek ağırlık hesaplamaları için gerekli değerler üretilir. Hata formatındaki ufalma algoritmadan kaynaklanmaktadır. Ancak algoritmadaki çarpma yerine bit kaydırma işlemi ile kaynaklardan tasarrufa gidilmiştir.

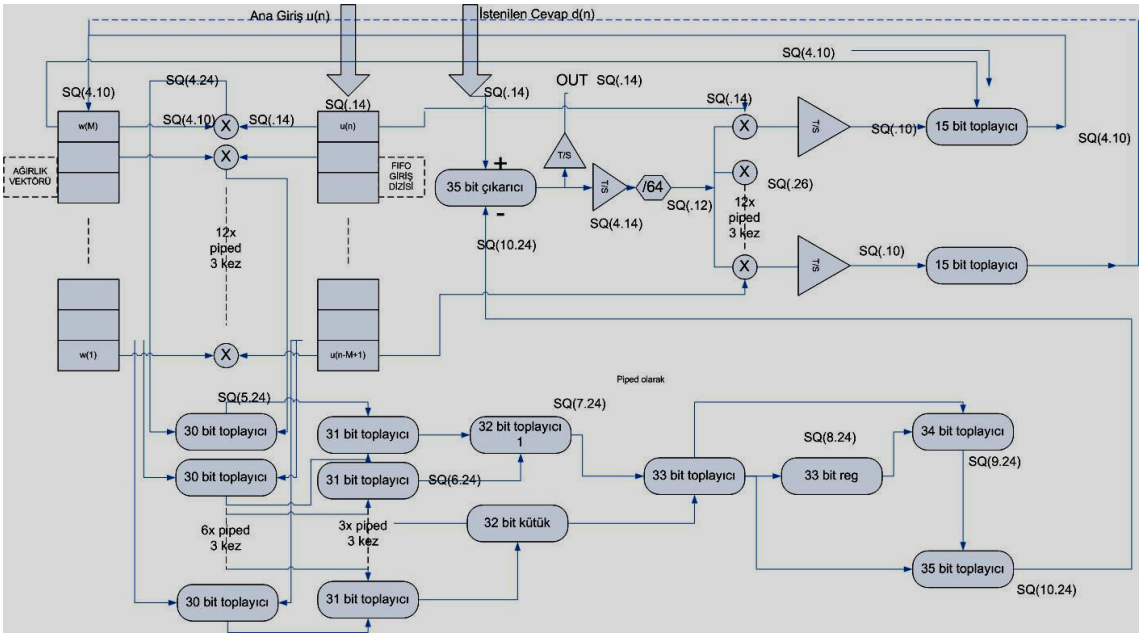
Durum 13. çarpma işlemi devam ederken önceki durumda üretilen bu değerler doyurulur.

Durum 14. çarpma işlemi devam ederken önceki durumda elde edilen veriler ilgili ağırlıkları günceller.

Durum 15. önceki durumda elde edilen veriler ilgili ağırlıkları günceller.

Durum 16. son ağırlık güncellemesi de tamamlanır. Durum 17, 18 ,19 ve 20 sayısal analog ve analog sayısal çeviricilerin ihtiyaç duyduğu bekleme zamanlarıdır. Açıklanan bu durum makinesi Şekil 8.4 de durum makinesi olarak gösterilmiştir.

Yukarıda anlatılan sonlu çözünürlüğü ayarlamak pekte kolay olmamaktadır. Bunun için algoritmanın davranışının iyi bilinmesi ve kesme yapılacak yerlerde verinin ne tarafa eğilimli olduğunun istatik bilgisinin değerlendirilmesi önemlidir. Örnek verecek olursak sabit noktalı bir çarpma işleminde çarpma sonucunda atılacak bitlerin en değerli kısımdan mı yoksa en değersiz kısımdan mı yapılacağına karar vermek için çarpma sonucunun mantıksal olarak neye yakınsadığı bilinmelidir. Eğer çarpma sonucu küçülten veriler ile yapılıyorsa en değerli kısımdan kesme yapmak mantıklı ve doğru olur. Sabit noktalı işlem aritmetiği belirlenirken aslında bir yandan da sistem mimarisinin temelleri atılmış olur. Çünkü alınan bu karalara göre kaynak tüketimi ve kullanımı da işin içine katılarak sistem mimarisi oluşturulur. Bu çalışmada oluşturulan sistem mimarisi şekil 8.5 te gösterilmiştir.



Şekil 8.5 Sistem mimarisi

Burada önemli olan diğer bir nokta sistem mimarisinin doğru yapılmasıdır. Bunun için işlemler sırasında dikkat edilecek hususlar:

Ortak kaynak kullanımının en iyi düzeyde olması

Hızlı çalışabilen bir sistem için gerekli paralelliğin ve pipelined mimarinin oluşturulması

Sistemin mümkün olduğunca parametre edilebilir olması

Sistemin kaynaklarının aşılmamış olunması

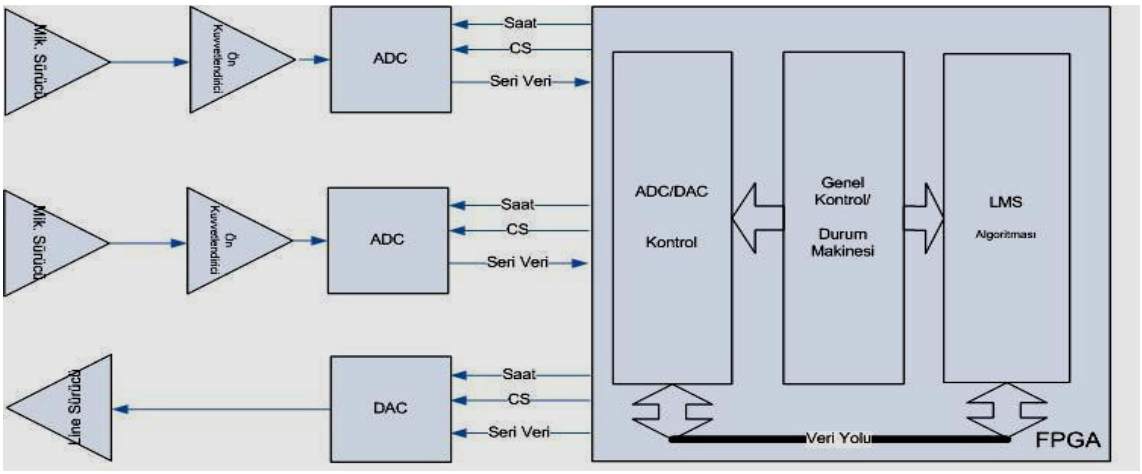
Üretilen kodun sentezlenebilir olmasıdır.

Bu noktalara dikkat edildiği sürece başarılı bir sistem mimarisi oluşturulup gerçek zamanda FPGA üzerinde çalışan sistemler elde edilebilir.

Bu çalışmadaki 36 taplı süzgeç ve giriş çıkış işlemi için kullanılan sistem 1920 dilimli(slice) bir Spartan3 FPGA'inde 1918 dilim kullanmıştır. Durum makinesinin kullandığı saat örnekleme hızının 20 katıdır ve bu devrede durum makinesinin kullanacağı hız maksimum ~43 MHz olabilir. Bu da ~2MHz örnekleme hızında çalışmanın mümkün olduğunu gösterir. Fakat ses için maksimum 40 KHz örnekleme hızı yeterlidir.

8.5 Uygulamada Veri Alış-Verişinde Kullanılan Devre

FPGA'e ses işaretini sayısala çevirip vermek gerekir. Bunun için mikrofon ön kuvvetlendiricisi, analog sayısal(ADC) ve sayısal analog(DAC) çeviriciler kullanılmıştır. Ayrıca çalışma sonucunu bilgisayara ses dosyası olarak kaydedebilmek için operasyonal kuvvetlendiriciler ile işaret seviyesini değiştirecek devre eklenmiştir. ADC olarak Anolog Devices'in AD7940 tümdevresi ve DAC larak Anolog Devices'in AD5640 tüm devresi kullanılmıştır bu devreye ait blok diagramı şekil 8.6' da verilmiştir. Ek-3 te veri alış verişinden sorumlu devreye ait şema ve Ek-4'te baskı devresi verilmiştir.



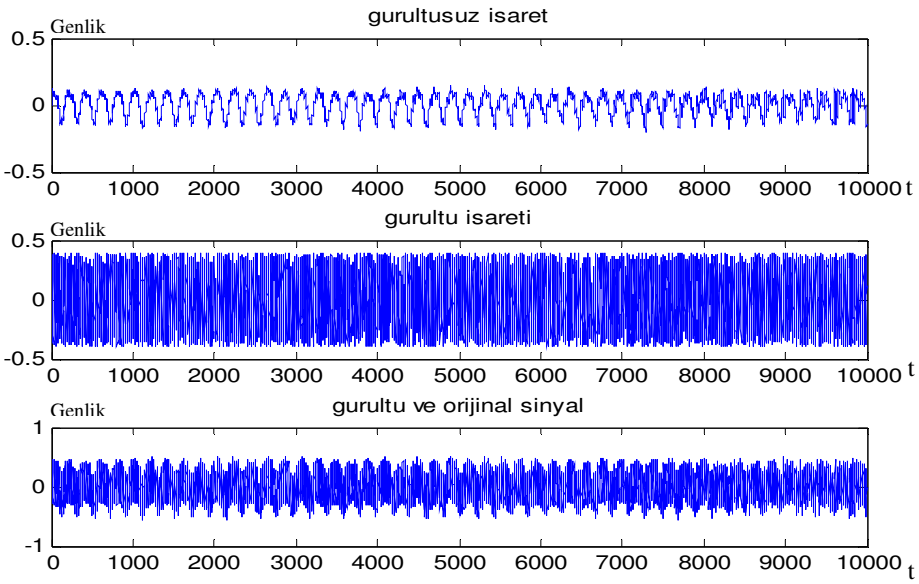
Şekil 8.6 Veri alış-verişinde kullanılan devre blok diagramı

Şekil 8.6'da bu çalışmada kullanılan veri-alışverişinden sorumlu devre bloğu görülmektedir. Devre üzerinde bulunan ADC ve DAC tüm devreleri seri veri arayüzüne sahiptir. Bu tüm devre elemanları FPGA tarafından üretilen gerekli saat ve CS (Devre seçme, Chip select) işaretleri ile kontrol edilir. Seri paralel ve paralel seri çevrimi FPGA içerisinde yapılmaktadır. Bu çevirimler ve kontrol işaretleri FPGA içerisinde bulunan durum makinesine göre üretilir. Bu işlemler sonrasında elde edilen veriler LMS çekirdeğinde işlenir ve çıkış işareti FPGA tarafından kontrol edilen DAC çevirici ile daha sonra sonuçların kontrol edilebilmesi için gerçek zamanlı olarak bilgisayarın hat (Line in) girişine verilir.

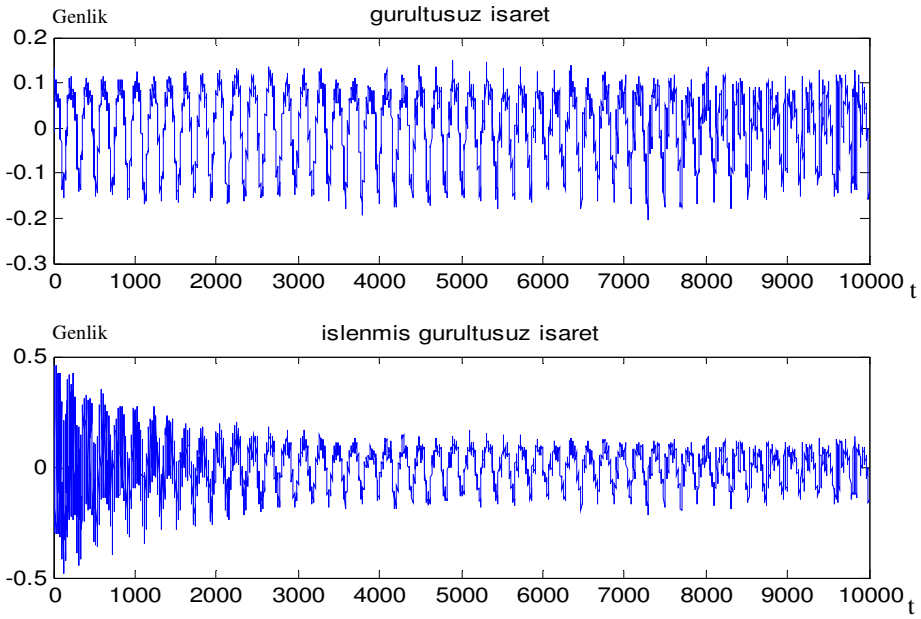
8.6 Uygulama Sonuçları

Uygulama sonuçları gürültü ve bilgi işaretlerinin karşılaştırılması ile görülebilir. Gerçek ortamda bulunan gürültüyü ve sesi aynı anda ayrı ayrı kayıt etmek mümkün değildir. Bu yüzden LMS algoritmasının performansını ölçebilmek için bilgisayar ortamında oluşturulan işaretlerin Matlab kullanarak sabit noktalı LMS algoritmasından geçirilmesi ile oluşan işaret gürültü oranı, SNR (Signal to noise ratio) performans değerlendirmede ikinci bir parametre olarak kullanılabilir. Bu işlemler sonucunda elde edilen SNR'lar gerçeğe yakın sonuçlar verecektir çünkü ModelSim ile oluşturulan VHDL kodunun, Matlab sabit noktalı modele göre doğruluğu sağlanmıştır. Aşağıdaki şekil 8.7'de beyaz gürültü ile insan konuşmaşına ait; gürültüsüz ses, gürültü ve konuşma, gürültü ve LMS süzgeç çıkışı gösterilmektedir.

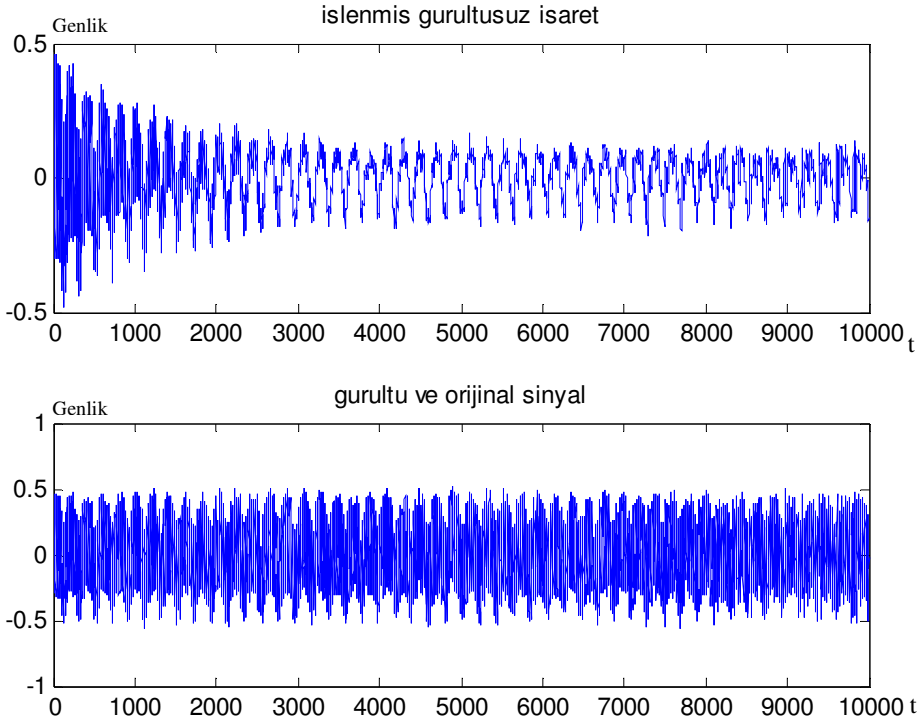
a)



b)



c)



Şekil 8.7 a) Gürültüsüz ses ve beyaz gürültü, b) Gürültüsüz ses ile işlenmiş ses, c) İşlenmiş ses ile gürültülü ses

Şekil 8.7 de gösterilen seslere ait giriş SNR ve çıkış SNR oranları aşağıdaki denkleme göre hesaplanmıştır.

Gürültüsüz orijinal ses r(n), gürültü ve ses birleşimi r(s,n) ve işlenmiş ses x(n) olmak üzere:

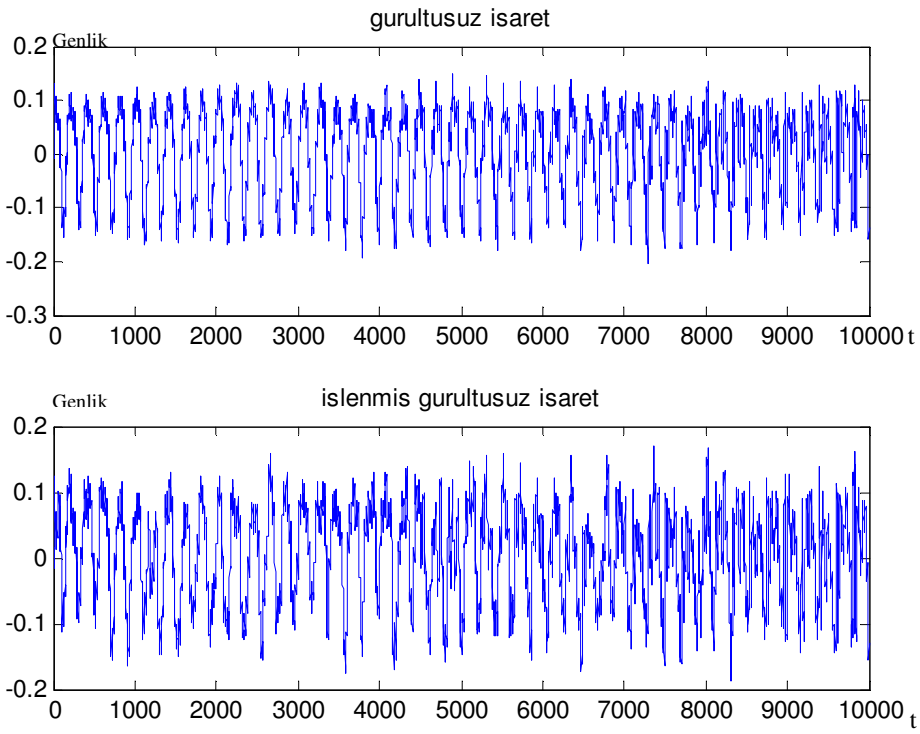
$$SNR_i = 10 * \log_{10}\left(\frac{\text{var}(r(n))}{\text{var}(r(s,n) - r(n))}\right) \quad 8.1$$

$$SNR_o = 10 * \log_{10}\left(\frac{\text{var}(r(n))}{\text{var}(x(n) - r(n))}\right) \quad 8.2$$

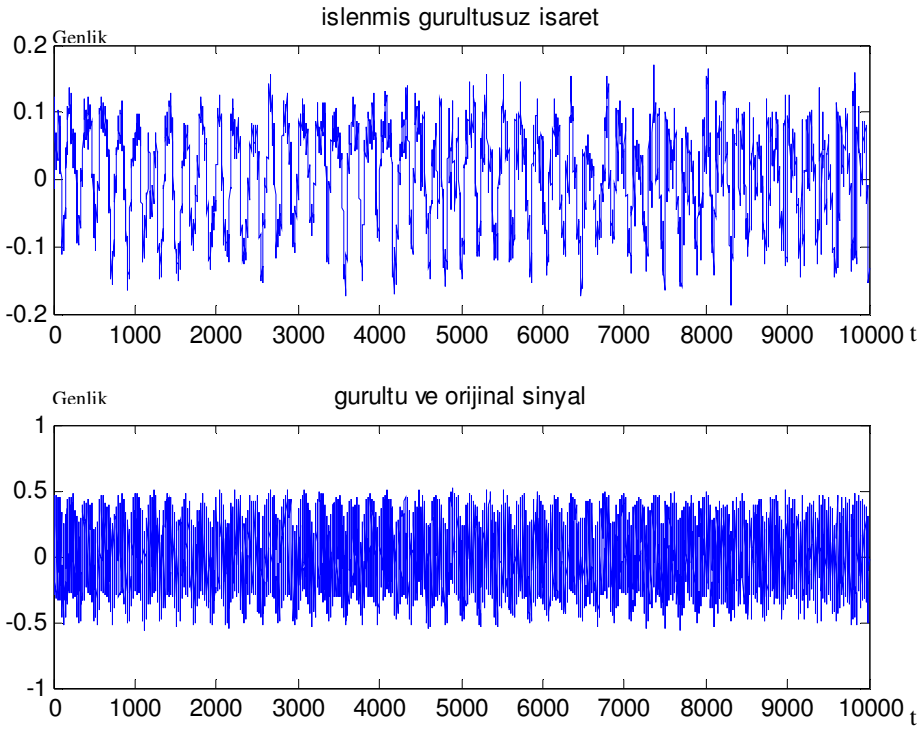
denklem 8.1 ve 8.2 de sırası ile giriş SNR ve çıkış SNR hesaplama formülleri verilmiştir. var() şeklinde gösterilen operatör varyans alma operatörüdür. Bu denklemlere göre beyaz gürültü uygulanan sistemde giriş SNR'ı -9.2378 db ve çıkış SNR'ı 2.5921 db bulunmuştur.

Aynı işlemi aynı ses için, Goldwave adlı bir program yardımıyla üretilen kahverengi gürültü(Brown noise) uygulanmış bir sistem için yaparsak aşağıdaki şekil 8.8 elde edilir.

a)



b)



Şekil 8.8 Kahverengi gürültü uygulanmış sistemde a) orijinal ses ve işlenmiş ses, b) gürültülü ses ve işlenmiş ses.

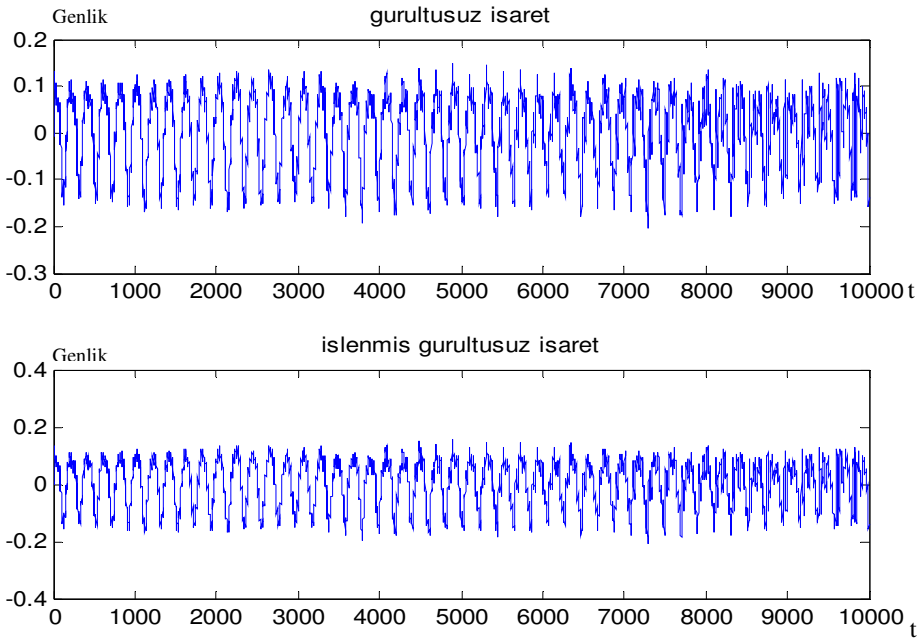
Bu sistemde giriş SNR'ı -9.5687 db ve çıkış SNR'ı 6.8030 db bulunmuştur.

Aynı ses işaretine Goldwave programı yardımıyla jet gürültüsü eklendiğinde elde edilen sonuçlar şekil 8.9'da verilmiştir. Bu sistemde giriş SNR'ı -8.5475 db ve çıkış SNR'ı 16.9914 db bulunmuştur. Elde edilen SNR sonuçları aşağıdaki çizelge 8.1'de topluca verilmiştir.

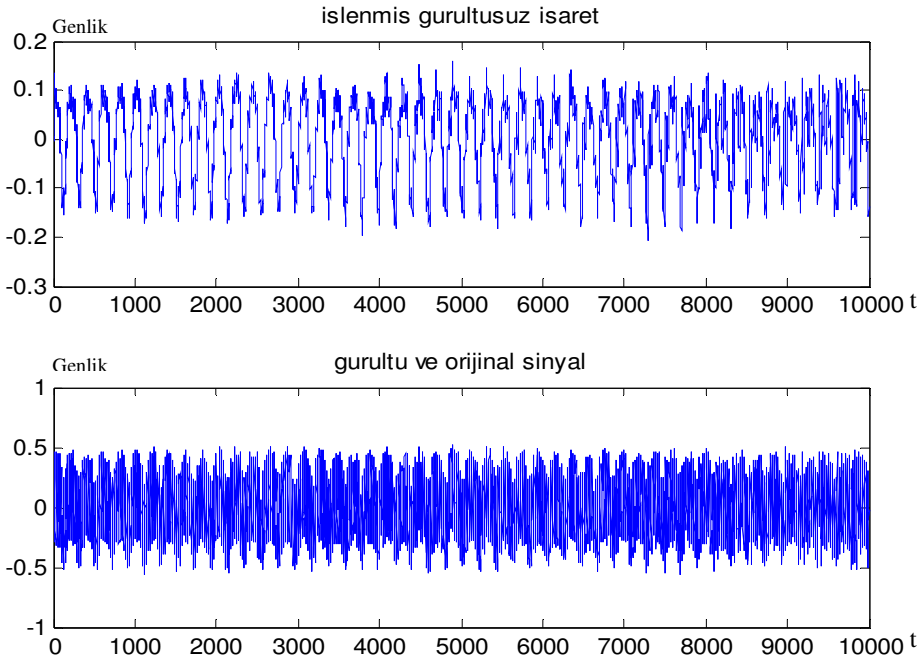
Çizelge 8.1 Sabit noktalı LMS'te süzgeç giriş ve çıkışına ait SNR oranları

Gürültü Çeşidi	Süzgeç girişi SNR (db)	Süzgeç çıkışı SNR (db)
Beyaz Gürültü	-9.2378	2.5921
Kahverengi Gürültü	-9.5687	6.8030
Jet Gürültü	-8.5475	16.9914

a)



b)



Şekil 8.9 Jet gürültüsü uygulanmış sistemde a) orijinal ses ve işlenmiş ses, b) gürültülü ses ve işlenmiş ses.

9. SONUÇ

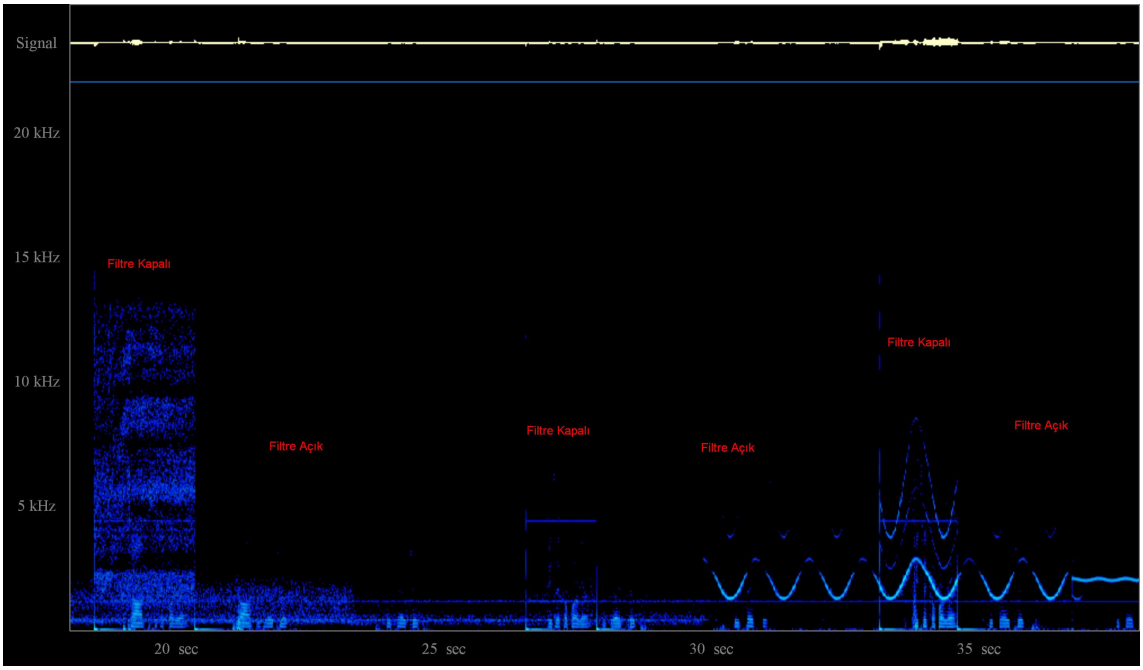
Gürültünün tamamen öngörülemez olduğu ve belli bir periyodu olmadığı hallerde kendinden uyarlamalı süzgeçler kullanılabilir. Kendinden uyarlamalı süzgeçler, süzgeç girişinden aldıkları bilgiyi kendi ağırlıklarını güncellemede kullanarak çıkışı girişe göre düzeltmeye çalışır. Günümüzde oldukça geniş uygulamaları bulunan kendinden uyarlamalı süzgeçlerden biri de LMS tabanlı süzgeçlerdir. LMS tabanlı süzgeçler sürekli olarak aldıkları bilgileri işleyerek kendinden uyarlamaları olmaları gereği kendini güncelleyerek çıkış üretir ve performansı yeterli görülür. Genel olarak gerçek zamanlı süzgeçlere geçildiği zaman bulunduğu sistemde donanımdan kaynaklanan problemleri hesaba katmak gerekir.

Donanım, algoritmaların yanında ikinci bir problemdir. Bulunan en uygun algoritmanın donanımı imkansız olabilir bu yüzden algoritma ile donanım arasında en iyi ilişkiyi kurabilmek çözümün diğer parçasıdır. LMS diğer algoritmalara göre donanıma uyarlaması bakımından avantajlara sahiptir. Bu çalışmada LMS süzgeçler, gerçek zamanda çalışması ve algoritmadan donanıma geçiş süreci dikkatli bir şekilde anlatılmıştır.

Bu çalışmda 5. bölüm incelenirse algoritmanın sonuçları görülebilir. Bilgisyar ortamında oluşturulan tesetlerde, gürültünün başarılı bir şekilde azaltıldığı OKH grafiklerinden ve işlenmiş işaret ile gürültülü işaretin karşılaştırıldığı grafiklerden görülebilir. Bu karşılatırmalar görülebilirliği artırabilmek için düşük frekanslı işaretler ve ayrıca gerçeğe benzemesi için insan sesinin konuşma sırasında oluşturduğu frekanslara yakın frekanstaki işaretler için yapılmıştır.

Donanıma gerçerken oluşan farka ise 8. bölümde değinilmiştir. LMS'in diğer algoritmlara göre avantaj olarak kabul edilen basit yapısı performansın kabul edilebilir düzeyde kalmasını sağlarken, gerçek zamanlı uygulamalara geçişteki kısıtlamalar bilgisayar ortamından az ölçüde de olsa farklı sonuçlara neden olmuştur. Bu farklara yol açan en önemli sonuçlardan biri sabit noktalı modele geçiştir fakat LMS'in dayanıklılığı(Robustness) 8. bölümde yapılan testlerde görüleceği gibi bilgisayar ortamındaki sonuçlara yakındır. Diğer önemli bir faktör gerçek zaman uygulamasında istenilen kabullerin sağlanamamasıdır. LMS'in başarılı olması için birincil girişteki gürültü işareti ile referans girişteki gürültü işareti birbiri ile ilişkili olmalı, konuşma işaretini taşıyan bilgi işareti bunlardan bağımsız olmalıdır. Sesi örneklerken bu kabuller tam olarak sağlanamamaktadır. Ayrıca LMS'in diğer bir kabulu alınan işaretlerin sıfır ortalamalı olmasıdır. Sesin örneklenmesi katındaki donanımın istenmeyen özelliklerinden dolayı bu kabul de tam olarak sağlanamamaktadır. Bu durumlar performansta

ciddi kayıba yol açabilir. Hazırlanan VHDL kodunda bunlara yönelik önlem alınmaya çalışılmıştır ancak bu önlemler teorideki kabulleri tamamen sağlayamamıştır ama aşağıdaki şekil 9.1 deki frekans spektrumu incelenirse LMS'in bütün bu olumsuzluklara rağmen gürültüyü büyük ölçüde azalttığı gösterilmiştir.



Şekil 9.1 LMS süzgecinin gerçek zamanlı uygulama sonucu

Şekil 9.1'de ses işaretinin zaman düzleminde hangi frekans bileşenlerine sahip olduğu gösterilmiştir. Buradaki ses işareti gerçek zaman uygulama sonucudur. Uygulamada kullanılan gürültü beyaz gürültü, jet gürültüsü ve farklı frekans bileşenlerine sahip gürültülerdir. Gürültü zamanda değişirken LMS süzgeci performans eğrisinin minimumunu bulmaya ve takip etmeye çalışmaktadır. Süzgecin açık ve kapalı olduğu yerler LMS'in çalıştığı ve farkı göstermek için çalışmadığı süzgecin kapalı olduğu yerleri göstermektedir. Şekil 9.1, LMS'in uygulamada çalışarak gürültü bileşenlerini azalttığını göstermektedir.

Yüksek lisans tezimi seçerken dikkate aldığım konular bugüne kadar almış olduğum dersler, ilgi alanlarım ve tezimin sonuçlarının pratik hayatta kullanılabilecek bir çıktıya sahip olabilmesiydi. Çalışmamı tamamladığımda gördüğüm, kendinden uyarlamalı süzgeçler hakkında bilgi sahibi olduğum ve bu süzgeçleri gerçek hayata nasıl aktarılacağı hakkında deneyim sahibi olduğumdur. Ayrıca tezimin çıktısı istenilirse ürüne dönüştürülebilir olması benim açımdan mutluluk verici bir durumdur.

Ayrıca tezimin ağırlığı elde edilen algoritmanın sayısal ortamda gerçek zamanlı olarak uygulanmasıdır. Bu sayede teoride kalmayıp çalışmamı pratiğe taşımamın bana katkısının büyük olduğunu düşünmekteyim. Bu sayede uygulamada ki zorluklar buna karşı alınabilecek önlemler ve sonuç olarak bilgisayar ortamından gerçek zamanlı uygulamalara geçiş hakkında bilgi ve deneyimim olmuştur.

KAYNAKLAR

Haykin Simon, (1986) "Adaptive Filter Theory", Prentice-Hall, New Jersey

Haykin Simon, (1996 3rd Ed.) "Adaptive Filter Theory", Prentice-Hall, New Jersey

Mitra K. Sanjit, (1998) "Digital Signal Processing A Computer Based Approach", McGraw - Hall, New York.

Oppenheim V. Alan, Ronald W. Schafer, (1999 2th Ed.)"Discrete-Time Signal Processing", Prentice-Hall, New Jersey

Pedroni A. Volnei, (2004) "Circuit design with VHDL", MIT Press, Cambridge, Massachusetts

Perry L. Douglas, (2002 4th Ed.), "VHDL Programming by Example", McGraw-Hill, New York.

İNTERNET KAYNAKLARI

http://www.analog.com/en/prod/0,,AD7940,00.html

http://www.analog.com/en/prod/0%2C2877%2CAD5640%2C00.html

www.ieee.org

www.xilinx.com

http://cslu.ece.ogi.edu/nsel/data/SpEAR_technical.html

EKLER

Ek 1 LMS algoritmasına ait VHDL kodu

```
----------------------------------------------------------------------------
-- Engineer     :    Olcay Sevim
--
-- Creation Date:    07/12/2006
-- Design Name  :    lms core
-- Module Name  :    lms core
-- Target Device:    Spartan3 FPGA
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity lms_core is
  Generic(
    datawidth          : integer := 14;
    tap                : integer := 36
    );

  Port (
           clk           : in std_logic;
           rst_n         : in  std_logic ;
           primary       : in std_logic_vector(datawidth-1 downto 0);
           desired       : in std_logic_vector(datawidth-1 downto 0);
       state_vec     : in std_logic_vector(5 downto 0);
       output        : out std_logic_vector(datawidth-1 downto 0)
  );
end lms_core;

architecture RTL of lms_core is
    type buffer_in is array (tap-1 downto 0) of std_logic_vector(datawidth-1 downto 0);
     type weight_b is array (tap-1 downto 0) of std_logic_vector(14 downto 0);
     type acumm_type is array (11 downto 0) of std_logic_vector(17 downto 0);
     type mult_type is array (11 downto 0) of std_logic_vector(35 downto 0);

    signal wave         : buffer_in;
    signal weight       : weight_b;
    signal weight_s     : weight_b;
    signal accum        : mult_type;
    signal operanda     : acumm_type;
    signal operandb     : acumm_type;
    signal state        : integer range 0 to 63;
    signal error        : std_logic_vector(18 downto 0);
    signal desired_s    : std_logic_vector(34 downto 0);

    type sum_vec29 is array (5 downto 0) of std_logic_vector(29 downto 0);
    signal sum_29       : sum_vec29;
    type sum_vec30 is array (2 downto 0) of std_logic_vector(30 downto 0);
    signal sum_30       : sum_vec30;
    signal sum_30ss     : std_logic_vector(32 downto 0);
    signal sum_31       : std_logic_vector(31 downto 0);
    signal sum_32       : std_logic_vector(32 downto 0);
    signal sum_32_reg   : std_logic_vector(33 downto 0);
    signal sum_33       : std_logic_vector(33 downto 0);
    signal sum_34       : std_logic_vector(34 downto 0);
    signal error34      : std_logic_vector(34 downto 0);
    signal sum_32_ext   : std_logic_vector(34 downto 0);
    signal operanderror : std_logic_vector(17 downto 0);

    begin

     state                 <= conv_integer(state_vec);
     gen0: for i in 0 to 11 generate
         operanda(i)   <= wave(i)(13)&wave(i)(13)&wave(i)(13)&wave(i) when (state<2) else
                    wave(12+i)(13)&wave(12+i)(13)&wave(12+i)(13)&wave(12+i) when (state<3) else
                    wave(24+i)(13)&wave(24+i)(13)&wave(24+i)(13)&wave(24+i) when (state<4) else
                    wave(i)(13)&wave(i)(13)&wave(i)(13)&wave(i) when (state<13) else
                    wave(12+i)(13)&wave(12+i)(13)&wave(12+i)(13)&wave(12+i) when(state<14) else
                    wave(12+i)(13)&wave(12+i)(13)&wave(24+i)(13)&wave(24+i);


          operandb(i) <= weight(i)(14)&weight(i)(14)&weight(i)(14)&weight(i)when(state<2) else
             weight(12+i)(14)&weight(12+i)(14)&weight(12+i)(14)&weight(12+i)when(state<3) else
             weight(24+i)(14)&weight(24+i)(14)&weight(24+i)(14)&weight(24+i)when(state<4) else
             operanderror;
        end generate gen0;
        process(clk, rst_n)
            begin
                if (rst_n='0') then
                    wave(0)    <= (others=>'0');
                    wave(1)    <= (others=>'0');
                    wave(2)    <= (others=>'0');
                    wave(3)    <= (others=>'0');
                    wave(4)    <= (others=>'0');
                    wave(5)    <= (others=>'0');
                    wave(6)    <= (others=>'0');
                    wave(7)    <= (others=>'0');
                    wave(8)    <= (others=>'0');
                    wave(9)    <= (others=>'0');
                    wave(10)   <= (others=>'0');
                    wave(11)   <= (others=>'0');
                    wave(12)    <= (others=>'0');
                    wave(13)    <= (others=>'0');
                    wave(14)    <= (others=>'0');
                    wave(15)    <= (others=>'0');
                    wave(16)    <= (others=>'0');
```

```
                    wave(17)    <= (others=>'0');
                    wave(18)    <= (others=>'0');
                    wave(19)    <= (others=>'0');
                    wave(20)    <= (others=>'0');
                    wave(21)    <= (others=>'0');
                    wave(22)   <= (others=>'0');
                    wave(23)   <= (others=>'0');
                    wave(24)    <= (others=>'0');
                    wave(25)    <= (others=>'0');
                    wave(26)    <= (others=>'0');
                    wave(27)    <= (others=>'0');
                    wave(28)    <= (others=>'0');
                    wave(29)    <= (others=>'0');
                    wave(30)    <= (others=>'0');
                    wave(31)    <= (others=>'0');
                    wave(32)    <= (others=>'0');
                    wave(33)    <= (others=>'0');
                    wave(34)   <= (others=>'0');
                    wave(35)   <= (others=>'0');
                   desired_s  <= (others=>'0');
                elsif (clk='1' and clk'event) then
                    if (state=0) then
                        wave(0) <= primary;
                        wave(tap-1 downto 1)<= wave(tap-2 downto 0);
                        if (desired(datawidth-1)='0') then
                            desired_s<= desired(datawidth-1)& desired(datawidth-1)&desired(datawidth-1)&desired(datawidth-1)
&desired(datawidth-1)&desired(datawidth-1)&desired(datawidth-1)&desired(datawidth-1)&desired(datawidth-1)&desired(datawidth-
1) &desired&"0000000000";
                        else
                            desired_s<= desired(datawidth-1)&desired(datawidth-1)&desired(datawidth-1)&desired(datawidth-1)
&desired(datawidth-1)&desired(datawidth-1)&desired(datawidth-1)&desired(datawidth-1)&desired(datawidth-1)&desired(datawidth-
1) &desired&"0000000000";
                        end if;
                    end if;
                end if;
            end process;
        gen1: for i in 0 to 11 generate
            process(clk, rst_n)
            begin
                if (rst_n='0') then
                    accum(i)         <= (others=>'0');
                elsif (clk='1' and clk'event) then
                    if (state=1) then
                        accum(i)      <= signed(operanda(i))*signed(operandb(i));
                    elsif (state=2) then
                        accum(i)      <= signed(operanda(i))*signed(operandb(+i));
                    elsif (state=3) then
                        accum(i)      <= signed(operanda(i))*signed(operandb(i));
                    elsif (state=12) then
                        accum(i)      <= signed(operanda(i))*signed(operandb(i));
                    elsif (state=13) then
                        accum(i)      <= signed(operanda(i))*signed(operandb(i));
                    elsif (state=14) then
                        accum(i)      <= signed(operanda(i))*signed(operandb(i));
                    end if;
                end if;
            end process;
         end generate gen1;
         gen2: for i in 0 to 5 generate
            process(clk, rst_n)
            begin
                if (rst_n='0') then
                    sum_29(i)         <= (others=>'0');
                elsif (clk='1' and clk'event) then
                    if (state=2) then
                        sum_29(i)     <= signed(accum(i*2)(29 downto 0))+signed(accum(i*2+1)(29 downto 0));
                    elsif (state=3) then
                        sum_29(i)     <= signed(accum(i*2)(29 downto 0))+signed(accum(i*2+1)(29 downto 0));
                    elsif (state=4) then
                        sum_29(i)     <= signed(accum(i*2)(29 downto 0))+signed(accum(i*2+1)(29 downto 0));
                    end if;
                end if;
            end process;
          end generate gen2;
          gen3: for i in 0 to 2 generate
            process(clk, rst_n)
            begin
                if (rst_n='0') then
                    sum_30(i)        <= (others=>'0');
                elsif (clk='1' and clk'event) then
                    if (state=3) then
                        sum_30(i)       <= signed(sum_29(i*2)(29)&sum_29(i*2))+signed(sum_29(i*2+1)(29)&sum_29(i*2+1));
                    elsif (state=4) then
                        sum_30(i)       <= signed(sum_29(i*2)(29)&sum_29(i*2))+signed(sum_29(i*2+1)(29)&sum_29(i*2+1));
                    elsif (state=5) then
                        sum_30(i)       <= signed(sum_29(i*2)(29)&sum_29(i*2))+signed(sum_29(i*2+1)(29)&sum_29(i*2+1));
                    end if;
                end if;
            end process;
         end generate gen3;
         --sum_30ss                    <= sum_30(2)(29)&sum_30(2)(29)&sum_30(2);--debug
         sum_32_ext                   <= sum_32(32)&sum_32(32)&sum_32;
            process(clk, rst_n)
              begin
                if (rst_n='0') then
                    sum_31           <= (others=>'0');
                    sum_32           <= (others=>'0');
                    error34          <= (others=>'0');
                    error            <= (others=>'0');
                    output           <= (others=>'0');
                    sum_30ss         <= (others=>'0');
                elsif (clk='1' and clk'event) then
                    sum_30ss                    <= sum_30(2)(30)&sum_30(2)(30)&sum_30(2);
                    if (state=4) then
```

```
                sum_31       <= signed(sum_30(0)(30)&sum_30(0))+signed(sum_30(1)(30)&sum_30(1));
            elsif (state=5) then
                sum_32       <= signed(sum_31(31)&sum_31)+signed(sum_30ss);
                sum_31       <= signed(sum_30(0)(30)&sum_30(0))+signed(sum_30(1)(30)&sum_30(1));
            elsif (state=6) then
                sum_32       <= signed(sum_31(31)&sum_31)+signed(sum_30ss);
                sum_31       <= signed(sum_30(0)(30)&sum_30(0))+signed(sum_30(1)(30)&sum_30(1));
                sum_32_reg   <= sum_32(32)&sum_32;
            elsif (state=7) then
                sum_32       <= signed(sum_31(31)&sum_31)+signed(sum_30ss);
                sum_33       <= signed(sum_32_reg)+signed(sum_32(32)&sum_32);
            elsif (state=8) then
                sum_34       <= signed((sum_33(33)&sum_33))+signed(sum_32_ext);
            elsif(state=9) then
                error34      <= signed(desired_s)-signed(sum_34);
            elsif(state=10) then
                if (error34(34)='0') then
                    if (error34(33 downto 28)>"000000") then
                        error        <= "01111111111111111111";
                    elsif(error34(9)='0') then
                        error        <='0'&error34(27 downto 10);
                    elsif (error34(27 downto 10)="11111111111111111") then
                       error        <='0'&error34(27 downto 10);
                    else
                        error        <='0'&(error34(27 downto 10)+"01");
                    end if;
                elsif (error34(33 downto 28)<"111111") then
                    error        <= "1000000000000000000";
                --elsif(error34(9)='1') then--debug
                elsif(error34(9)='0') then
                    error        <='1'&error34(27 downto 10);
                elsif (error34(27 downto 10)="00000000000000000") then
                    error        <='1'&error34(27 downto 10);
                else
                    error        <='1'&(error34(26 downto 10)-"01");
                end if;
                if (error34(34)='0') then
                    if(error34(33 downto 24)>"0000000000") then
                       output  <= "011111111111111";
                    elsif (error34(9)='0') then
                       output  <= '0'&error34(23 downto 10);
                    elsif (error34(23 downto 10)="1111111111111") then
                       output  <= '0'&error34(23 downto 10);
                    else
                       output  <= '0'&error34(23 downto 10)+"01";
                    end if;
                elsif (error34(33 downto 24)<"1111111111") then
                       output  <= "100000000000000";
                --elsif (error34(9)='1') then--debug
                elsif (error34(9)='0') then
                       output  <= '1'&error34(23 downto 10);
                elsif  (error34(23 downto 10)="00000000000000") then
                       output  <= '1'&error34(23 downto 10);
                else
                       output  <= '1'&error34(22 downto 10)-"01
                end if;
            elsif(state=11) then
                     if (error(18)='0') then
                        if (error(7)='0') then
                        operanderror       <= error(18)&error(18)&error(18)&error(18)&error(18)&error(18)&error(18)&
                                              error(18 downto 8);
                        elsif (error(17 downto 8)="1111111111") then
                        operanderror       <= error(18)&error(18)&error(18)&error(18)&error(18)&error(18)&error(18)&
                                              error(18 downto 8);
                        else
                        operanderror       <= error(18)&error(18)&error(18)&error(18)&error(18)&error(18)&error(18)&
                                              (error(18 downto 8)+"01");
                        end if;
                    elsif (error(7)='1') then
                    operanderror       <= error(18)&error(18)&error(18)&error(18)&error(18)&error(18)&error(18)&
                                          error(18 downto 8);
                    elsif (error(16 downto 8)="000000000") then
                         operanderror       <= error(18)&error(18)&error(18)&error(18)&error(18)&error(18)&error(18)
                                               &error(18 downto 8);
                    else
                    operanderror       <= error(17)&error(17)&error(17)&error(17)&error(17)&error(17)&error(17)&
                                          error(17)&(error(17 downto 8)-"01);
                    end if;
            end if;
        end if;
     end process;
gen4: for i in 0 to 11 generate
   process(clk, rst_n)
   begin
       if (rst_n='0') then
           weight_s(i)                 <= (others=>'0');
           weight(i)                   <= (others=>'0');
           weight_s(12+i)                 <= (others=>'0');
           weight(12+i)                   <= (others=>'0');
           weight_s(24+i)                 <= (others=>'0');
           weight(24+i)                   <= (others=>'0');
       elsif (clk='1' and clk'event) then
           if (state=13) then
               weight_s(i)      <= accum(i)(26)&accum(i)(26)&accum(i)(26)&accum(i)(26)&accum(i)(26 downto
                                   16)+accum(i)(15);
           elsif (state=14) then
               weight_s(12+i)   <= accum(i)(26)&accum(i)(26)&accum(i)(26)&accum(i)(26)&accum(i)(26 downto
                                   16)+accum(i)(15);
               weight(i)               <= signed(weight_s(i))+signed(weight(i));
           elsif (state=15) then
               weight_s(24+i)   <= accum(i)(26)&accum(i)(26)&accum(i)(26)&accum(i)(26)&accum(i)(26 downto
                                   16)+accum(i)(15);
               weight(12+i)               <= signed(weight_s(12+i))+signed(weight(12+i));
```

```
                elsif (state=16) then
                    weight(24+i)                <= signed(weight_s(24+i))+signed(weight(24+i));
                end if;
            end if;
        end process;
    end generate gen4;

end RTL;
```

Ek 2 LMS Algoritmasına Ait Sentezlenen VHDL Kodunun Place&Route Raporu

```
Release 8.1.03i par I.27
Copyright (c) 1995-2005 Xilinx, Inc.  All rights reserved.

URNO::  Tue Feb 13 11:58:13 2007

par -w -intstyle ise -pl high -rl high -xe n -t 1 lms_top_v2_map.ncd
lms_top_v2.ncd lms_top_v2.pcf

Constraints file: lms_top_v2.pcf.
WARNING:Par:331 - You are using an evaluation version of Xilinx Software.
In 50 days, this program will not operate. For
   more information about this product, please refer to the Evaluation
Agreement, which was shipped to you along with
   the Evaluation CDs.
   To purchase an annual license for this software, please contact your
local Field Applications Engineer (FAE) or
   salesperson. If you have any questions, or if we can assist in any way,
please send an email to: eval@xilinx.com
   Thank You!
Loading device for application Rf_Device from file '3s200.nph' in
environment C:\Xilinx.
   "lms_top_v2" is an NCD, version 3.1, device xc3s200, package ft256,
speed -4

Initializing temperature to 85.000 Celsius. (default - Range: 0.000 to
85.000 Celsius)
Initializing voltage to 1.140 Volts. (default - Range: 1.140 to 1.260
Volts)

Device speed data version:  "PRODUCTION 1.37 2005-11-04".

INFO:Par:253 - The Map -timing placement will be retained since it is
```

likely to achieve better performance.

Device Utilization Summary:

Number of BUFGMUXs	2 out of 8	25%
Number of External IOBs	10 out of 173	5%
Number of LOCed IOBs	10 out of 10	100%
Number of MULT18X18s	12 out of 12	100%
Number of Slices	1747 out of 1920	90%
Number of SLICEMs	0 out of 960	0%

Overall effort level (-ol): Not applicable because -pl and -rl switches are used

Router effort level (-rl): High

Starting initial Timing Analysis. REAL time: 14 secs

Finished initial Timing Analysis. REAL time: 14 secs

Total REAL time to Router completion: 1 mins 14 secs

Total CPU time to Router completion: 1 mins 8 secs

Generating "PAR" statistics.

****

Generating Clock Report

****

Clock Net	Resource	Locked	Fanout	Net Skew(ns)	Max Delay(ns)
clk_125	BUFGMUX5	No	1704	0.041	1.052
clk_BUFGP	BUFGMUX0	No	6	0.039	1.050

* Net Skew is the difference between the minimum and maximum routing only delays for the net. Note this is different from Clock Skew which is reported in TRCE timing report. Clock Skew is the difference between the minimum and maximum path delays which includes logic delays.

The Delay Summary Report

The NUMBER OF SIGNALS NOT COMPLETELY ROUTED for this design is: 0

The AVERAGE CONNECTION DELAY for this design is: 1.629
The MAXIMUM PIN DELAY IS: 6.765
The AVERAGE CONNECTION DELAY on the 10 WORST NETS is: 5.679

Listing Pin Delays by value: (nsec)

d < 1.00	< d < 2.00	< d < 3.00	< d < 4.00	< d < 7.00	d >= 7.00
4264	5629	4248	983	243	0

Timing Score: 0

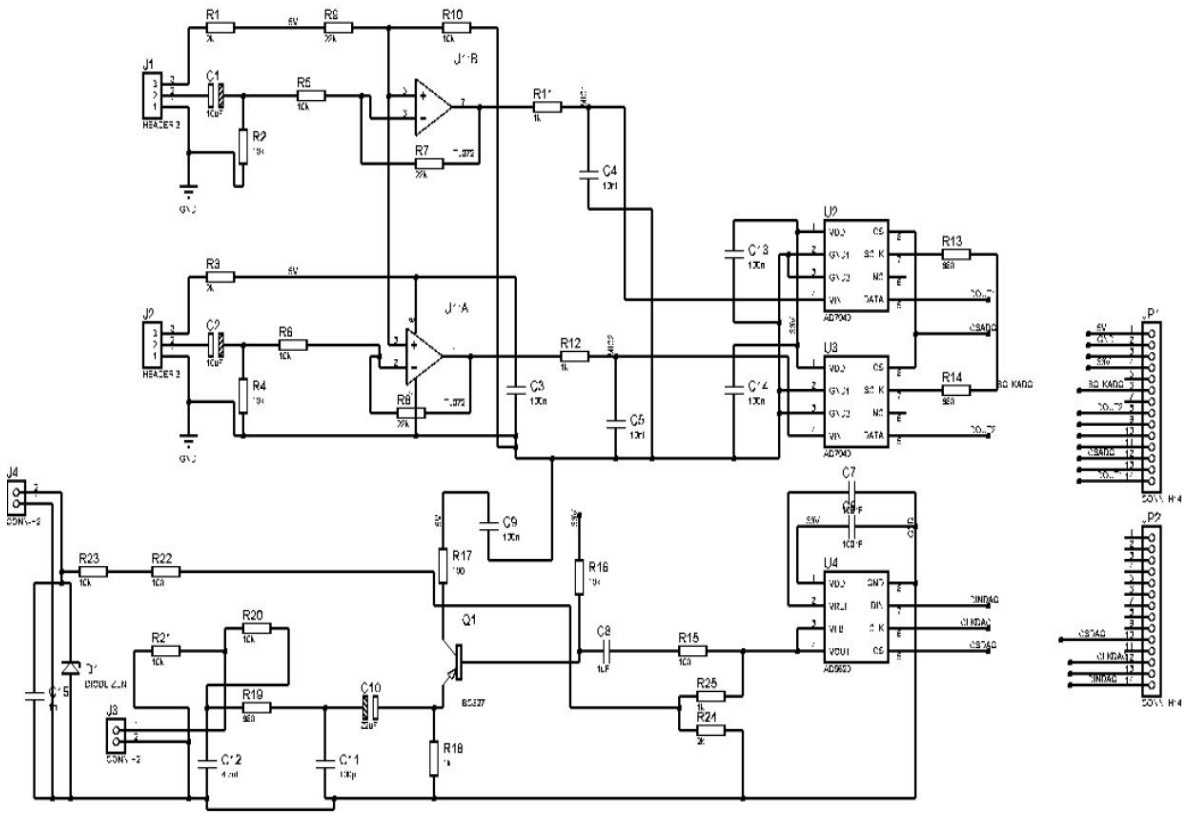
Asterisk (*) preceding a constraint indicates it was not met.
This may be due to a setup or hold violation.

Constraint	Requested	Actual
TS_clk_div_5_ = PERIOD TIMEGRP "clk_div<5	40.000ns	24.595ns
NET "clk_BUFGP/IBUFG" PERIOD = 25 ns HIGH	25.000ns	4.288ns

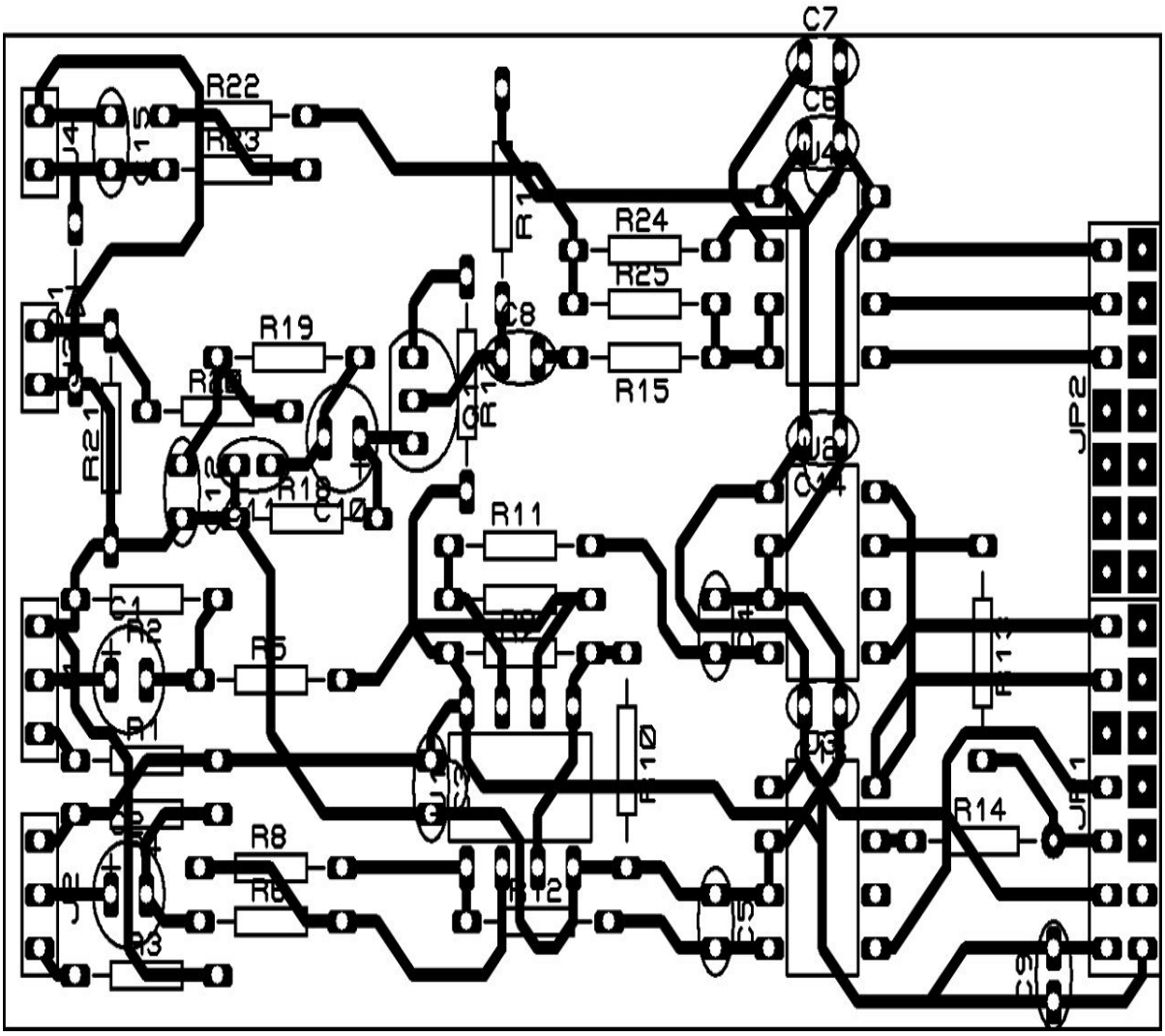
All constraints were met.

```
Generating Pad Report.

All signals are completely routed.

Total REAL time to PAR completion: 1 mins 19 secs
Total CPU time to PAR completion: 1 mins 13 secs

Peak Memory Usage:  175 MB

Placer: Placement generated during map.
Routing: Completed - No errors found.
Timing: Completed - No errors found.

Number of error messages: 0
Number of warning messages: 1
Number of info messages: 1

Writing design to file lms_top_v2.ncd



PAR done!
```

Ek 3 Veri Alış Verişinde Kullanılan Devre Şeması



Ek 4 Veri Alış Verişinde Kullanılan Baskı Devre



ÖZGEÇMİŞ

Doğum tarihi	14.02.1980	
Doğum yeri	İstanbul	
Lise	1994-1997	Bayrampaşa Tuna Lisesi
Lisans	1998-2003	ITU Elektrik-Elektronik Fak. Elektronik ve Hab. Mühendisliği Bölümü

Çalıştığı kurum(lar)

	2005-2007	Vestek Elektronik Ar-GE A.Ş.