YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

GÖRSEL ARABİRİMLİ SAYISAL SES İŞLEYİCİSİ TASARIMI UYGULAMALARI

Arş. Gör. Osman Nuri ERTÜRK

F.B.E. Elektronik ve Haberleşme Mühendisliği Anabilim Dalı Haberleşme Programında Hazırlanan

YÜKSEK LİSANS TEZİ

Tez Danışmanı: Yrd. Doç. Dr. Soner ÖZGÜNEL

Yrd. Doç. Dr. Soner ÖZGÜNEL

Prof. Yahya KARSLIĞIL

İSTANBUL, 1998

Doç. Dr. Ertuğrul ERİŞ

İÇİNDEKİLER

Sayfa

KISALTMA LİSTESİ

AFD	Ayrık Fourier Dönüşümü
AGS	Alçak Geçiren Süzgeç
BPS	Basit Parametrik Süzgeç
ÇÇSG	Çok Çıkışlı Sayısal Geciktirici
DFO	Düşük Frekans Osilatörü
GEÜ	Geciktirme Etkisi Üreteci
GSİ	Görsel Ses İşleyicisi
HFD	Hızlı Fourier Dönüşümü
KOD	Kareköksel Ortalama Değer
KSFD	Kısa Süreli Fourier Dönüşümü
MS	Müzikal Süzgeç
SDY	Sonlu Dürtü Yanıtlı
TÇY	Tümleşik Çınlatıcı Yapı
YGS	Yüksek Geçiren Süzgeç

ŞEKİL LİSTESİ

ÇİZELGE LİSTESİ

ÖNSÖZ

Başta Yrd. Doç. Dr. Soner Özgünel, Arş. Gör. Bülent Bolat, Arş. Gör. Cumhur Erkut, Arş. Gör. Güner Arslan, Arş. Gör. Murat Aytekin ve Elektr. Müh. Ahmet Akın olmak üzere, çalışmada emeği geçen herkese teşekkür ederim.

ÖZET

Ses etkileri oluşturma, süzgeçleme ve spektral analiz işlemleri, bilgisayar tabanlı müziğin temel konu başlıkları arasında yer almaktadır. Ses etkileri daha çok, işaret üzerinde müzikal bir değişim yaratmak veya doğal bir olayı yapay olarak gerçeklemek amacıyla tasarlanmaktadır. Süzgeçleme işleminde ise, işaretin belirli bir aralıktaki frekans bileşenlerinin güçlendirilmesi ya da zayıflatılması yoluyla ses kalitesinin yükseltilmesine çalışılmaktadır. Bilgisayar tabanlı müzik uygulamalarında, özel elektronik düzenekler veya kişisel bilgisayar ortamında çalışan yazılımlar kullanılmaktadır. Sunulan bu tez çalışmasında, Microsoft Windows ortamında çalışan, görsel arabirimli bir ses işleyicisi yazılımının tasarlanması amaçlanmıştır. Yazılım yoluyla sayısal bir işaretin spektral analizi yapılabilmekte, farklı türde ses etkileri ve süzgeçleme uygulamaları gerçekleştirilebilmektedir. Tasarım aşamasında, C++ programlama dili kullanılmıştır. Yazılıma ait her bir işlev, tümleşik sayısal yapılar aracılığıyla gerçekleştirilmektedir. Tümleşik yapılar, nesne yönelimli programlama tekniği yoluyla, bağımsız sayısal birimlerin uygun biçimlerde bir araya getirilmesiyle oluşturulmuştur. Yapılan denemeler sonucunda, yazılımın bazı özel durumlar dışında sorunsuz biçimde çalıştığı saptanmıştır. Herhangi bir kişisel bilgisayar kullanıcısı, gerçek-zaman uygulamaları dışında, yüksek maliyetli özel bir elektronik cihaza gereksinim duymadan, farklı türde sayısal ses işlemlerini tasarlanan yazılım aracılığıyla gerçekleştirebilmektedir.

ABSTRACT

Sound effects, filtering, and spectral analysis are some of the main subjects of computer music. Sound effects are generally designed to add a musical meaning to a signal or simulate a natural phenomenon. As for filtering, it is mostly used to increase the quality of a digital sound signal by making changes on some particular frequencies in its spectrum. In general, computer music applications are realised with the help of special electronic devices or PC based software programs. In this work, it was aimed to design a visual wave editor program running in Microsoft Windows environment. The editor was written in C++ programming language. Any functions belonging to the editor are performed by integrated software structures. The structures were built of a number of independent digital units which were combined in a suitable way by using object oriented programming style. After lots of tests on different kind of digital sound signals, it was seen that the program worked well excluding some special worst cases. In conclusion, it is possible for any PC user to realise different types of digital sound processes except real-time applications through the designed editor instead of expensive electronic devices.

1. GİRİŞ

Sunulan bu tez çalışmasında, sayısal ses işaretleri üzerinde farklı türde işlemler gerçekleştirebilen, görsel arabirimli bir ses işleyicisi (GSİ) yazılımının tasarlanması amaçlanmıştır. GSİ yazılımı, Microsoft Windows ortamında, kullanıcı etkileşimli bir biçimde çalışmaktadır. Tasarım aşamasında, C++ programlama dili, nesne yönelimli programlama tekniği kalıpları göz önünde bulundurularak kullanılmıştır. GSİ yazılımı aracılığıyla, sayısal bir ses işareti üzerinde farklı özellikte ses etkileri oluşturulabilmekte ve değişik karakteristikli süzgeçleme işlemleri yapılabilmektedir. Yazılım ayrıca, sayısal ses işaretini zaman ve frekans düzlemlerinde görüntüleyebilmekte ve işarete ait frekans bileşenlerinin zamanda değişimini sergileyebilmektedir.

İkinci bölümde, sayısal süzgeç yapıları ve farklı türde ses etkilerine ilişkin temel bilgiler verilmiştir. Yapılan tanımlamalar ve incelenen sayısal yapılar, bilgisayar tabanlı müzik uygulamalarına yönelik bir bakış açısıyla ele alınmıştır (Roads, 1996). Ses etkileri ve süzgeçleme işlemleri yoluyla işaret üzerinde yaratılan değişimlerin nitelikleri bu bölümde açıklanmıştır. Ses etkileri, geciktirme etkileri, çınlama ve dinamik değer işlemleri olarak adlandırılan üç ana başlık altında incelenmiştir. Her bir etkinin kullanım nedeni ve gerçekleştirilme yöntemleri üzerinde ana hatlarıyla durulmuştur.

Üçüncü bölüm, spektral analiz yöntemlerinin incelenmesine ayrılmıştır. Spektral analiz yöntemleri, pencereleme, HFD (Hızlı Fourier Dönüşümü) ve KSFD (Kısa Süreli Fourier Dönüşümü) ana başlıkları altında incelenmiştir. GSİ yazılımında, bu yöntemlerden yararlanılarak sayısal ses işaretinin spektral analizi yapılmakta ve işarete ait frekans bileşenlerinin zamana bağlı değişimi görüntülenmektedir.

Dördüncü bölümde, tasarlanan GSİ yazılımının temel işlevleri ve tasarımda kullanılan yöntemler ele alınmıştır. Yazılıma ait her bir işlev, tümleşik sayısal yapılar aracılığıyla gerçekleştirilmektedir. Tümleşik yapılar, nesne yönelimli programlama tekniği yoluyla, bağımsız sayısal birimlerin uygun biçimlerde bir araya getirilmesiyle oluşturulmuştur.

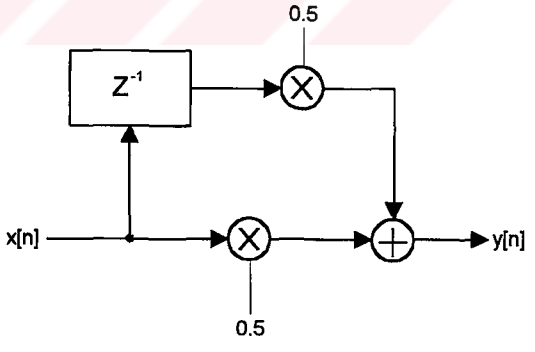
2. TEMEL KAVRAMLAR

2.1 Sayısal Süzgeçler

Girişine gelen sayısal bir işareti belirli bir dönüşümden geçirerek çıkışına farklı bir sayısal işaret olarak veren işlemsel süreçler veya algoritmalar sayısal süzgeç olarak adlandırılmaktadır (Rabiner vd., 1972). Bu tanımlama uyarınca, bir giriş ve çıkışa sahip herhangi bir sayısal yapının sayısal bir süzgeç olarak değerlendirilmesi olasıdır. Sayısal süzgeç terimi daha çok, bir ses işareti spektrumundaki belirli frekans bileşenlerini bastıran veya güçlendiren yapıları tanımlamakta kullanılmaktadır. Bununla birlikte, kimi sayısal süzgeç uygulamaları, giriş işaretinin spektral özelliklerine ek olarak zamansal özellikleri üzerinde de değişimler oluşturmak amacıyla gerçeklenmektedir.

2.1.1 Basit alçak geçiren süzgeç

Basit bir alçak geçiren süzgeç (AGS), girişinde o anda bulunan örnek ile bir önceki örneğin aritmetik ortalamasını çıkış olarak vermektedir. Bu tür bir AGS yardımıyla, giriş işaretindeki yüksek frekans bileşenlerinin yol açabileceği ani değişimlerin etkileri yumuşatılabilmektedir.

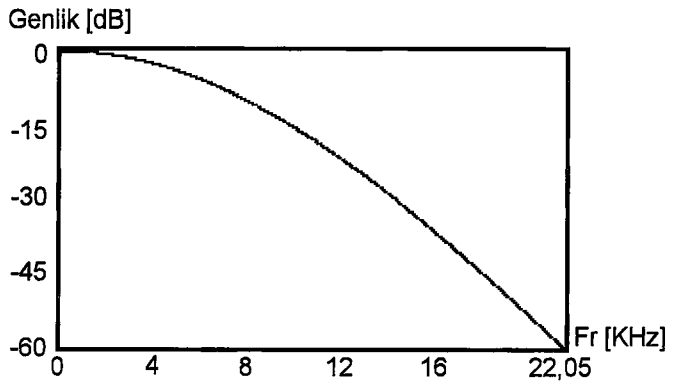


Şekil 2.1 Basit AGS yapısı.

Şekil 2.1'de, basit bir AGS yapısı gösterilmiştir. Süzgeç denklemi,

$$y[n] = 0{,}5x[n] + 0{,}5x[n-1] \tag{2.1}$$

eşitliği ile verilmektedir. $y[n]$ ve $x[n]$ terimleri sırasıyla, süzgecin o anki çıkışına ve girişine ait örnek değerlerini, $x[n-1]$ terimi ise süzgecin bir örnek önceki (bir örnek geciktirilmiş) giriş değerini ifade etmektedir. Eşitlikte yer alan 0,5 değerli çarpanlar süzgeç katsayıları olarak adlandırılmaktadır.

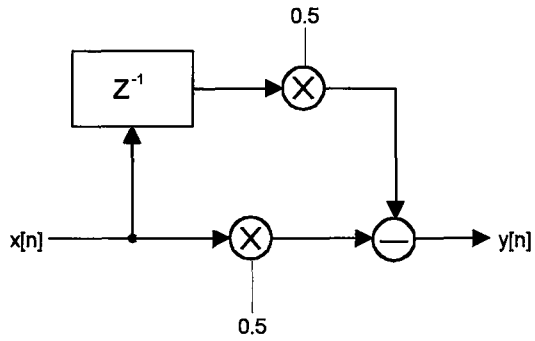


Şekil 2.2 Basit AGS frekans yanıtı.

Şekil 2.2'de, 44100 Hz örnekleme hızı için basit bir alçak geçiren süzgece ait frekans yanıtı gösterilmiştir. Şekilde görüldüğü gibi, AGS yüksek frekans bileşenleri üzerinde bastırıcı bir etki oluşturmaktadır. Süzgeç tasarımı, çıkış işareti örnek değeri, girişe ait ikiden fazla örnek değerinin ortalamasından elde edilecek şekilde de gerçekleştirilebilmektedir. Süzgecin derecesi arttırıldıkça giriş işaretinin yüksek frekans bileşenleri daha yüksek oranlarda bastırılarak süzgeç çıkışına iletilmektedir.

2.1.2 Basit yüksek geçiren süzgeç

Basit bir yüksek geçiren süzgeç (YGS), alçak geçiren süzgeçten farklı olarak, girişten o an gelen örnek değeri ile bir önceki örnek değerinin farkının ortalamasını çıkışına vermektedir. Bir başka deyişle, basit bir YGS, girişindeki ardışıl örnekler arasındaki farkı hesaplayarak çıkışına iletmektedir. Bu nedenle YGS çıkışında, örnekler arasındaki farkın küçük değerler aldığı alçak frekans bileşenleri bastırılırken, örnekler arası farkın büyük değerler aldığı yüksek frekans bileşenlerinin etkisi güçlendirilmektedir. Şekil 2.3'de, basit bir YGS yapısı gösterilmiştir.



Şekil 2.3 Basit YGS yapısı.

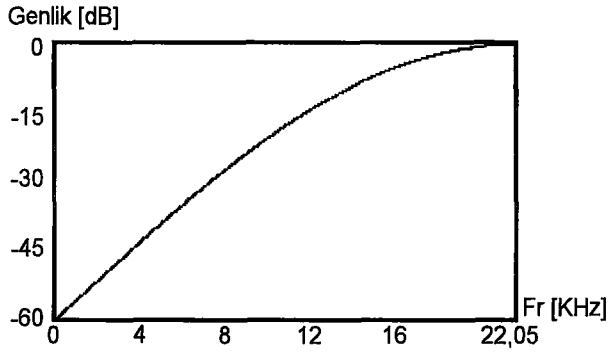
Basit YGS denklemi,

$$y[n] = 0{,}5x[n] - 0{,}5x[n-1] \tag{2.2}$$

eşitliği ile ifade edilmektedir. Eşitlikte yer alan 0,5 değerli çarpanların yerine değişken katsayılar kullanıldığında,

$$y[n] = a_0 x[n] - a_1 x[n-1] \tag{2.3}$$

bağıntısı ile verilen daha esnek bir süzgeç denklemi elde edilmektedir. a_0 ile, o anki örnek değerine ilişkin süzgeç katsayısı, a_1 ile ise bir geciktirmeli örnek değerine ilişkin süzgeç katsayısı ifade edilmektedir. Süzgeç katsayılarının değerlerinin değiştirilmesi farklı karakteristikte frekans yanıtlarının oluşmasına yol açmaktadır.



Şekil 2.4 Basit YGS frekans yanıtı.

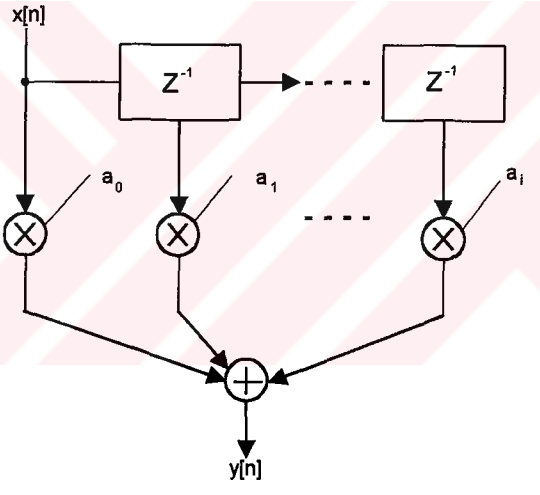
Şekil 2.4'de, 44100 Hz örnekleme hızı için, (2.2) eşitliği ile tanımlanan basit yüksek geçiren süzgece ait frekans yanıtı gösterilmiştir.

2.1.3 Genel SDY süzgeç yapısı

Genel bir SDY (sonlu dürtü yanıtlı) süzgeç denklemi,

$$y[n] = a_0 x[n] \pm a_1 x[n-1] \pm \ldots \pm a_i x[n-i] \tag{2.4}$$

eşitliği ile ifade edilmektedir. Eşitlikte yer alan a_i süzgeç katsayılarının pozitif değerlerden seçilmesi yoluyla AGS, negatif değerlerden seçilmesi yoluyla ise YGS yapıları elde edilmektedir.



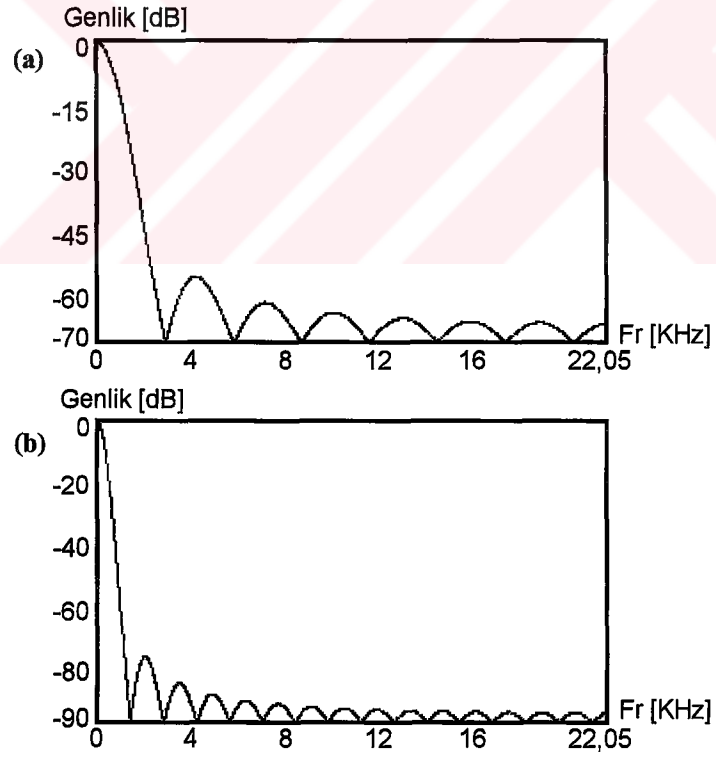
Şekil 2.5 Genel SDY süzgeç yapısı.

Genel SDY süzgeç yapısı ile i uzunluklu bir sayısal geciktirici arasında benzerlikler bulunmaktadır. i uzunluklu geciktirici, girişindeki işareti i sayıda örnek kadar geciktirerek çıkışına veren çevrimsel bir bellek birimidir. Bu tür bir birimin *"belleği"*, ancak i sayıda örnek değeriyle belirlenen geciktirici uzunluğuna denk düşen sonlu bir zaman aralığı kadar geriye uzanabilmektedir. Benzer şekilde, genel SDY süzgeç yapısının, dürtü işareti gibi kısa süreli bir giriş işaretine yanıtı da, sonlu bir zaman aralığını takiben sönümlenen bir işaret

olmaktadır. Bu nedenden dolayı, (2.4)'de bağıntısı verilen türden süzgeçler sonlu dürtü yanıtlı veya enine süzgeçler olarak adlandırılmaktadır.

Şekil 2.5'de i. dereceden genel SDY süzgeç yapısı gösterilmiştir. Yapıda i adet bir örneklik geciktirici bulunmaktadır. Geciktirici çıkışlarının her biri ilgili süzgeç katsayısı ile çarpılmaktadır. Süzgeç çıkışı, bu geciktirilmiş ve ilgili katsayı ile çarpılmış örneklerin toplamından elde edilmektedir.

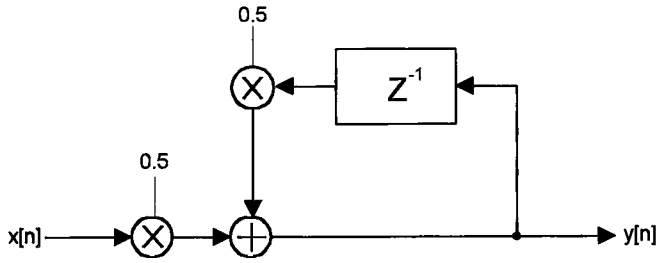
Süzgeç uzunluğu arttırıldıkça süzgecin iletim bandı daralmaktadır. Şekil 2.6'da, 44100 Hz örnekleme hızı için, genel SDY süzgece ait frekans yanıtını gösterilmiştir. Şekilde görüldüğü gibi, süzgecin derecesini arttırmak daha keskin bir eğimle kesime gidilmesini sağlamaktadır. Bununla birlikte, süzgeç sıfırlarının sayısındaki artış yan kulakçıklarda bir sıklaşmaya da yol açmaktadır. Daha yüksek dereceli süzgeçlerin daha büyük bir işlem hacmi ile gerçeklenebilir olduğunun da, uygulamalarda hesaba katılması gerekmektedir.



Şekil 2.6 SDY süzgeç frekans yanıtları: (a) 15 dereceli SDY süzgeç frekans yanıtı; (b) 31 dereceli SDY süzgeç frekans yanıtı.

2.1.4 Basit yinelemeli süzgeç

Geri besleme veya yineleme olarak adlandırılan işlem yoluyla, girişinde önceki çıkışlarına ait örnek değerlerini kullanan süzgeçler, sonsuz dürtü yanıtlı veya yinelemeli süzgeç olarak adlandırılmaktadır. Çıkışı girişine yönlendirilen yinelemeli bir süzgeç ile, basit bir SDY süzgece göre, işaretin önceki örnek değerlerinin çok daha fazlası, çok daha az süzgeç katsayısı, yani daha basit bir süzgeç yapısı kullanılarak girişe aktarılabilmektedir.



Şekil 2.7 Basit yinelemeli süzgeç yapısı.

Şekil 2.7'de, basit bir yinelemeli AGS yapısı gösterilmiştir. Süzgeç denklemi,

$$y[n] = 0{,}5x[n] + 0{,}5y[n-1] \tag{2.5}$$

eşitliği ile verilmektedir. Basit yinelemeli AGS çıkışı, o anki giriş ile bir önceki çıkışa ait örnek değerlerinin toplamının aritmetik ortalaması alınarak elde edilmektedir. Süzgecin SDY eşdeğeri,

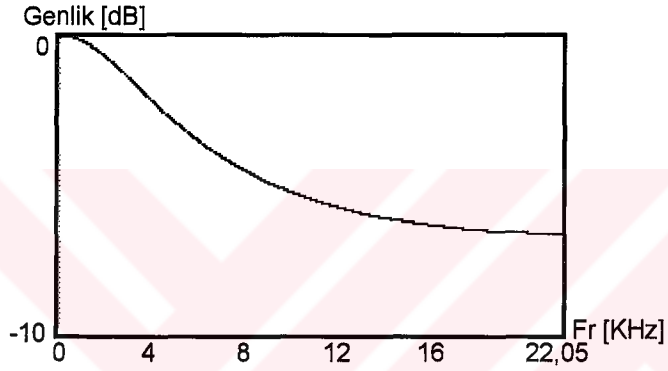
$$y[n] = \frac{1}{2}x[n] + \frac{1}{4}x[n-1] + \frac{1}{8}x[n-2] + \ldots \tag{2.6}$$

eşitliği ile belirtildiği gibi, sonsuz uzunluktadır. Şekil 2.8'de, 44100 Hz örnekleme hızı için, basit yinelemeli AGS'in frekans yanıtı gösterilmiştir.

(2.5)'de yer alan 0,5 değerli çarpanların yerine değişken katsayılar kullanıldığında,

$$y[n] = a.x[n] + b.y[n-1] \tag{2.7}$$

bağıntısı ile verilen daha esnek bir yinelemeli süzgeç denklemi elde edilmektedir. (2.7)'deki b katsayısı, geri besleme yolunu, yani çıkıştan girişe iletilen değeri etkilemektedir. b katsayısı arttırıldıkça süzgecin kesim frekansı giderek daha düşük değerler almaktadır. Süzgeç çıkışının kararlı durumda kalması için b katsayısının $b<1$ olacak şekilde seçilmesi gerekmektedir. Aksi takdirde, süzgeç kararsız duruma geçerek $y[n]$ çıkışının giderek artan değerler alması nedeniyle çıkışta sayısal bir taşma oluşmasına ve gürültülü bir işaretin elde edilmesine yol açmaktadır.



Şekil 2.8 Basit yinelemeli AGS frekans yanıtı.

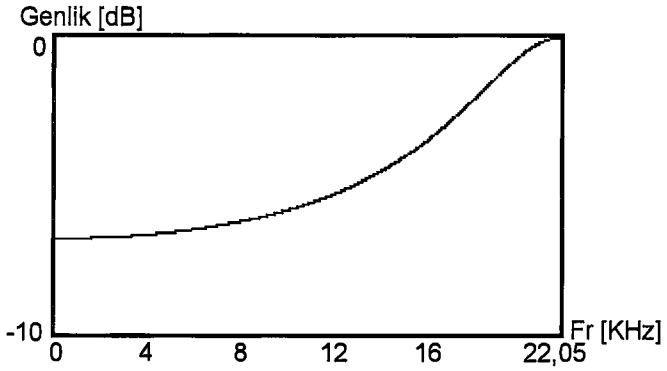
Basit bir yinelemeli YGS,

$$y[n] = 0{,}5x[n] - 0{,}5y[n-1] \tag{2.8}$$

bağıntısı ile ifade edilmektedir. Süzgeç çıkışı o anki giriş ile bir önceki çıkışa ait örnek değerlerinin farkının aritmetik ortalaması alınarak elde edilmektedir. Süzgeç bağıntısı daha genel bir biçimde,

$$y[n] = a.x[n] - b.y[n-1] \tag{2.9}$$

eşitliği ile verilmektedir. *b* katsayısının arttırılması süzgecin kesim frekansının daha yüksek değerler almasına neden olmaktadır. Bu durum, süzgecin, alçak frekans bileşenleri üzerindeki bastırıcı etkisini güçlendirmektedir. Şekil 2.9'da, 44100 Hz örnekleme hızı için, yinelemeli YGS'nin frekans yanıtı gösterilmiştir.



Şekil 2.9 Basit yinelemeli YGS frekans yanıtı.

2.1.5 Genel yinelemeli süzgeç yapısı

Genel bir yinelemeli süzgeç yapısında çıkış, önceki çıkış örnek değerlerinden elde edilen geri besleme ve önceki giriş örnek değerleri kullanılarak oluşturulmaktadır. Süzgeç denklemi,

$$y[n] = a_0 x[n] + \ldots a_M x[n-M] - b_1 y[n-1] - \ldots b_N y[n-N]$$
$$y[n] = \sum_{i=0}^{M} a_i x[n-i] - \sum_{j=1}^{N} b_j y[n-j] \qquad (2.10)$$

eşitliği ile ifade edilmektedir.

2.1.6 SDY süzgeç ile yinelemeli süzgecin karşılaştırılması

SDY ve yinelemeli türden süzgeçler arasında bir karşılaştırmaya gidildiğinde, iki temel yapının da olumlu ve olumsuz yanları olduğu görülmektedir. Genel olarak, doğrusal faz yanıtlı bir SDY süzgeç tasarımı oldukça kolay bir biçimde gerçeklenebilmektedir. Doğrusal faz yanıtlı süzgeç yapıları, frekans bağımlı gecikmeler nedeniyle ortaya çıkan faz gürültüsü

oluşumunu önledikleri için özellikle ses işleme uygulamalarında sıklıkla kullanılmaktadır. SDY süzgeçlerin bir diğer tercih nedeni de, geri beslemesiz yapıları sayesinde daima kararlı durumda bulunmaları ve asla rezonansa girmemeleridir. Bununla birlikte SDY süzgeç tasarımında, benzer frekans yanıtına sahip bir yinelemeli yapıya göre daha yüksek bir işlem hacmine ve daha büyük sığalarda belleğe ihtiyaç duyulmaktadır.

Geri besleme yoluyla önceki çıkışlara ait örnek değerlerinden yararlanıldığı için, yinelemeli yapı kullanılarak istenilen frekans yanıtı oldukça az sayıda işlem adımıyla elde edilebilmektedir. Bununla birlikte, yinelemeli süzgeçlerle yapılan uygulamalarda faz gürültüsü ve ötme (ringing) sorunlarının ortaya çıkma olasılığı daha yüksektir (Preis, 1982). Yinelemeli süzgeç, geçişlerden sonra rezonansa girerek geçişlerin zamanda yayılmasına ve yüksek frekans bileşenlerinin bozulmasına yol açabilmektedir.

2.1.7 Tarak süzgeç

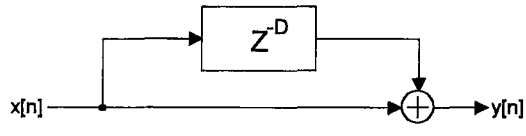
Tarak süzgeç, giriş işaretinin frekans spektrumunda birbirini izleyen düzenli iniş-çıkışlar oluşturmaktadır. İniş-çıkışlar frekans düzleminde birbirine eşit aralıklarla konumlanmaktadır.

2.1.7.1 SDY tarak süzgeç

Basit bir ileri beslemeli (SDY) tarak süzgeç denklemi,

$$y[n] = x[n] + x[n - D] \tag{2.11}$$

eşitliği ile verilmektedir. Şekil 2.10'da görüldüğü gibi, süzgeç çıkışı, süzgeç girişi ve girişin D örnek kadar geciktirilmiş değerinin toplamından elde edilmektedir.



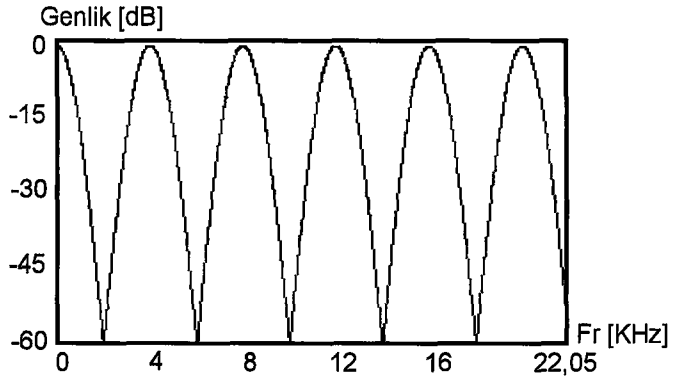
Şekil 2.10 Basit SDY tarak süzgeç yapısı.

SDY tarak süzgeç yapısı, SDY alçak geçiren süzgeç yapısıyla benzerlikler göstermektedir. Bununla birlikte, SDY tarak süzgeç yapısında, SDY alçak geçiren süzgeçten farklı olarak, ne özgün işaret, ne de D örnek geciktirilmiş işaret ilgili bir süzgeç katsayısı ile çarpılmamaktadır. Ayrıca, D geciktirme zamanı alçak geçiren süzgece kıyasla oldukça uzun seçilmektedir. 48 KHz örnekleme hızı için, bir örneklik bir geciktirici yaklaşık 0.02 ms süreli bir geciktirmeye neden olmaktadır. Yalnızca bir örneklik bir geciktirme süresi kullanılarak tasarlanan bir süzgecin girişine etkisi, hafif bir alçak geçiren süzgeç etkisinden farklı olmamaktadır. Geciktirmenin 0.1 ms'den büyük tutulduğu süzgeç yapılarında ise, faz giderme (phase cancellation) etkisi nedeniyle, girişin frekans spektrumunda genliğin sıfıra düştüğü sıfır noktaları oluşmaya başlamakta ve tarak süzgeç etkisi ortaya çıkmaktadır.

Tarak süzgeç, geciktirilmiş işaret ile özgün işaret arasında bir güçlendirme ve faz giderme etkisine neden olmaktadır. Pozitif toplamalı tarak süzgeçte (2.11) olduğu gibi, özgün işaretle geciktirilmiş işaretin birbirine eklendiği durumlarda, D örnek geciktirme ve F_S örnekleme hızı için, süzgeç çıkışındaki ilk tepe noktası,

$$f = F_S / D \ [Hz/Örnek] \tag{2.12}$$

frekansında ve ardışıl tepe noktaları $2f, 3f, 4f, \ldots$ frekanslarında oluşmaktadır. Sonuç olarak, bu tür bir pozitif toplamalı tarak süzgecin, f temel frekansı ve tüm bileşenleri üzerinde güçlendirici bir etki yarattığı görülmektedir.



Şekil 2.11 Basit SDY tarak süzgeç frekans yanıtı.

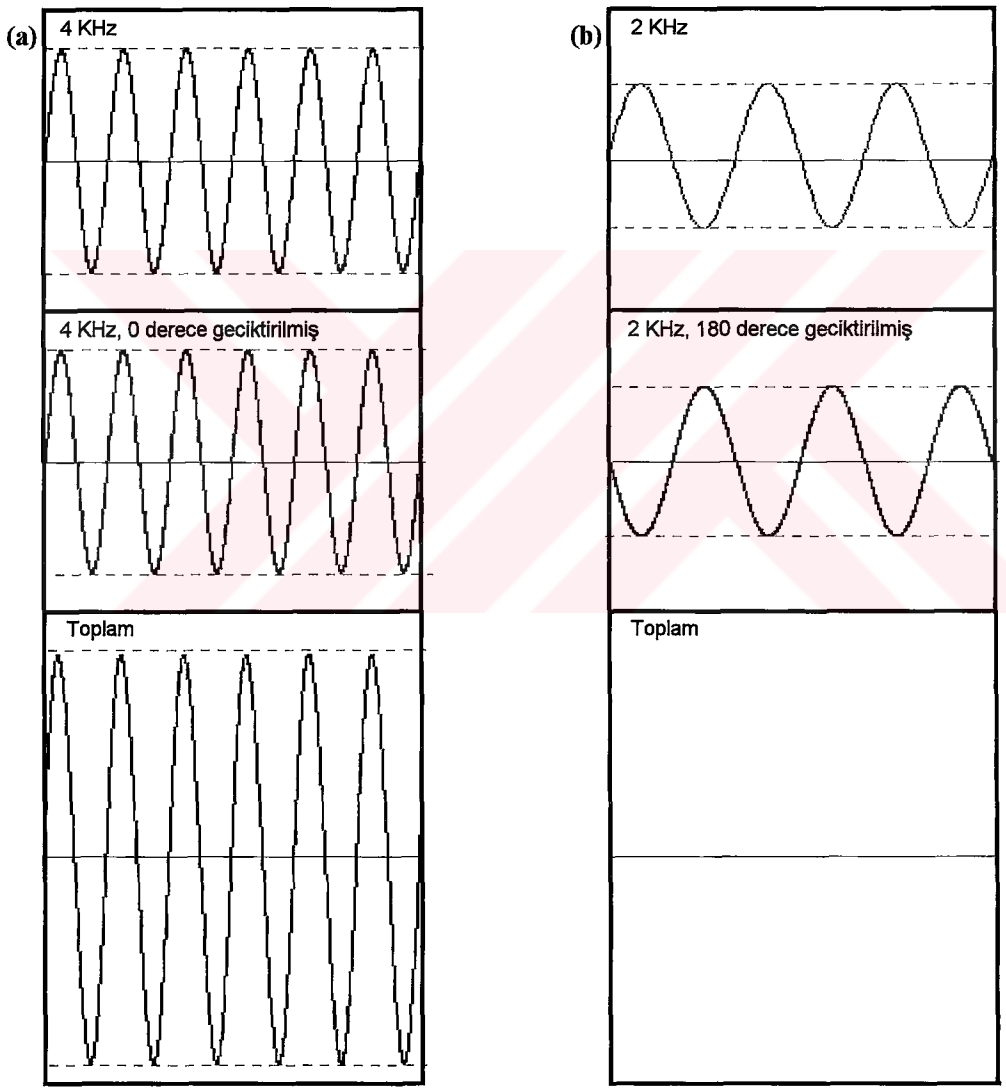
Şekil 2.11'de, 44100 Hz örnekleme hızı ve 11 örnek geciktirme için, pozitif toplamalı bir tarak süzgece ait frekans yanıtı gösterilmiştir. Şekilde görüldüğü gibi, ilk tepe noktası $f = 44100/11 \cong 4$ KHz frekansında oluşmaktadır. Tepe noktalarının ortaya çıkışı ardışıl bir biçimde, 4 KHz aralıklarla, Nyquist frekansına kadar devam etmektedir. Tarak süzgece ait sıfır noktalarının da 2 KHz, 6 KHz, ... frekanslarında, yine 4 KHz aralıklarla Nyquist frekansına kadar devam ettiği şekilde görülmektedir. Düşük frekanslarda, gecikmenin işaretin fazı üzerinde herhangi bir etkisi yoktur ve özgün işaretle geciktirilmiş işaretin toplanması süzgeç çıkışında elde edilen işaretin genliğinin artmasına yol açmaktadır. Frekans arttıkça, geciktirilmiş işaret, özgün işaretin fazı 180 derece ötelenmiş haline yaklaşmaktadır. 2 KHz frekansında, 0.25 ms geciktirme nedeniyle, geciktirilmiş işaret ile özgün işaret arasındaki faz farkı tam 180 dereceye ulaşmakta ve bu iki işaretin toplamından elde edilen süzgeç çıkışı sıfıra eşit olmaktadır. 2 KHz'de oluşan bu ilk sıfır noktasından sonra işaretler arasındaki faz farkı 0 veya 360 derece olana kadar özgün işaretle geciktirilmiş işaret toplamı, çıkışa genliği artan bir işaret olarak yansımakta ve 4 KHz frekansında ilk tepe noktasına ulaşılmaktadır. İniş çıkışlar özgün işaretle geciktirilmiş işaret arasındaki faz farkına bağlı olarak 4 KHz aralıklarla Nyquist frekansına kadar birbirini izleyerek tekrarlanmaktadır.

Faz güçlendirme ve faz gidermenin çıkış üzerindeki etkileri sırasıyla Şekil 2.12.a ve Şekil 2.12.b'de gösterilmiştir.

Çizelge 2.1 Tarak süzgeç frekans yanıtındaki tepe aralıklarının, geciktirme süresine bağlı olarak değişimi.

Geciktirme Süresi $[ms]$	Tepe Aralıkları $[KHz]$
10	0,1
2	0,5
1	1
0.5	2
0.25	4
0.125	8
0.1	10

Çizelge 2.1'de, tarak süzgeç geciktirme süresi D ile tarak süzgeç frekans yanıtındaki tepe noktalarının oluşma aralıkları arasındaki ilişki verilmiştir. Çizelgede belirtildiği gibi, geciktirme süresi arttırıldıkça tarak süzgeç frekans yanıtındaki iniş-çıkışlar sıklaşmaktadır. Tarak süzgecin etkisi en belirgin bir biçimde 5 ms civarında geciktirmeler kullanılarak elde edilmektedir. 5 ms'den küçük geciktirmeler için, tarak süzgeç daha çok alçak geçiren karakteristikli bir çıkış üretmektedir. 5 ms'den büyük geciktirmelerde ise, süzgeç çıkışındaki iniş-çıkışlar insan kulağı tarafından belirgin bir biçimde algılanamayacak sıklıkta birbirini izlemektedir.



Şekil 2.12 Faz güçlendirme ve faz gidermenin tarak süzgeç çıkışı üzerindeki etkileri.

Negatif toplamalı tarak süzgecin çıkışı ise,

$$y[n] = x[n] - x[n-D] \tag{2.13}$$

eşitliği uyarınca, özgün işaret ile D örnek geciktirilmiş işaretin farkından elde edilmektedir.

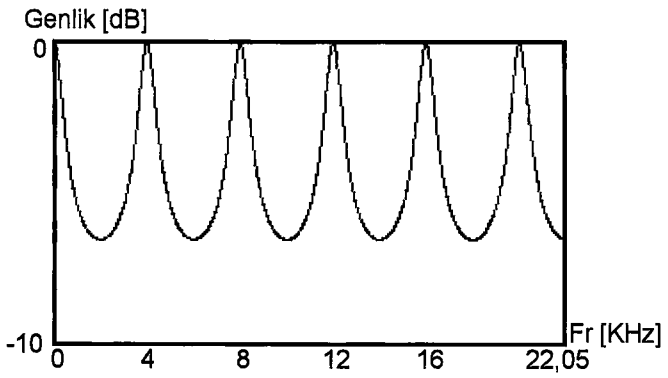
Negatif toplamalı tarak süzgecin çıkışı, pozitif toplamalı tarak süzgeç çıkışının, fazı 180 derece ötelenmiş halidir. Negatif toplamalı tarak süzgecin frekans yanıtında, ilk sıfır noktası 0 Hz frekansında ve ardışıl sıfır noktaları $f, 2f, 3f, 4f, \ldots$ frekanslarında oluşmaktadır. Tepe noktaları ise, $f/2, 3f/2, 5f/2, \ldots$ frekanslarında yer almaktadır. Sonuç olarak, negatif toplamalı tarak süzgeç, $f/2$ temel frekansı ve tüm bileşenleri üzerinde güçlendirici bir etki yaratmaktadır.

2.1.7.2 Yinelemeli tarak süzgeç

Yinelemeli bir tarak süzgecin çıkışı, giriş işareti ile çıkıştan alınan D örnek geciktirilmiş bir geri besleme işaretinin toplamından elde edilmektedir.

$$y[n] = a.x[n] + b.y[n-D] \tag{2.14}$$

(2.14) eşitliğinde verilen yinelemeli tarak süzgeç bağıntısında a ve b, 0 ila 1 arasında değerler alabilen süzgeç katsayılarıdır. Şekil 2.13'de, 44100 Hz örnekleme hızı için, yinelemeli tarak süzgece ait frekans yanıtı gösterilmiştir.



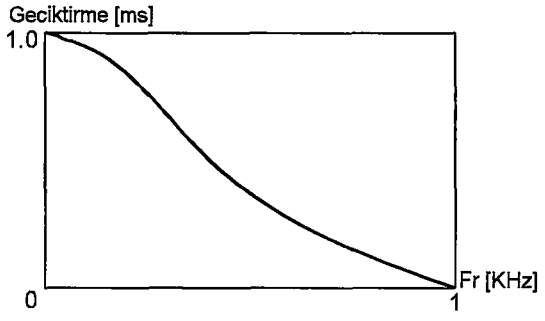
Şekil 2.13 Yinelemeli tarak süzgeç frekans yanıtı

Yinelemeli tarak süzgeç, özellikle *b* süzgeç katsayısı için seçilen değere bağlı olarak, benzeri bir SDY tarak süzgece göre işaret üzerinde daha belirgin ve daha vurgulu bir salınım etkisi yaratmaktadır. Bununla birlikte, *b* süzgeç katsayısının çok yüksek değerlerde seçilmesi, girişe eklenen geri beslemenin gereğinden büyük değerler alarak, süzgeç çıkışında sayısal bir taşma oluşturmasına yol açmaktadır.

2.1.8 Tüm geçiren süzgeç

Tüm geçiren bir süzgeç, girişine uygulanan bir kararlı durum işaretinin tüm frekans bileşenlerini, genlikleri üzerinde herhangi bir değişim yaratmadan çıkışına iletmektedir. Bununla birlikte, tüm geçiren süzgeç, giriş işareti üzerinde frekansa bağımlı bir faz ötelemesi oluşturmakta ve giriş işaretinin farklı frekans bölgelerinin farklı miktarlarda geciktirilmesine yol açmaktadır. Frekansa bağımlı bu tür bir gecikme, yayılma (dispersion) olarak adlandırılmaktadır

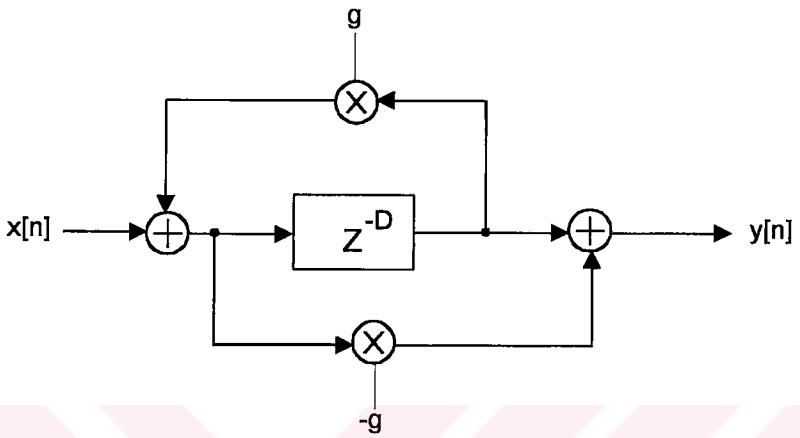
Şekil 2.14'de, tüm geçiren bir süzgecin geciktirme-frekans eğrisi gösterilmiştir. Şekilde görüldüğü gibi, düşük frekanslar daha yüksek miktarlarda geciktirilmektedir. Bir başka deyişle, frekans arttıkça geciktirme miktarı azalmaktadır. Tüm geçiren bir süzgecin giriş işareti üzerindeki etkisi, ani ataklar ve düşüşler şeklinde kendini göstermektedir. Fazı frekansa bağımlı olarak ötelenen giriş işareti, çıkışta daha renkli bir hale gelmektedir (Preis, 1982; Chamberlin, 1985; Deer vd., 1985).



Şekil 2.14 Tüm geçiren süzgeç geciktirme-frekans eğrisi.

$$y[n] = -g.x[n] + x[n-D] + g.y[n-D] \tag{2.15}$$

(2.15) eşitliğinde, 0 Hz ile $F_s/2$ Hz arasında düz bir frekans yanıtına sahip, basit bir tüm geçiren süzgeç denklemi verilmiştir. D geciktirme süresi büyük değerlerde seçildiğinde, tüm geçiren süzgeç çıkışında bir dizi, zamanla sönümlenen yankı işareti oluşmaktadır. Tüm geçiren süzgecin giriş üzerinde yaratığı bu etki, tüm geçiren çınlatıcı birimi tasarımında kullanılmaktadır (Bölüm 2.3.6.2).



Şekil 2.15 Tüm geçiren süzgeç yapısı.

Şekil 2.15'de, (2.15) eşitliği ile verilen tüm geçiren süzgeç yapısı gösterilmiştir (Schroeder, 1961, 1962). Şekilde görüldüğü gibi, tüm geçiren süzgeç yapısı, g süzgeç katsayısıyla denetlenen bir yinelemeli tarak süzgeç yapısını içermektedir. Devrede ayrıca, girişten alınan $-g$ kazançlı bir ileri besleme bulunmaktadır. Tarak süzgecin, giriş işareti frekans bileşenlerinin genlikleri üzerinde herhangi bir değişim oluşturması, bu negatif ileri besleme sayesinde engellenmektedir. Bununla birlikte, negatif ileri besleme tarak süzgecin geciktirme ve yankı özelliklerinde bir değişime yol açmamaktadır.

Derece cinsinden alındığında, tüm geçiren süzgecin oluşturduğu faz ötelemesinin, geciktirmenin logaritmik bir fonksiyonu olduğu görülmektedir. Örneğin, 100 ms değerinde bir geciktirme, düşük frekanslarda yalnızca bir kaç derecelik bir ötelemeye neden olurken, 10 KHz frekansında aynı geciktirme, 360 derecelik bir öteleme oluşturmaktadır.

Devrilme frekansı ve geçiş aralığı, tüm geçiren bir süzgeci tanımlamakta kullanılan iki temel parametredir. Tüm geçiren süzgecin oluşturduğu faz ötelemesinin 180 dereceye ulaştığı

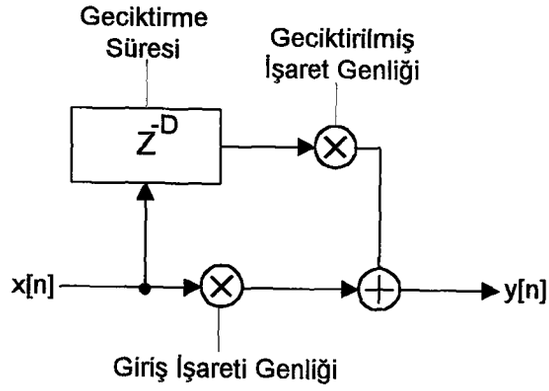
frekans, devrilme frekansı olarak adlandırılmaktadır. Geçiş aralığı ise, süzgeç iletiminin keskinliğini belirlemekte ve 0 derece faz ötelemesinden 360 derece faz ötelemesine kadar olan bir aralıkta değerler alabilmektedir.

2.2 Geciktirme Etkileri

2.2.1 Sayısal geciktirici

Zaman geciktirme özellikle müzikal işaret işleme uygulamalarında sıklıkla kullanılan bir yöntemdir. Geciktirici veya geciktirme hattı, girişine gelen sayısal bir işarete ait örnek değerlerini belleğinde saklayarak belirli bir süre sonra çıkışına ileten sayısal yapılara denir. Geciktirilmiş işaretle özgün işaretin karıştırılması yoluyla pek çok farklı müzikal etki elde edilebilmektedir. Uygulama esnasında geciktirme süresinin sabit tutulduğu yapılar sabit zamanlı geciktirici olarak adlandırılmaktadır. Geciktirme süresinin düzenli olarak değiştirildiği, uygulama esnasında örnekleme peryoduna eşit aralıklarla geciktirme süresine birbirinden farklı değerlerin verildiği yapılara ise değişken zamanlı geciktiriciler denilmektedir.

2.2.2 Sayısal geciktirici ile SDY AGS ve tarak süzgecin karşılaştırılması



Şekil 2.16 Basit sayısal geciktirici yapısı.

Şekil 2.16'da basit bir sayısal geciktirici gösterilmiştir. Sayısal geciktirici ile, sırasıyla Şekil 2.1 ve Şekil 2.10'da gösterilen basit SDY AGS ve SDY tarak süzgeç arasında yapısal bir

benzerlik bulunmaktadır. Seçilen geciktirme süresi D, bu yapıları birbirinden ayıran temel özelliktir. Alçak geçiren süzgeç yapısında bir örneklik bir geciktirici kullanılarak ardışıl örneklerin ortalamasının alınması sağlanmaktadır. Tarak süzgeç yapısında kullanılan geciktirici, kayda değer bir tarak süzgeç etkisinin elde edilebilmesi için 0.1 ms ila 1 ms arasında bir geciktirme oluşturacak şekilde seçilmektedir. Sayısal geciktirici yapısında ise, elde edilen geciktirme 1 ms'den büyük değerler almaktadır.

2.2.3 Sabit zamanlı geciktirici kullanılarak gerçeklenen ses etkileri

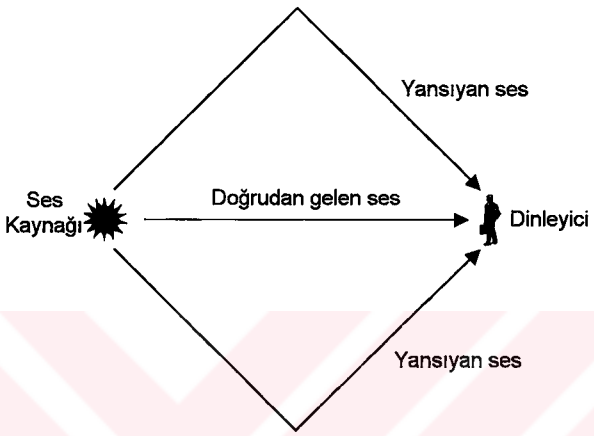
Sabit zamanlı bir geciktirici kullanılarak oluşturulan bir yapının, girişindeki müzikal işaret üzerinde yaratacağı etkinin karakteristiği belirleyen en temel özellik geciktirme süresidir. Geciktirme süresi üç ana zaman dilimi halinde gruplandırılabilir;

- Kısa süreli geciktirme (10 ms'den küçük değerler)
- Orta süreli geciktirme (10 ms ile 50 ms arasında değerler)
- Uzun süreli geciktirme (50 ms'den büyük değerler)

Kısa süreli bir geciktirmeye sahip yapılar, daha çok giriş işaretinin spektral özellikleri üzerinde bir değişim yaratmak amacıyla kullanılmaktadır. Örneğin, bir veya birkaç örneklik bir geciktiricinin çıkışı ile özgün işaretin toplanması yoluyla, SDY alçak geçiren süzgeç yapısının giriş işareti üzerinde oluşturduğu etkiye benzer bir etki, sabit zamanlı bir geciktirici kullanılarak elde edilebilmektedir. Geciktirme süresinin 0.1 ms ile 10 ms arasında bir değerde seçilmesi ise, yapının işaret üzerinde SDY tarak süzgecinkine benzer bir değişim yaratmasına yol açmaktadır.

Orta süreli bir geciktirmeye sahip yapılar zayıf bir müzikal işaretin duyulabilirliğini arttırmakta veya başka bir deyişle, işaretin etkisini belirginleştirmekte kullanılmaktadır. Özellikle popüler müzikte, vokal, davul veya tuşlu çalgılara ait ses bileşenlerinin duyulabilirliğinin arttırılmasında, orta süreli bir geciktirmeye sahip sabit zamanlı geciktiricilerden sıklıkla yararlanılmaktadır. Orta süreli bir geciktirme, müzikal işarete ait her bir örnek değerlerinin, geciktirme süresi kadar sonra kendini tekrar etmesine yol açmaktadır.

Bu sayede dinleyicide, müzikal işaretteki boğukluğun, uğultunun azaldığı ve işaretin belirginleştiği hissi uyandırılmaktadır. Dikkat edilmesi gereken nokta, sabit zamanlı geciktiricinin bu işlemi işaretin genliği üzerinde herhangi bir fiziksel yükseltmeye neden olmadan gerçekleştirdiğidir. Orta süreli bir geciktirme yoluyla elde edilen müzikal etkilerin en sık kullanılanlarından biri, çiftleme (doubling) etkisidir. Çiftleme etkisi, 10 ms ila 50 ms arasında bir değerde geciktirilmiş işaret ile özgün işaretin toplanması yoluyla yaratılmaktadır.



Şekil 2.17 Yankı olayı.

Uzun süreli bir geciktirmeye sahip yapılar kullanılarak, yankılı (echo) bir işaret, yani özgün işaretin geciktirme süresi kadar sonra duyulan ayrık tekrarları elde edilebilir. Doğada yankı olayı, Şekil 2.17'de gösterildiği gibi, sesin kaynağından yayılarak bir yüzeye çarpıp yansıması ve ortamda bulunan bir dinleyiciye, özgün işaretten belirli bir süre sonra, özgün işaretin ayrık bir tekrarı olarak ulaşması yoluyla gerçekleşmektedir.

Ses saniyede 344 metre hızla ilerlediğinden, 1 ms değerinde bir geciktirme ses kaynağından dinleyiciye olan uzaklığın yaklaşık 0,3 metre olduğu bir yankı yoluna denk düşmektedir. Bu değerde bir uzaklık ile algılanabilir bir yankı etkisi oluşturulamamaktadır. Algılanabilir, ayrık bir yankı etkisi oluşturabilmek için, ses kaynağından dinleyiciye olan uzaklığın en az 16 metre olduğu bir yankı yoluna gereksinim duyulmaktadır. Bu nedenle, yankı etkisi yaratmakta kullanılacak sabit zamanlı geciktiriciye ait geciktirme süresi 50 ms'den büyük değerlerde seçilmektedir.

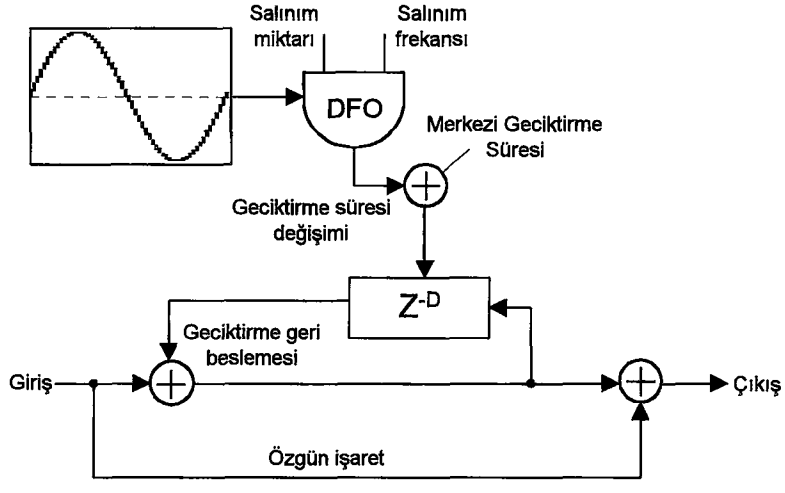
2.2.4 Değişken zamanlı geciktirici kullanılarak gerçeklenen ses etkileri

2.2.4.1 Dalgalandırma etkisi

Üzerinde dalgalandırma (flange) etkisi yaratılmış bir işaret, özgün işaret ile değişken sürelerle geciktirilmiş işaretin toplamından elde edilmektedir. Sayısal geciktiricinin geciktirme süresi değerleri düzenli aralıklarla değiştirilmektedir. Bu işlem, düşük frekanslı bir sinüs veya üçgen dalga osilatörü aracılığıyla gerçekleştirilmektedir. Osilatörün salınım frekansı, genellikle 0.1 Hz ila 20 Hz arasında bir değer alacak şekilde seçilmektedir.

Dalgalandırma etkisi, işaretin spektrumu üzerinde tarak süzgeç etkisine benzer bir değişim meydana getirmektedir. D geciktirme süresi, F_S örnekleme frekansı ve $f = F_S/D$ olmak üzere, işaretin spektrumunda $2f, 3f, 4f, \ldots$ frekanslarında tepe noktaları ve $f/2, 3f/2, 5f/2, \ldots$ frekanslarında sıfır noktaları oluşturmaktadır.

Yukarıda yapılan tanımlamalar uyarınca, dalgalandırma etkisinin değişken geciktirme süreli bir SDY tarak süzgeç yapısı aracılığıyla gerçekleştirilmesi olası gözükmektedir. Bununla birlikte, uygulamalarda daha çok, değişken zamanlı bir geciktiriciye sahip yinelemeli tarak süzgeç yapıları kullanılmaktadır.



Şekil 2.18 Yinelemeli tarak süzgeç ile oluşturulmuş dalgalandırıcı yapı.

Şekil 2.18'de, değişken zamanlı bir yinelemeli tarak süzgeç kullanılarak oluşturulmuş bir dalgalandırıcı (flanger) gösterilmiştir. Genellikle, devredeki geri beslemenin pozitif ya da negatif olarak seçileceğinin kararı, belirli bir giriş işareti üzerinde hangisinin daha doyurucu bir etki oluşturduğuna bakılarak deneme yanılma yoluyla verilmektedir. Devredeki düşük frekans osilatörü, sayısal geciktiriciye ait geciktirme süresinin merkezi geciktirme süresi etrafında sinüzoidal bir biçimde değişim göstermesini sağlamaktadır. Sayısal geciktiriciden alınan geri besleme ile özgün işaretin kendilerine ait birer çarpan tarafından denetlenmesi olasıdır. Bu sayede, geciktirilmiş işaret ile özgün işaretin birleşme oranları değiştirilerek dalgalandırma etkisinin derinliği ayarlanabilir hale getirilmektedir. Buna ek olarak, geri beslemenin fazını çevirme olanağı da elde edilmektedir.

2.2.4.2 Koro etkileri

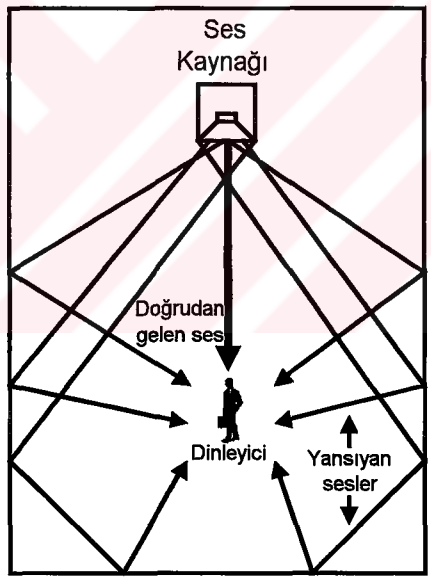
Koro (chorus) etkisi, adından anlaşılacağı gibi, tek bir çalgı veya tek bir insandan gelen bir ses işaretini, benzer seslerden oluşan bir korodan gelen bir ses işaretine dönüştürmektedir. Koro etkisini oluşturabilmek için, girişteki ses işaretinden türetilmiş, birbirlerine göre küçük farklılıklar gösteren bir dizi ses işaretinin elde edilmesi gerekmektedir. Bu bir dizi ses işareti, girişteki işaretin farklı değerlerde geciktirilmesiyle, giriş işareti temel frekans bileşenleri üzerindeki küçük oynamalarla ve eşzamansız bir titreşim (vibrato) yoluyla elde edilebilmektedir.

Koro etkisi oluşturmada yararlanılan yöntemlerden biri, çok çıkışlı bir değişken zamanlı geciktirici kullanmaktır. Geciktiricinin her bir çıkışı farklı değerlerde geciktirme sürelerine sahiptir. Geciktirme süreleri, dar bir aralık üzerinden, düzenli olarak değiştirilmektedir. Giriş işareti üzerinde yaratılan etki, değişken zamanlı ve değişken perdelerde gezinen bir çiftleme etkisine benzetilebilir. Yapılan işlem, giriş işaretini geciktirme süreleri dalgalandırma etkisi oluşturmak için gerekli olandan daha büyük değerlerde seçilmiş, paralel bağlı bir dizi dalgalandırıcıya göndermekle eştir. Dalgalandırıcılar negatif geri beslemeli olarak kullanılarak, geciktirilmiş işaretlerin fazlarının çevrilmesi yoluyla daha belirgin bir koro etkisinin yaratılması sağlanmaktadır. Negatif geri besleme kullanımı ayrıca, rezonansa girme ve çıkışta sayısal bir taşmayla karşılaşma sorunlarına karşı bir önlem niteliği de taşımaktadır.

Koro etkisi oluşturmada yararlanılan diğer bir yöntem ise, giriş işaretini frekans bileşenlerine ayırmak ve her bir frekans bileşenini ayrı ayrı öteleyerek bileşenler arasındaki harmonik ilişkiyi zayıflatmaktır. Öteleme işlemi, her bir frekans bileşenine ortak bir değerin eklenmesi yoluyla gerçekleştirilebilmektedir. Bu ortak değer, dar bir aralık üzerinden rasgele bir biçimde değiştirilmektedir. Örneğin, 220 Hz, 440 Hz ve 880Hz'de frekans bileşenlerinin olduğu ve her bir bileşene 10 Hz'lik bir öteleme yaptırıldığı varsayıldığında, bileşenler 230 Hz, 450 Hz ve 890 Hz değerlerine taşınmakta ve bu sayede aralarındaki harmonik ilişki zayıflatılmaktadır. Bu tür bir frekans ötelemeli yapı, değişken zamanlı bir geciktirici kullanılarak tasarlanabilir (Chamberlin, 1985).

2.3 Çınlama

2.3.1 Çınlamanın genel tanımı



Şekil 2.19 Kapalı bir mekanda çınlama olayının meydana gelişi.

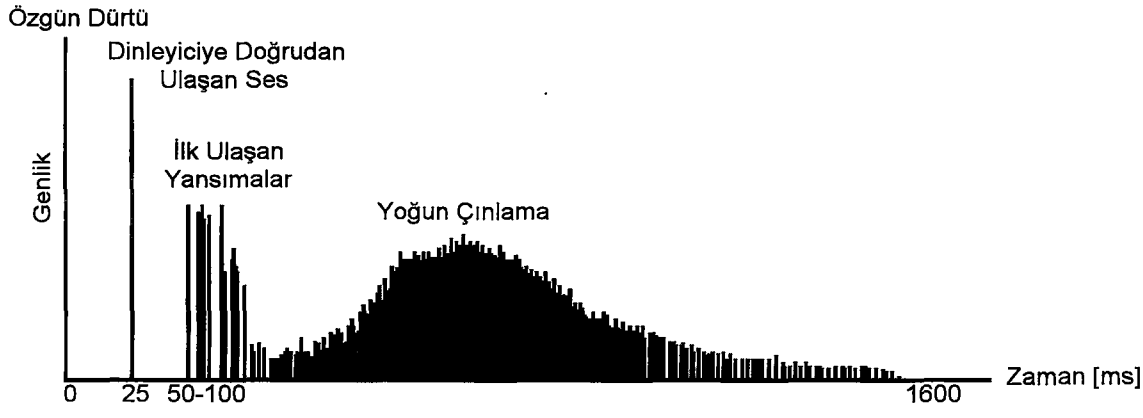
Çınlama (reverberation), konser salonları veya geniş odalar gibi yüksek tavanlı ve yansıtıcı yüzeylere sahip mekanlarda oluşan doğal bir akustik etkidir. Bu tür bir ortamda, herhangi bir ses kaynağı tarafından üretilen bir işaret, tavana, duvarlara ve tabana çarpıp yansımakta ve çok sayıda, birbirlerine oldukça yakın aralıklı yankı işaretleri oluşmaktadır (Şekil 2.19). Oluşan bu yankı işaretleri, dinleyiciye ulaşana dek, ortamda bulunan yansıtıcı yüzeylere

çarpıp yansıyarak dolanan bir yol izlediğinden, dinleyiciye doğrudan ulaşan özgün ses ile aralarında belirli bir zaman farkı bulunmaktadır. Yansıyan işaretler özgün işarete göre, genellikle daha düşük seviyeli genliklere sahiptir ve kısa süreli bir geciktirmeye uğramıştır. Ayrıca, yüzeylere çarpıp yansımaları ve havada katettikleri yol nedeniyle, yansıyan işaretlerin yüksek frekans bileşenlerinden bir kısmı bastırılmıştır. İnsan kulağı bu yüzden, doğrudan ulaşan özgün işaretle yansıyan işaretler arasındaki farkı algılayabilmektedir. Ses kaynağının ürettiği işaretin doğrudan algılanışından bir süre sonra, çok sayıda ve yoğun bir yankı işaretleri topluluğu dinleyiciye ulaşmaktadır. Bu yoğun yankı işaretleri dizisi, kulakta, kalıcı, etkisi hemen kaybolmayan bir çınlama etkisi bırakmaktadır.

Küçük ve dar bir stüdyoda çınlama etkisi çok az olduğundan, cansız, zayıf ve derinlik hissinden yoksun bir ses kaydı elde edilmektedir. Bu nedenle, özellikle müzikal ses işaretlerinin kaydında çınlama etkisinin yapay olarak yaratılmasına ihtiyaç duyulmaktadır.

2.3.2 Bir odanın dürtü yanıtı

Dürtü yanıtı analizi, konser salonları gibi fiziki mekanların veya yapay çınlatıcı devrelerinin tasarımında sıklıkla başvurulan bir yöntemdir. Çınlatma özelliğine sahip doğal bir mekanın dürtü yanıtı Şekil 2.20'de gösterilmiştir. Çınlama eğrisi, en yüksek tepe noktasına, üstele yakın bir artışla, yaklaşık yarım saniye içinde ulaşmakta ve artışı kadar keskin olmayan, yumuşak bir eğimle sönümlenmektedir.



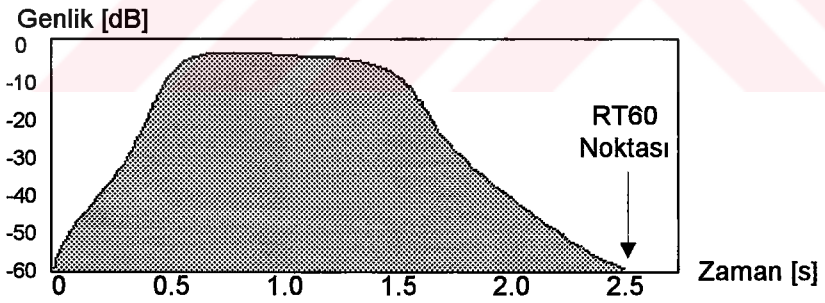
Şekil 2.20 Bir odanın dürtü yanıtı.

Şekil 2.20'de görüldüğü gibi, çınlama olayı, dinleyiciye doğrudan ulaşan ses, ilk ulaşan yansımalar ve yoğun çınlama bölgesi olmak üzere üç ana kısımdan oluşmaktadır. Kaynaktan çıkan sesin doğrudan dinleyiciye ulaşması için geçen süre, ön gecikme (predelay) olarak adlandırılmaktadır. Şekil 2.20'de, ön gecikme süresi 25 ms değerinde alınmıştır.

Konser salonu veya yapay çınlatıcı tasarımlarında, çınlama eğrisine ait tepe noktalarının rasgele sürelerle birbirini izlemesine çalışılmaktadır. Tepe noktalarının birbirini düzenli aralıklarla izlemesi, rezonansa yol açarak algılanan seste bir ötme etkisi oluşturduğu için kaçınılması gereken bir durumdur.

2.3.3 Çınlama süresi (RT60)

Çınlama süresi, çınlama olayını tanımlamakta kullanılan önemli özelliklerden biridir. Çınlamanın, tepe genliğin 60 dB altına düşmesi veya başka bir deyişle, tepe seviyesinin binde biri değerinde bir enerjiye ulaşması için geçen süre, çınlama süresi olarak adlandırılmaktadır (Şekil 2.21). Konser salonlarına ait RT60 süreleri, genellikle 1,5 ila 3 saniye arasında değerler almaktadır.



Şekil 2.21 Çınlama süresi.

2.3.4 Sayısal çınlatıcılar

Sayısal bir çınlatıcı, dürtü yanıtı bir odanın dürtü yanıtına (Şekil 2.20) benzeyecek şekilde tasarlanmış sayısal bir süzgeçtir. Sayısal çınlatıcı tasarımında, sesin bir oda veya herhangi kapalı bir mekan içindeki dağılımına benzer bir etki oluşturabilmek amacıyla, geciktirme

elemanları, karıştırıcılar ve farklı türden süzgeçler bir arada kullanılmaktadır. İlk sayısal çınlatıcı tasarımları, 1960'lı yılların başlarında, Manfred Schroeder tarafından Bell Laboratuvarlarında gerçekleştirilmiştir (Schroeder, 1961, 1962).

2.3.5 Sayısal bir çınlatıcıyı oluşturan temel kısımlar

Çınlama olayı, Şekil 2.20'de gösterildiği gibi, üç ana kısımdan oluşmaktadır:

- Dinleyiciye doğrudan, herhangi bir yerden yansımadan, ilk olarak ulaşan ses,
- Doğrudan ulaşan sesten hemen sonra dinleyicinin algıladığı, bir dizi ayrık yankıdan oluşan, ilk ulaşan yansımalar,
- Oluşması ve sönümlenmesi için ilk iki temel kısımdan daha uzun bir zaman gerektiren, zamanda sık aralıklarla yerleşmiş çok sayıda yankı işaretinden oluşan, yoğun çınlama bölgesi.

Sayısal çınlatıcı tasarımları, genellikle kullanıcının bu üç temel kısmı birbirinden bağımsız olarak denetlemesini mümkün kılacak şekilde gerçekleştirilmektedir. Çınlatıcı tasarımında, doğrudan gelen ses *"kuru (dry)"*, çınlatma işlemine sokulmuş ses ise *"ıslak (wet)"* olarak nitelendirilmekte ve kullanıcının, *ıslak/kuru* oranını değiştirerek çınlama etkisinin derecesini ayarlayabilmesine olanak tanınmaktadır.

Gerçeğe yakın bir çınlatma etkisi yaratabilmek için, zamanda çok sık aralıklarla birbirini izleyen, yüksek yoğunluklu yankı işaretlerine gereksinim duyulmaktadır. Herhangi bir konser salonunda, saniyede ortalama 1000 adet yankı işareti oluşmaktadır. Sayısal çınlatıcılar, kullanıcının yankı yoğunluğunu degiştirerek çınlatma etkisininin derecesini denetleyebileceği şekilde tasarlanmaktadır. Kullanıcı, ayrık yankı işaretlerinden yüksek değerde bir yankı yoğunluğuna kadar uzanan bir yelpaze üzerinden seçimini yapabilmektedir.

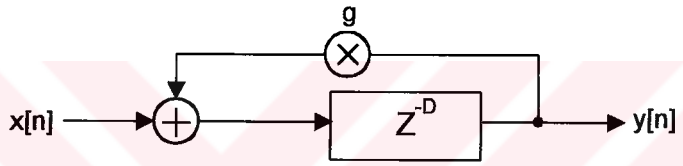
İlk ulaşan yansımaların yapay olarak gerçeklenmesinde çok çıkışlı bir sayısal geciktirici kullanılmaktadır. Sayısal geciktiricinin, her biri farklı geciktirme sürelerine sahip çok sayıda çıkışı bulunmaktadır. Yoğun çınlama bölgesini gerçekleyebilmek için ise, çok çıkışlı bir

sayısal geciktirici yetersiz kaldığından, farklı yapılarda çınlatma birimlerinden yararlanılması gerekmektedir. Yinelemeli tarak süzgeçler ve tüm geçiren süzgeçler, çınlatma birimlerinin tasarımında kullanılan iki temel yapıdır.

2.3.6 Çınlatma birimleri

2.3.6.1 Yinelemeli tarak süzgeç çınlatma birimi

Bölüm 2.1.7.2'de açıklandığı gibi, yinelemeli bir tarak süzgecin çıkışı, giriş işareti ile çıkıştan alınan D örnek geciktirilmiş g kazançlı bir geri besleme işaretinin toplamından elde edilmektedir (Şekil 2.22).



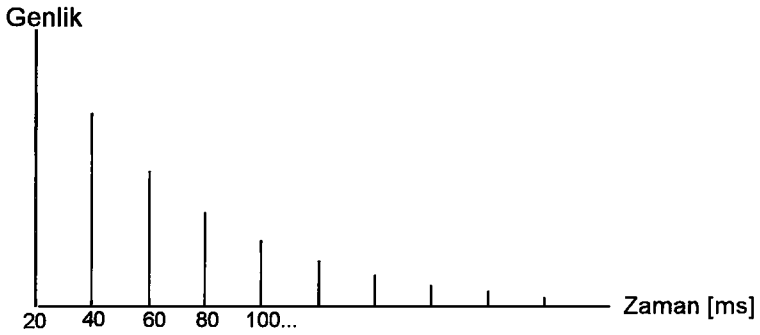
Şekil 2.22 Yinelemeli tarak süzgeç çınlatma birimi.

Geciktirme süresi D'nin 10 ms'den küçük seçildiği durumlarda tarak süzgeç, giriş işaretinin frekans yanıtında birbirini izleyen düzenli iniş-çıkışlar (tepe ve sıfır noktaları) oluşturarak işaret üzerinde spektral bir etki yaratmaktadır. Geciktirme süresinin 10 ms'den büyük değerlerde seçildiği durumlarda ise, tarak süzgeç çıkışında bir dizi, zamanla sönümlenen yankı işareti elde edilmektedir. Yankı işaretlerinin genliği üstel bir düşüş göstermekte ve g geri besleme katsayısı 1.0 değerine yaklaştıkça, süzgeç çıkışında elde edilen yankı işaretlerinin sayısı artmaktadır. Tarak süzgeç çıkışında 60 dB değerinde bir düşüş oluşana dek geçen süre, sönüm veya çınlama süresi olarak adlandırılmakta ve,

$$\begin{aligned} &\textit{sönüm süresi} = (60/-\textit{çevrim kazancı}).\textit{çevrim geciktirmesi} \\ &\textit{çevrim kazancı}\,[dB] = 20.\log_{10}(g) \\ &\textit{çevrim geciktirmesi}\,[s] = D/\textit{örnekleme hızı} \end{aligned} \qquad (2.16)$$

bağıntısıyla hesaplanmaktadır (Moore, 1990). Bağıntıda, g geri besleme kazancı desibel, D

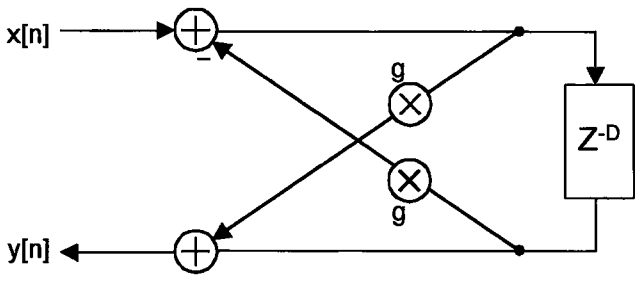
geciktirme süresi ise saniye cinsinden ifade edilmiştir. Şekil 2.23'de, geciktirme süresi 20 ms ve geri besleme katsayısı 0.7 değerinde seçilmiş bir tarak süzgece ait dürtü yanıtı gösterilmiştir.



Şekil 2.23 Yinelemeli tarak süzgeç çınlatma birimi dürtü yanıtı.

2.3.6.2 Tüm geçiren süzgeç çınlatma birimi

Bölüm 2.1.8'de açıklandığı gibi, tüm geçiren bir süzgeç, girişine uygulanan bir kararlı durum işaretinin tüm frekans bileşenlerini, genlikleri üzerinde herhangi bir değişim yaratmadan çıkışına iletmektedir. Bununla birlikte, tüm geçiren süzgeç, giriş işareti üzerinde frekansa bağımlı bir faz ötelemesi oluşturmakta ve giriş işaretinin farklı frekans bölgelerinin farklı miktarlarda geciktirilmesine yol açmaktadır. Tüm geçiren süzgecin giriş işareti üzerindeki etkisi ani ataklar ve düşüşler şeklinde kendini göstermektedir. Özellikle ses işaretleriyle yapılan uygulamalarda, süzgeç bu davranış biçimi nedeniyle çıkışta daha "*renkli*" bir ses işaretinin elde edilmesini sağlamaktadır.

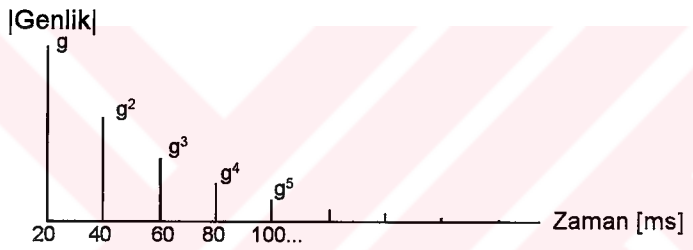


Şekil 2.24 Tüm geçiren süzgeç çınlatma birimi.

Şekil 2.24'de, tüm geçiren bir süzgeç yapısı gösterilmiştir. D geciktirme süresi, 5 ms ila

100 ms arasında yeterli derecede büyük bir değerde seçildiğinde, süzgeç çıkışında, büyük değerde bir geciktirmeye sahip bir tarak süzgecin çıkışında olduğu gibi, bir dizi, zamanla üstel olarak sönümlenen yankı işareti oluşmaktadır.

Şekil 2.25'de, geciktirme süresi 20 ms ve süzgeç katsayısı 0.6 değerinde seçilmiş bir tüm geçiren süzgece ait dürtü yanıtı gösterilmiştir. Şekilde görüldüğü gibi, süzgeç çıkışında üretilen darbeler (yankı işaretleri) birbirini eşit aralıklarla, düzgün dağılımlı bir biçimde izlemekte ve üstel olarak sönümlenmektedir. Bu karakteristik nedeniyle, tüm geçiren süzgeç girişine kısa süreli ve keskin geçişli bir ses işareti uygulandığında, tüm geçiren süzgeç çıkışında geciktirme süresine eşit bir peryotla birbirini izleyen ve üstel olarak sönümlenen bir dizi tekrar işareti elde edilmektedir. Sonuç olarak, geciktirme süresi uygun değerlerde seçildiğinde, tüm geçiren süzgeç giriş işareti üzerinde bir çınlatma etkisi yaratmaktadır.

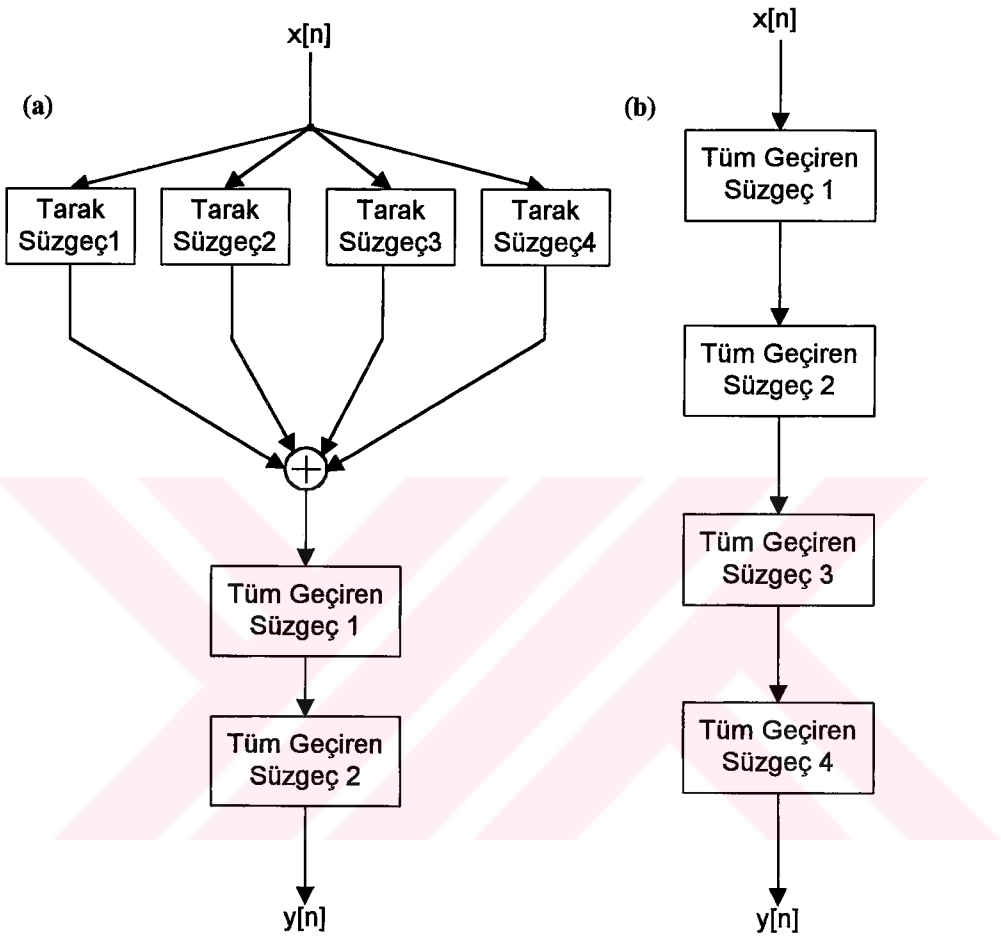


Şekil 2.25 Tüm geçiren süzgeç çınlatma birimi dürtü yanıtı.

2.3.7 Çınlatıcılar

Yinelemeli tarak süzgeç veya tüm geçiren süzgeç gibi çınlatma birimlerinin, bir dizi, zamanla sönümlenen yankı işareti üretmelerine rağmen, doyurucu bir çınlama etkisi yaratabilmek için çok daha sık aralıklarla birbirini izleyen, çok daha yoğun bir yankı işaretleri topluluğuna ihtiyaç duyulmaktadır. Bu nedenle, çok sayıda çınlatma biriminin bir araya getirilmesiyle oluşturulmuş tümleşik yapılarının tasarlanması gerekmektedir. Çınlatma birimlerinin paralel bağlı olarak kullanıldığı yapılarda, çıkışta elde edilen yankı işaretlerinin sayısı, her bir çınlatma biriminin ürettiği yankı işaretlerinin sayısının toplamına eşit olmaktadır. Seri bağlı çınlatma birimlerinden oluşan yapılarda ise, bir çınlatma biriminin ürettiği yankı işaretlerinden her biri, kendinden sonra gelen çınlatma biriminin çıkışında bir dizi yankı

işaretinin üretilmesine yol açmaktadır. Bu sayede çınlatıcı çıkışında, çok sayıda işaretten oluşmuş, yüksek yoğunluklu bir yankı işaretleri topluluğu elde edilmektedir. Seri bir çınlatıcının çıkışından alınan yankı işaretlerinin sayısı, yapıyı oluşturan her bir çınlatma biriminin ürettiği yankı işaretlerinin sayısının çarpımına eşittir.



Şekil 2.26 Schroeder'in (1961,1962) çınlatıcı yapıları: (a) dört adet paralel bağlı tarak süzgeç ve iki adet seri bağlı tüm geçiren süzgeçten oluşmuş çınlatıcı; (b) dört adet seri bağlı tüm geçiren süzgeçten oluşmuş çınlatıcı.

Şekil 2.26.a ve 2.26.b'de, Schroeder (1961, 1962) tarafından tasarlanan iki çınlatıcı yapı gösterilmiştir. Şekil 2.26.a'daki çınlatıcıda, paralel bağlı tarak süzgeçlerin oluşturduğu yankı işaretlerinin toplamı, seri bağlı iki tüm geçiren süzgeci tetiklemektedir. Şekil 2.26.b'deki yapı ise, dört adet seri bağlı tüm geçiren süzgeçten oluşmaktadır. Çınlatıcı çıkışında elde edilen yankı işaretlerinin sayısı, her bir tüm geçiren süzgecin oluşturduğu yankı işaretlerinin sayısının çarpımına eşittir.

Tarak süzgeçlerin seri, tüm geçiren süzgeçlerin ise paralel bağlı olarak kullanımı, çınlatıcı yapılarının tasarımında tercih edilmeyen yöntemlerdir. Tarak süzgeçler seri bağlandığı takdirde, süzgeçlerden birinin çıkışına ait bir frekans bileşeninin, kendinden sonra gelen diğer bir süzgeç tarafından bastırılması, çınlatıcı çıkışında elde edilen işaretin frekans özelliklerini olumsuz yönde etkilemektedir. Paralel bağlı tüm geçiren süzgeçler ise, oluşturdukları faz giderme etkisi nedeniyle çınlatıcı çıkışında düzensiz bir genlik değişimine yol açmaktadır.

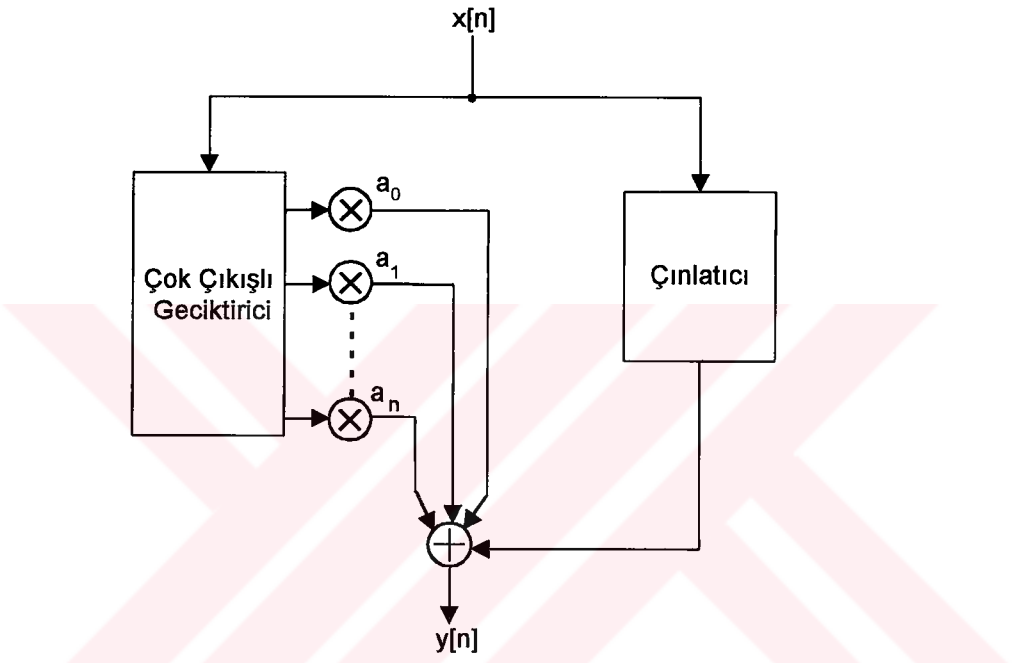
Çınlama etkisinin derecesi, yapıyı oluşturan her bir çınlatma birimine ait D geciktirme süreleri ve g geri besleme kazançları tarafından belirlenmektedir. Çınlatma birimi çıkışında üretilen yankı işaretleri, D geciktirme süresine eşit zaman aralıklarıyla oluşmaktadır. Yankı işaretlerinin ne kadar bir süre sonra sönümleneceği, yani birimin çınlama süresi ise, g geri besleme kazancına bağlı olarak değişmektedir.

Doğal bir çınlatma etkisi yaratabilmek için, çınlatıcıyı oluşturan birimlere ait geciktirme sürelerinin birbirlerine göre asal değerlerde, ortak bölenleri olmayacak şekilde seçilmesi gerekmektedir (Moorer, 1977, 1979). Aksi takdirde, çınlatma birimlerinin oluşturduğu yankı işaretleri, belirli zamanlarda birbirleriyle çakışarak ani sıçramalar oluşturmaktadır. Bu durum, düzgün bir eğimle sönümlenmesi gereken çınlatıcı çıkışında bozulmalara yol açmaktadır.

Geciktirme süreleri attırıldıkça, çınlatıcı girişine uygulanan ses işareti üzerinde daha büyük hacimli bir fiziki ortama ait bir çınlatma etkisi yaratılmaktadır. Örneğin, Şekil 2.26.a'da gösterilen çınlatıcı yardımıyla geniş bir konser salonu etkisi yaratabilmek için, yapıda yer alan tarak süzgeçlerin geciktirme sürelerinin, en büyük geciktirmenin en küçük geciktirmeye oranı 1,7/1 olacak şekilde, yaklaşık 50 ms değerinde seçilmesi gerekmektedir. Küçük bir oda etkisi yaratabilmek için ise, yaklaşık 10 ms değerinde bir geciktirme yeterli olmaktadır. Yapıda yer alan tüm geçiren süzgeçlerin kullanım amacı, çınlatma süresini uzatmak değil, yalnızca çıkış işaretinin daha fazla sayıda yankı işaretinden oluşmasını sağlamaktır. Bu nedenle, tüm geçiren süzgeçlere 5 ms veya daha düşük değerlerde geciktirme sürelerinin verilmesi yeterli olmaktadır.

2.3.8 İlk ulaşan yansımaların gerçeklenmesi

Bölüm 2.3.5'de açıklandığı gibi, çınlama olayı, dinleyiciye doğrudan ulaşan ses, ilk ulaşan yansımalar ve yoğun çınlama bölgesi olmak üzere üç ana kısımdan oluşmaktadır. Schroeder'in (1961, 1962) tasarımları göz önünde tutularak Bölüm 2.3.7'de tanımlanan çınlatıcı yapılar yoluyla, bu üç ana kısmından yalnızca biri olan yoğun çınlama bölgesi yapay olarak gerçeklenebilmektedir.



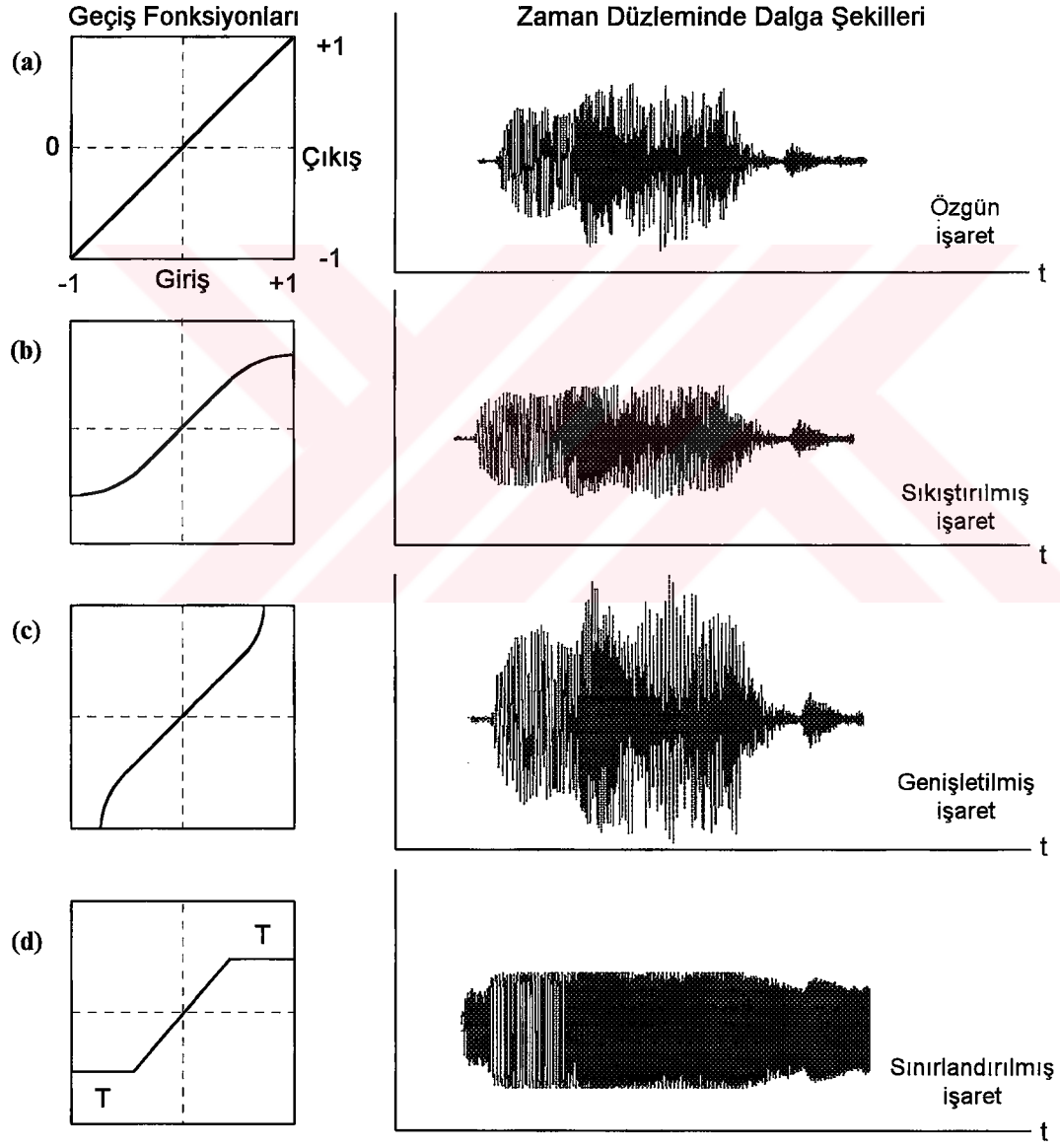
Şekil 2.27 İlk ulaşan yansımaların, çok çıkışlı bir sayısal geciktirici ile gerçeklenmesi.

Shroeder (1970), ilk ulaşan yansımaları da gerçekleyebilmek amacıyla, tasarladığı çınlatıcıya çok çıkışlı bir sayısal geciktirici eklemiştir. Bu sayısal geciktiriciden alınan çıkışlar sayesinde, yoğun çınlama bölgesi başlangıcından hemen önce dinleyiciye ulaşan ayrık yankı işaretleri de üretilebilmektedir.

Şekil 2.27'de, çınlatıcı ile birlikte çok çıkışlı bir sayısal geciktiricinin kullanıldığı tümleşik bir tasarım gösterilmiştir. Daha ileri bir adım olarak, sesin yüzeylere çarpıp yansırken ve havada ilerlerken yüksek frekans bileşenlerinden bir kısmının bastırılması olayını gerçeklemek amacıyla, çınlatıcı çıkışının alçak geçiren bir süzgeçten geçirilmesi de mümkündür.

2.4 Dinamik Değer İşleme

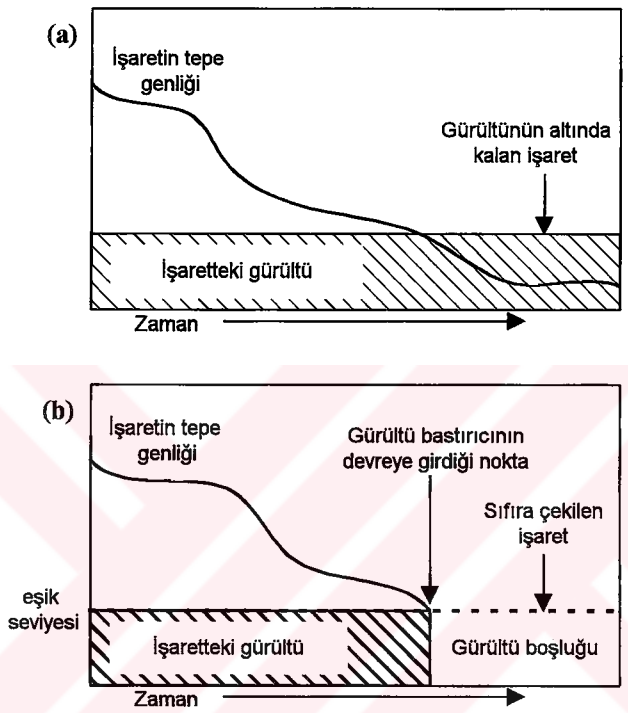
Dinamik değer işlemleri yoluyla, işaretlerin genlikleri üzerinde değişimler oluşturulmaktadır. Gürültü bastırıcılar (noise gates), sıkıştırıcılar (compressors), genişleticiler (expanders) ve sınırlandırıcılar (limiters), bu amaçla kullanılan başlıca yapılardır (McNally, 1984). Dinamik değer işlemlerinin, herhangi bir işaretteki gürültüyü ortadan kaldırmak gibi temel işlevlerinin yanı sıra, müzikal bir işaretin zarfiyla oynarak farklı ses etkileri oluşturmaya yönelik kullanım alanları da bulunmaktadır.



Şekil 2.28 Temel dinamik değer işlemleri.

2.4.1 Gürültü bastırıcılar

Gürültü bastırıcı yapılar, müzikal bir işarette yer alan uğultu veya tıslama türünden sabit değerli gürültü bileşenlerini ortadan kaldırmak amacıyla kulanılmaktadır. Bir gürültü bastırıcının çalışması, girişine uygulanan işaret belirli bir eşik seviyesinin altına düştüğünde çıkışını sıfir değerine çekmek ilkesine dayanmaktadır.



Şekil 2.29 Gürültü bastırıcı bir yapının temel çalışma ilkesi.

Şekil 2.29'da, gürültü bastırıcı bir yapının çalışma mantığı gösterilmiştir. Şekil 2.29.a'da görüldüğü gibi, belirli bir genlik seviyesinin altında, işarete düşük seviyeli bir gürültü eklenmektedir. Şekil 2.29.b'de gösterilen, gürültü bastırıcıdan geçirilmiş işarette ise, genlik belirli bir seviyesinin altına düştüğünde gürültü bastırıcı devreye girmekte ve gürültü ile birlikte işareti de ortadan kaldırarak çıkışı sıfira çekmektedir.

2.4.2 Sıkıştırıcılar ve sınırlandırıcılar

Sıkıştırıcı bir yapı, kazancı giriş işaretinin genliğine göre değişen bir yükseltici (amplifier) olarak düşünülebilir. Belirli bir eşik seviyesinin üzerine çıkıldığında, sıkıştırıcı devreye

girerek işaretin genliğini aşağıya çekmektedir. Sıkıştırıcının giriş işareti zarfında yarattığı değişim, işaret genliğinin anlık değeri veya zaman ortalaması üzerinden gerçekleştirilmektedir. Anlık değer algılayıcılı bir sıkıştırıcı, giriş genliğindeki değişimlere bire bir tepki gösterebilmektedir. Bu sayede, giriş işaretinde oluşabilecek ani sıçramalar karşısında sistemin aşırı yüklenmesi engellenerek etkili bir koruma sağlanmaktadır. Ortalama değer algılayıcı bir sıkıştırıcı ise, girişteki değişimlere genellikle bir kaç saniyelik bir gecikmeyle tepki vermektedir. Bununla birlikte, ortalama değer algılayıcılı bir yapı kullanılarak, ani düşüşler ve keskin geçişlerden arındırılmış, daha yumuşak bir çıkış elde edilmektedir.

Şekil 2.28.b'de, anlık değer algılayıcılı bir sıkıştırıcı yapıya ait geçiş fonksiyonu ve sıkıştırıcının, özgün hali Şekil 2.28.a'da verilen işaret üzerinde yarattığı etki gösterilmiştir.

Sıkıştırma oranı, veya daha genel tanımıyla *giriş/çıkış* oranı, giriş işareti genliğindeki değişimlere karşılık, çıkış işareti genliğinin nasıl değiştirileceğini belirleyen bir büyüklüktür. Örneğin, 4 : 1 sıkıştırma oranına sahip bir yapı, giriş işaretindeki 4 dB'lik bir değişime karşılık, çıkışta yalnızca 1 dB'lik bir değişim oluşturmaktadır. Ses işaretlerine uygulanan 8 : 1 oranından büyük sıkıştırmalar, işaret zarfını düzleştirerek müzikal geçişleri *"kırptığı"* için, işaretin müzikal özellikleri üzerinde köklü değişimler yaratmaktadır.

Sınırlandırıcı, yüksek sıkıştırma oranına sahip bir sıkıştırıcıdır. Sınırlandırı tasarımında, genellikle 10 : 1 değerinden büyük sıkıştırma oranları kullanılmaktadır. Şekil 2.28.d'de, sınırlandırıcı bir yapıya ait geçiş fonksiyonu ve sınırlandırıcının, özgün hali Şekil 2.28.a'da verilen işaret üzerinde yarattığı etki gösterilmiştir. Şekilde görüldüğü gibi, T eşik seviyesine kadar çıkış, girişe göre bire bir doğrusal olarak değişmekte, sınır değerinin üzerindeki genlikler için ise, girişten bağımsız olarak çıkışa sabit T seviyesi gönderilmektedir.

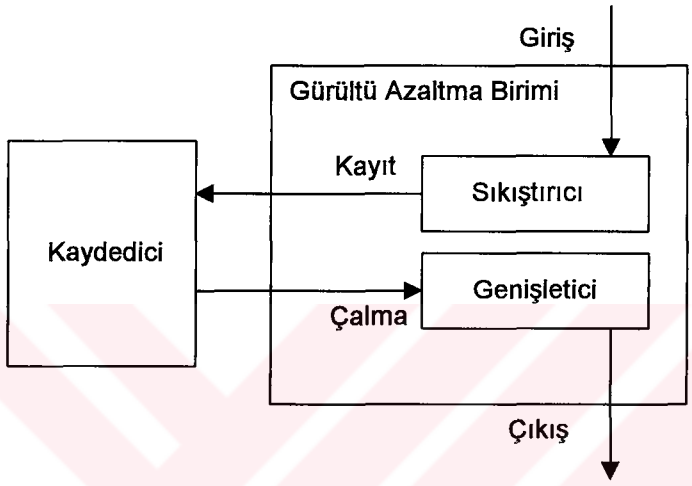
2.4.3 Genişleticiler

Genişleticiler, sıkıştırıcı yapıların tersine, belirli bir eşik seviyesinden sonra giriş geniğindeki küçük değişimlere karşılık çıkışlarında daha büyük değişimler oluşturan yapılardır.

Genişleticiler genellikle gürültü azaltma birimlerinin tasarımında, sıkıştırıcılarla birlikte kullanılmaktadır.

Şekil 2.28.c'de, anlık değer algılayıcılı bir genişletici yapıya ait geçiş fonksiyonu ve genişleticinin, özgün hali Şekil 2.28.a'da verilen işaret üzerinde yarattığı etki gösterilmiştir.

2.4.4 Gürültü azaltma birimleri



Şekil 2.30 Gürültü azaltma birimi.

Şekil 2.30'da gösterildiği gibi, bir gürültü azaltma biriminde genellikle, kaydedilecek giriş işareti bir sıkıştırıcıdan geçirilmekte, kaydediciden alınan işaret ise, genişletildikten sonra çıkışa gönderilmektedir. Sıkıştırma aşamasında, işaretteki ani sıçramalar kırpılmakta ve işaretin geri kalanın genliği arttırılmaktadır. İşaretteki ani atakların ve keskin geçişlerin kırpılması yoluyla, kayıt esnasında genlik taşmaları nedeniyle oluşabilecek gürültünün ve cızırtıların önüne geçilmektedir. İşaretin geri kalanının genliğinin yükseltilme amacı ise, özellikle analog teyp kayıtları veya az sayıda bitle yapılan sayısal kayıtlarda işarete eklenen düşük seviyeli kanal gürültüsünün etkisini ortadan kaldırmaktır. İşaretin genliği, kanal gürültüsünün üzerinde kalacak şekilde attırılarak çıkışa, tıslama veya uğultu gibi sabit seviyeli bir gürültünün mümkün olduğunca az hissedilebilir olduğu bir işaret iletilmeye çalışılmaktadır. Çalma aşamasına gelindiğinde, kaydediciden alınan işaret, çıkışa gönderilmeden önce bir genişleticiden geçirilmekte ve girişteki özgün haline

dönüştürülmektedir. Sonuç olarak sistem çıkışında, görece olarak çok düşük seviyede gürültülü, yüksek dinamik değerde bir çıkış işareti elde edilmektedir.

Dolby laboratuvarlarında geliştiren gürültü azaltma birimleri gibi daha karmaşık bazı yapılarda frekansa bağımlı bir yöntem kullanılmaktadır. Bu yöntem uyarınca, giriş işareti önce temel frekans bileşenlerine ayrılmakta ve daha sonra her bir frekans bileşeni farklı geçiş fonksiyonlarına sahip sıkıştırıcı ve genişleticilerden geçirilmektedir. Giriş işareti frekans bileşenlerine ayrıldığında sıkıştırma işlemine sokulması gerekli olmayan bileşenler belirlenmektedir. Diğer bileşenlerin her biri üzerinde ise, bileşene özel bir sıkıştırma ve genişletme işlemi uygulanmaktadır. Bu sayede, sesin bileşenlerinin tümüyle oynanmamış olması, üzerinde değişim yapılması gerekenlerin ise daha etkin bir biçimde işlenmesi yoluyla, gürültü azaltıcı birim çıkışında daha yüksek kalitede bir ses işareti elde edilmektedir.

3. PENCERELEME, HFD ve KSFD

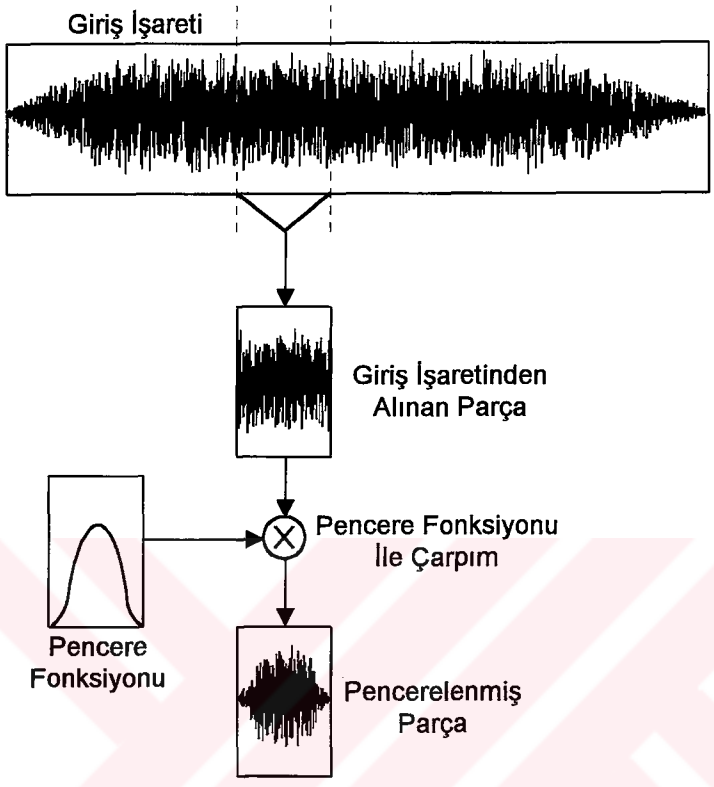
Bilindiği gibi, zamanda sürekli, peryodik işaretlerin sinüzoidal toplamlar cinsinden ifade edilerek frekans bileşenlerine ayrıştırılması işlemi Fourier serisi açınımı yoluyla gerçekleştirilmektedir. Sonlu zaman aralıklı, peryodik olmayan işaretlerin spektral analizi için ise Fourier dönüşümü kullanılmaktadır.

Ayrık zaman işaretlerinin frekans bileşenlerine ayrıştırılması işlemi, AFD (Ayrık Fourier Dönüşümü "Discrete Fourier Transform") olarak adlandırılmaktadır. Uygulamada, ayrık örnek değerlerinden oluşmuş, sayısal bir işaretin spektral analizi için, yüksek hızlı, olabildiğince az bilgisayar işlemiyle gerçeklenebilen algoritmaların kullanılması gerekmektedir. KSFD (Kısa Süreli Fourier Dönüşümü "Short Time Fourier Tranform"), bu amaçla geliştirilmiş bir yöntemdir (Schroeder ve Atal, 1962; Flanagan, 1972; Schafer ve Rabiner, 1973; Allen ve Rabiner, 1977). Sonlu zaman aralıklı, ayrık bir işareti, KSFD yoluyla frekans bileşenlerine ayrıştırma işlemi iki temel adımdan oluşmaktadır. Birinci adımda işaret, pecereleme (windowing) olarak adlandırılan bir işlemden geçirilerek bir dizi, kısa süreli parçaya ayrılmaktadır. İkinci adımda ise, Fourier dönüşümü yoluyla her bir pencerelenmiş parçaya ait frekans bileşenleri ayrı ayrı hesaplanmaktadır. Fourier dönüşümü hesabında, geleneksel AFD işlemine göre daha etkin ve daha hızlı bir yöntem olan HFD (Hızlı Fourier Dönüşümü "Fast Fourier Transform") kullanılmaktadır.

3.1 Pencereleme

Ayrık zamanlı, sayısal bir işaret Fourier dönüşümü alınmadan önce pencereleme işleminden geçirilmektedir. Pencereleme yoluyla, işaret özel bir pencere fonksiyonu ile çarpılarak bir dizi kısa süreli parçaya ayrılmaktadır. Pencere fonksiyonu, sınırlı süreli bir zaman aralığında tanımlanmış, sıfırdan farklı değerler alan özel bir matematik fonksiyondur. Pencere fonksiyonu, zaman ekseninde giriş işareti boyunca ötelenmektedir. Giriş işareti ile pencere fonksiyonunun çarpılması yoluyla, her birinin süresi pencere fonksiyonunun tanımlandığı sınırlı süreli zaman aralığına eşit, bir dizi farklı pencerelenmiş parça oluşturulmaktadır. Her bir pencerelenmiş parçanın ayrı ayrı Fourier dönüşümlerinin alınması sonucunda bulunan

değerler bir araya getirildiğinde, ana işaretin zamana bağımlı frekans özellikleri elde edilmektedir. Bir başka deyişle KSFD işlemi, işarete ait frekans bileşenlerinin zamanda değişimini göstermektedir.



Şekil 3.1 Pencereleme işlemi.

Fourier dönüşümü, işaretin tümü yerine her bir pencerelenmiş parça için ayrı ayrı hesaplandığından, pencereleme yoluyla spektral analiz için gerekli işlem miktarı önemli ölçüde azaltılmaktadır. Bununla birlikte, her bir pencerelenmiş parça, giriş işareti ile pencere fonksiyonunun çarpımından elde edildiği için, Fourier dönüşümü yoluyla hesaplanan frekans bileşenleri yalnızca giriş işaretine değil, giriş işareti ile pencere fonksiyonunun konvolüsyonuna ait değerler olmaktadır. Bu durum, spektral analizde belirli bir hataya yol açmaktadır.

KSFD işleminde, farklı türden pencere fonksiyonları kullanılabilir. Pencere uzunluğu (boyutu) ve pencere şekli, bir pencere fonksiyonunu tanımlayan iki temel özelliktir.

3.1.1 Pencere uzunluğu

Pencere uzunluğu, pencere fonksiyonunun zamanda tanımlı olduğu aralığı ve dolayısıyla, giriş işaretinden alınan her bir pencerelenmiş parçanın boyutunu belirlemektedir. Spektral analizi yapılacak pencerelenmiş parçanın boyutu, analizin frekans çözünürlüğünü doğrudan etkilemektedir. KSFD işlemi, her bir pencerelenmiş parça için, giriş işareti üzerine eşit frekans aralıklarıyla yerleştirilmiş bir dizi süzgecin çıkışını incelemek gibi düşünülebilir. Pencerelenmiş işarete ait frekans bileşenleri analizinin, KSFD işlemi kullanılarak hangi frekans değerlerinde yapılacağı, işaretin örnekleme hızına ve pencere uzunluğuna göre belirlenmekte ve,

$$analiz\ aralığı = \frac{örnekleme\ hızı}{pencere\ uzunluğu} \quad [Hz/Örnek\ Değeri] \tag{3.1}$$

bağıntısıyla hesaplanmaktadır. Örnek olarak, 50 KHz örnekleme hızı ve 1000 örneklik bir pencere uzunluğu için analiz aralığı, $50000/1000 = 50$ Hz olmaktadır. Bir başka deyişle, KSFD işleminin analiz frekansları, pencerelenmiş işaret üzerinde 0 Hz'den başlayarak 50 Hz'lik aralıklarla konumlanmaktadır.

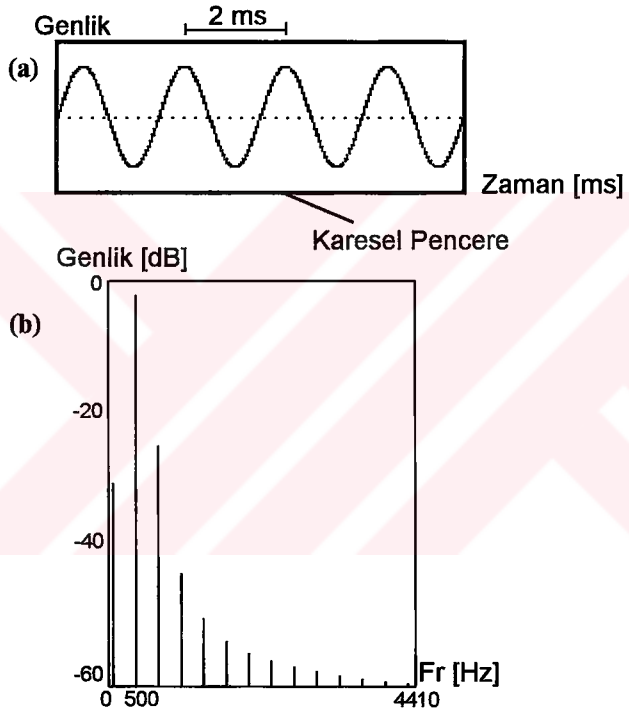
3.1.2 Pencere şekli

Bölüm 3.1'de açıklandığı gibi, her bir pencerelenmiş parça, giriş işareti ile pencere fonksiyonunun çarpımından elde edilmektedir. Bu durum, KSFD yoluyla hesaplanan frekans bileşenlerinin yalnızca giriş işaretine değil, giriş işareti ile pencere fonksiyonunun konvolüsyonuna ait değerler olarak bulunmasına yol açmaktadır. Bu nedenle, kullanılan pencerenin şekli, giriş işaretine ait frekans bileşenlerinin analizinde önemli ölçüde belirleyici olmaktadır.

Karesel pencereleme, spektral analizde kullanılabilecek en basit pencereleme yöntemidir. Bu tür bir pencereleme işleminde, giriş işareti bir kapı fonksiyonuyla çarpılmakta ve işaretin

belirli bir zaman aralığındaki değerleri olduğu gibi alınırken, bu aralık dışındaki tüm değerleri sıfıra çekilmektedir.

İdeal olarak, tek bir sinüs dalgasının frekans spektrumu HFD yoluyla analiz edildiğinde, sonucun dalga frekansına eşit frekansta tek bir dürtü olarak bulunması gerekmektedir. Bir başka deyişle, tüm enerjinin dalga frekansı üzerinde toplanması beklenmektedir. Pencerelenmiş bir sinüs dalgasının spektrumu ise, pencereleme işleminin spektral analizde yarattığı bozulma nedeniyle, dalga frekansında bir dürtüye ek olarak ana kulakçığın (lobe) her iki yanında konumlanan yan kulakçıkları da içermektedir (Jaffe, 1978; Harris, 1978).

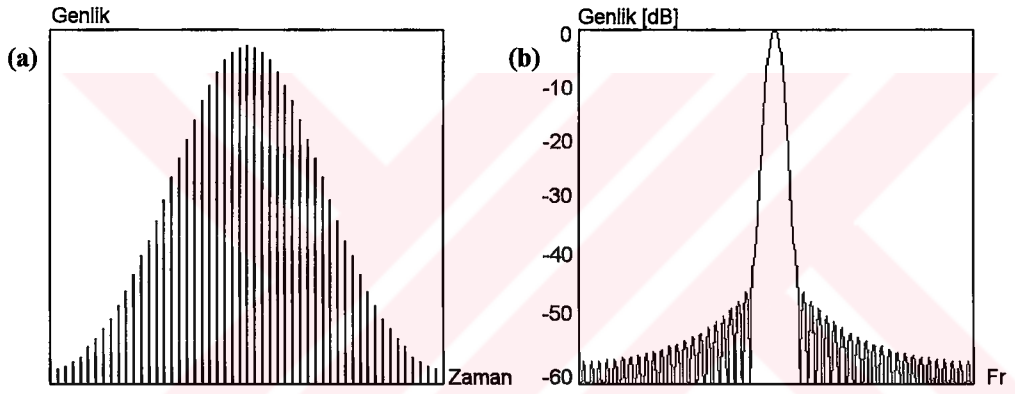


Şekil 3.2 (a) Karesel pencerelenmiş sinüs dalgası; (b) pencerelenmiş parçanın frekans spektrumu.

Şekil 3.2'de, karesel pencerelenmiş bir sinüs dalgası ve pencerelenmiş işarete ait frekans spektrumu gösterilmiştir. Şekil 3.2.a'da görüldüğü gibi, 500 Hz frekanslı bir sinüs dalgasının 4 peryotluk bir parçası pencerelenerek HFD işlemine sokulmaktadır. Pencerelenmiş işaretin frekans spektrumu Şekil 3.2.b'de gösterilmiştir. Karesel pencereleme nedeniyle, spektrumda bozulmalar meydana gelmekte ve 500 Hz frekansındaki dürtüye ek olarak, yaklaşık 0 Hz'den 4400 Hz'e uzanan bir aralıkta yan kulakçıklar oluşmaktadır.

KSFD işleminde kullanılan standart pencerelerin tümü simetrik yapıdadır ve hiç biri karesel özellik taşımamaktadır. Standart bir pencerenin frekans spektrumu, ana bir frekans kulakçığı ve her iki yanında uzanan bir dizi yan kulakçıktan oluşmaktadır ve *sinc* fonksiyonu ($\sin(t)/t$) ile ifade edilebilir niteliktedir. Ana kulakçık genişliği ve en yüksek yan kulakçık seviyesi, standart bir pencereyi tanımlamakta kullanılan iki temel özelliktir. Yan kulakçıkların genliği pencere şekline bağlı olmakla birlikte, pencere uzunluğundan bağımsızdır.

Şekil 3.3'de, en sık kullanılan standart pencere türlerinden biri olan Gauss penceresi ve pencereye ait frekans spektrumu gösterilmiştir. Şekil 3.3.b'de görüldüğü gibi, en yüksek yan kulakçık seviyesi ana kulakçığın yaklaşık 45 dB aşağısındadır.



Şekil 3.3 (a) Gauss penceresi; (b) pencereye ait frekans spektrumu.

Standart bir KSFD penceresi ile giriş işaretinin çarpımından elde edilen pencerelenmiş parçanın frekans spektrumu, karesel pencere kullanılarak ele edilen spektruma göre yan kulakçıklarda daha sınırlı miktarda bir *"sızıntı"* içermektedir. Bununla birlikte, pencere ile giriş işaretinin konvolüsyonu nedeniyle, frekans spektrumunda yan kulakçıkların oluşması kaçınılmazdır. Pencereleme yoluyla elde edilen frekans spektrumunda, yan kulakçıkların ana kulakçık frekans bileşenlerini bastıracak değerde yüksek genlikli frekans bileşenleri içermesi, giriş işaretine ait frekans bileşenlerinin belirlenmesini engelleyerek, spektral analizde yanıltıcı sonuçların alınmasına yol açabilmektedir.

Standart bir pencerenin, giriş işareti frekans spektrumunda yan kulakçık sızıntısı nedeniyle

oluşturduğu bozulmanın derecesi, en yüksek yan kulakçık seviyesine ve yan kulakçıkların sönümlenme oranına göre belirlenebilmektedir. Çizelge 3.1'de, standart pencere türlerine ait yan kulakçık karakteristikleri verilmiştir (Harris, 1978; Nuttal, 1981). Gauss ve Kaiser pencerelerinde, a katsayısının değeri arttırıldıkça ana kulakçık frekans bölgesi genişlemekte ve dolayısıyla yan kulakçıklar daha dar bir frekans aralığında yer almaktadır.

Çizelge 3.1 Standart pencere türlerine ait yan kulakçık karakteristikleri.

Pencere Türü	En yüksek yan kulakçık seviyesi (dB)	Yan kulakçık sönüm oranı (dB)
Dikdörtgen	-13	-6
Hamming	-43	-6
Hanning	-31	-18
Gauss		
a = 2.5	-42	-6
a = 3.5	-69	-6
Blackman	-58	-18
Kaiser		
a = 2	-40	-6
a = 3	-65	-6
a = 4	-91	-6

3.1.3 Pencere seçimi

Pencere spektrumu ile giriş işareti spektrumunun konvolüsyonu sonucu, Fourier dönüşümü kullanılarak elde edilen spektrumdaki en dar frekans aralığının genişliği, giriş işaretinden bağımsız olarak yalnızca pencere ana kulakçığının frekans genişliğine göre değişmektedir. Bir başka deyişle, HFD yoluyla elde edilen spektrumda, analizi yapılan işarete ait bileşenlerin yer aldığı frekans bölgesinin genişliği, pencere ana kulakçığının frekans aralığına bağlı olarak belirlenmektedir. Bu nedenle, pencere ana kulakçığı frekans genişliğinin arttırılması, spektral analiz gözlem süresinde bir artışa yol açmakta ve analiz aralığındaki bu artış analiz çözünürlüğünün düşmesine sebep olmaktadır.

Özet olarak, yan kulakçıklara ait frekans genişliğinin azaltılması analiz sonucu elde edilen spektrumdaki bozulmaların önüne geçmekle birlikte, ana kulakçık frekans aralığının genişlemesine yol açtığından, analiz çözünürlüğünde bir düşüşü de beraberinde getirmektedir.

Bu nedenle, KSFD işleminde kullanılacak pencere türü seçiminde, yan ve ana kulakçık frekans genişliklerinin yol açacağı sonuçların bir arada düşünülmesi gerekmektedir. Genellikle, belirli sayıda sinüzoidal bileşenden oluşan işaretlerin analizinde, dar bir ana kulakçığa sahip pencere fonksiyonları tercih edilmektedir. Gürültülü işaretler gibi, enerjinin belirli frekanslar üzerinde yoğunlaşmadığı işaretlerin analizi için ise, yan kulakçıkları olabildiğince dar bir frekans bölgesini kapsayan pencere fonksiyonları kullanılmaktadır.

3.2 Hızlı Fourier Dönüşümü

Ayrık zaman işaretleri spektral analizinin AFD yoluyla gerçeklenmesi, oldukça uzun bir bilgisayar işlem süresi gerektirmektedir. N sayıda $(0 \le n \le N-1)$ örnek değerinden oluşan bir $x[n]$ ayrık işaretine ait N adet $(0 \le k \le N-1)$ frekans bileşeni, AFD kullanılarak,

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{nk} \qquad k = 0,1,2,\ldots N-1 \tag{3.2}$$

bağıntısıyla hesaplanmaktadır. Eşitlikte yer alan W_N terimi,

$$W_N = e^{-j(2\pi/N)} \tag{3.3}$$

değerine eşit karmaşık üstel bir büyüklüktür.

Bir başka deyişle, frekans bileşenlerine ait N uzunluklu $X[k]$ vektörü, giriş işareti örnek değerlerinden oluşan N uzunluklu $x[n]$ vektörü ile W matrisinin çarpımından elde

edilmektedir. $X[k]$ vektörünün her bir elemanı karmaşık bir büyüklük olup, ilgili frekans bileşeninin genlik ve faz bilgisini içermektedir. W matrisi, $[N \times N]$ boyutlu,

$$\mathbf{W} = \begin{bmatrix} W_N^0 & W_N^0 & W_N^0 & \cdots & W_N^0 \\ W_N^0 & W_N^1 & W_N^2 & \cdots & W_N^{(N-1)} \\ W_N^0 & W_N^2 & W_N^4 & \cdots & W_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \cdots & \\ W_N^0 & W_N^{(N-1)} & W_N^{2(N-1)} & \cdots & W_N^{(N-1)(N-1)} \end{bmatrix} \tag{3.4}$$

eşitliği ile ifade edilen bir kare matristir. W matrisine ait her bir *n.* satır, *k.* sütun elemanı, $W_N^{n.k}$ değerine eşittir ve döndürme katsayısı veya faz katsayısı olarak adlandırılmaktadır (Rabiner vd., 1972).

AFD hesabı, yaklaşık N^2 adet karmaşık toplama ve çarpma işlemi gerçekleştirilerek sonuçlandırılmaktadır. Buna ek olarak, $W_N^{n.k}$ döndürme katsayılarının kuvvetlerinin elde edilmesi için, her bir karmaşık çarpım başına dört adet gerçel çarpma işleminin,

$$(A + jB) \times (C + jD) = [(A \times C) - (B \times D)] + j.[(A \times D) + (B \times C)] \tag{3.5}$$

şeklinde yapılması gerekmektedir.

Spektral analizi yapılacak işaretin az sayıda örnekten oluştuğu durumlarda, AFD hesabının gerektirdiği N^2 sayıda karmaşık çarpma ve toplama işleminin bilgisayar ortamında gerçeklenmesinde herhangi bir sorunla karşılaşılmamaktadır. Bununla birlikte, özellikle müzikal işaret işleme uygulamalarında, çok sayıda kanal üzerinden, yüksek hızlarda örneklenerek, geniş boyutlarda pencerelenmiş işaretlerin gerçek zaman spektral analizine ihtiyaç duyulmaktadır. Örneğin, 48 KHz hızda örneklenmiş, çiftli (stereo) bir giriş işaretinden, saniyede 50 pencere alınarak gerçekleştirilen bir AFD hesabı, saniye başına 66 milyondan fazla çarpma işlemi yapılarak sonuçlandırılabilmektedir. Bu durum, AFD işleminde bir iyileştirmenin gerekliliğini açıkça ortaya koymaktadır.

W matrisini oluşturan $W_N^{n.k}$ dönüştürme katsayılarının tümü, birbirinden farklı değerler almamaktadır. Bu özellik, W matrisinde büyük oranda bir indirgeme yapma olanağı vermekte ve HFD işleminin en önemli dayanak noktalarından birini oluşturmaktadır. N uzunluklu bir ayrık işaretin spektral analizi, HFD yoluyla, N^2 sayıda adım yerine, $N\log_2(N)$ adımda sonuçlandırılabilmekte ve işarete ait frekans bileşenleri, çok daha kısa süreli bir bilgisayar işlemi ile bulunabilmektedir (Cooley ve Tukey, 1965).

3.2.1 İki tabanına göre HFD işlemleri

HFD algoritmasının işleyiş mantığı spektral analizi yapılacak işaretin uzunluğunun ikinin kuvveti bir değerde olmasını gerektirmektedir. Uygulamalarda, bu şartı sağlamayan işaretler, sıfır ekleme (zero padding) yoluyla, ikinin kuvveti, en yakın bir değere genişletilerek uygun uzunluğa getirilmektedir.

HFD işleminin ana ilkesi, N noktalı (N uzunluklu) giriş işaretini bir dizi olarak ele alıp, tek sıra numaralı elamanlar bir yarıyı, çift sıra numaralı elemanlar ise diğer yarıyı oluşturacak biçimde, $N/2$ noktalı iki diziye bölmeye dayanmaktadır. Bu ikiye bölme işlemi, ana dizi iki noktalı dizilere ayrıştırılana dek yinelemeli bir biçimde sürdürülmekte ve iki noktalı dizilerin AFD işlemine sokulmasıyla alınan sonuçlar bir araya getirilerek ana diziye ait frekans bileşenleri elde edilmektedir. İki noktalı AFD hesabı karmaşık çarpma işlemi içermemekte, toplama ve çıkarma işlemleri kullanılarak gerçeklenebilmektedir. Çarpma işlemine yalnızca, ikiden fazla noktalı AFD işlemlerinde, döndürme katsayısının hesaba katılması nedeniyle gereksinim duyulmaktadır. İkiye bölme işlemi, $\log_2(N)$ aşamada gerçekleştiği için, N noktalı giriş dizisine ait AFD hesabında, yaklaşık $(N/2)\log_2(N)$ sayıda karmaşık çarpma işlemi yapılmaktadır.

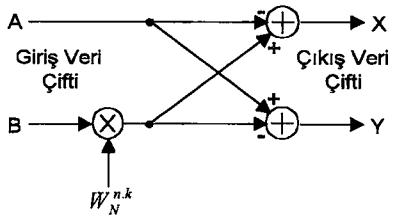
3.2.2 Kelebek yapı

Kelebek yapı, iki tabanlı AFD işlemleri yoluyla N noktalı bir giriş dizisine ait frekans bileşenlerini hesaplamak amacıyla tasarlanmıştır. Şekil 3.4'de görüldüğü gibi, kelebek yapı

bir çift giriş ve çıkışa sahiptir. A ve B girişleri kullanılarak, X ve Y çıkışları,

$$X = A + W_N^{nk} B$$
$$Y = A - W_N^{nk} B \tag{3.6}$$

işlemleri yoluyla elde edilmektedir. Çıkışların hesaplanmasında, X çıkışı bulunurken kullanılan $W_N^{nk} B$ çarpımı, Y çıkışı için de kullanıldığından, yalnızca bir adet çarpma işlemi ile sonuca varılabilmektedir.



Şekil 3.4 Kelebek yapı.

Bilgisayar ortamında, N noktalı bir giriş dizisi için bellekte bir alan tahsis edilmekte ve AFD işlemi aşamalarında kelebek yapı çıkışlarından elde edilen değerler, gene aynı bellek bölgesinin ilgili sıra numaralı gözlerine yazılmaktadır. Kelebek yapı sayesinde, N elemanlı bellek bölgesindeki değiş tokuş işlemleri için yalnızca bir adet ek bellek gözünün kullanımı yeterli olmaktadır.

3.2.3 Bit çevrimi sıralaması

HFD işlemi sonucunda, $X[k]$ çıkış dizisinin, $k = 0,1,2,\ldots N-1$ değerlerini alacak şekilde, doğal bir sıralama ile elde edilmesi gerekmektedir. Böyle bir sonucun alınabilmesi için, kelebek yapı kullanılarak iki noktalı AFD işlemlerinin gerçekleştirilmesine başlanmadan önce $x[n]$ giriş dizisi sıralaması bit çevrimi yoluyla değiştirilmektedir. Bit çevrimi ile, N noktalı giriş dizisi, çift sıra numaralı elemanlı bir tarafta, tek sıra numaralı elemanları ise diğer tarafta toplanacak şekilde düzenlenmektedir. Bu düzenleme, Çizelge 3.2'de gösterildiği gibi, sıra

numaralarının ikili düzendeki karşılıkları üzerinde yapılan bit çevrimleri yoluyla gerçekleştirilmektedir.

Çizelge 3.2 Bit çevrimi ile sıralama.

Sıra No	İkilik Düzen	Bit Çevrimi	Yeni Sıra No
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

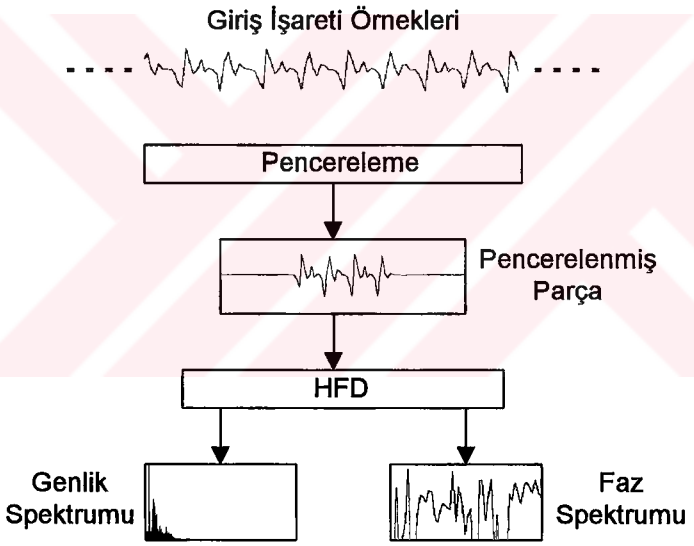
Şekil 3.5'de, HFD algoritmasının temel yapısı gösterilmiştir. *N* elemanlı giriş işaretini iki noktalı dizilere ayrıştırma ve ardışıl bir biçimde, ikinin kuvveti dereceli AFD hesaplama aşamalarında yinelemeli bir algoritma kullanılarak HFD işlemi oldukça az sayıda satırdan oluşan bir bilgisayar programı aracılığıyla gerçeklenebilmektedir.

Şekil 3.5 HFD algoritmasının temel yapısı.

3.3 Kısa Süreli Fourier Dönüşümü

Şekil 3.6'da gösterildiği gibi, KSFD işleminden geçirilen ayrık zaman giriş işareti, ilk adımda pencereleme yoluyla kısa süreli parçalara ayrılmaktadır. İkinci adımda ise, her bir pencerelenmiş parçanın spektral analizi HFD kullanılarak ayrı ayrı yapılmakta ve alınan sonuçlar bir araya getirilerek ana işaret frekans bileşenlerinin zamanda değişimi elde edilmektedir.

Pencerelenmiş parçaların ayrı ayrı HFD işlemine sokulmasıyla elde edilen sonuçlar ardışıl bir biçimde sıralandıklarında, ana işaret frekans bileşenlerinin zamanda değişimini gösterdikleri için, HFD işleminden alınan veri paketleri çerçeve (frame) olarak adlandırılmaktadır. Her bir çerçeve, analizden geçmiş ilgili frekans bileşenlerine ait genlik ve faz bilgilerini içermektedir.



Şekil 3.6 KSFD işlemi.

4. GÖRSEL SES İŞLEYİCİSİ

Görsel ses işleyicisi (GSİ), ses dosyaları üzerinde çeşitli işlemler yapabilen, uygulama amaçlı tasarlanmış bir yazılımdır. Yazılım Microsoft Windows ortamında çalışmakta ve PCM (Vuru Kod Modülasyonu "Pulse Code Modulation") biçimli, Microsoft Windows Ses Dalgası dosyalarını (PCM Wave File Format) işleyebilmektedir (Ek 1). GSİ, herhangi bir görsel yazılımda bulunması gereken kesme, yapıştırma, kopyalama, kaydetme, farklı kaydetme, geri alma ve seçilen bir bölgeyi işleme gibi standart özellikleri desteklemektedir. GSİ yazılımında kullanılan temel yapı ve algoritmalara ait program dökümleri Ek 3'de verilmiştir. Yazılımın temel işlevleri, beş ana başlık altında toplanabilir:

- Dinamik değer işlemleri
- Geciktirme etkileri
- Çınlatma
- Süzgeçleme
- Gösterim

Dinamik değer işlemleri, Bölüm 2.4'de anlatıldığı gibi, işaretlerin zarfları üzerinde değişimler oluşturmaktadır. GSİ kullanılarak, sönüm (fade in, fade out), gürültü bastırma, sıkıştırma, genişletme ve kırpma (distortion) işlemleri gerçeklenebilmektedir.

Geciktirme etkileri (Bölüm 2.2), çevrimsel yapıda, çok çıkışlı bir değişken zamanlı sayısal geciktirici kullanılarak tasarlanmıştır. Sayısal geciktiricinin her bir çıkışına ait geciktirme süreleri yazılımsal olarak değiştirilebilmektedir. Titreşim (vibrato), dalgalandırma, koro, çiftleme ve yankı etkileri, genel amaçlı bir yapı yoluyla gerçeklenmektedir. Genel amaçlı yapı, çok çıkışlı bir sayısal geciktirici içermekte ve sınırlı sayıda temel parametre tarafından denetlenmektedir. Parametrelere değerleri değiştirilerek, farklı türde geciktirme etkilerinin oluşturulması sağlanmaktadır.

Çınlatma etkisi yaratabilmek için, tümleşik bir sayısal yapı kullanılmıştır. Yoğun çınlama bölgesi (Bölüm 2.3.5), tarak süzgeçler ve tüm geçiren süzgeçlerden oluşan bir çınlatıcı

(Bölüm 2.3.7) yoluyla gerçeklenmektedir. İlk ulaşan yansımaların (Bölüm 2.3.8) gerçeklenmesinde ise, çok çıkışlı bir sayısal geciktiriciden yararlanılmıştır. Uygulama aşamasında, Schroeder'in (1961, 1962, 1970) özgün tasarımları temel alınmıştır (Bölüm 2.3.7, Şekil 2.26.a). Bununla birlikte, çınlatıcı yapı, tarak süzgeçlerin tasarımı ve geciktirici kullanımları açısından Schroeder'in (1970) çınlatıcısına göre farklı özellikler içermektedir. Yapılan geliştirmeler daha etkin bir çınlatma etkisinin elde edilebilmesini sağlamaktadır.

GSİ yazılımında, süzgeçleme işlemi için iki farklı yinelemeli süzgeç (Bölüm 2.1.5) seçeneği bulunmaktadır. Süzgeçler, belirli sayıda parametre ile denetlenen genel amaçlı yapılar biçimindedir. Tasarım için kapalı form bağıntılardan yararlanılmıştır. Yapıların frekans yanıtları parametrelere atanan değerler yoluyla değiştirilebilmektedir. Bu sayede, tek bir genel amaçlı yapı kullanılarak farklı karakteristikte süzgeçleme işlemleri gerçeklenebilmektedir.

GSİ işlediği veriyi iki farklı biçimde gösterebilmektedir. Zamanda gösterim, işareti oluşturan örnek değerlerinin zaman ekseninde değişimini görüntülemektedir. Spektral gösterimde ise, işaret KSFD (Bölüm 3.3) işleminden geçirilerek frekans bileşenlerinin zamanda değişimi bulunmaktadır. KSFD işlemi sonucunda, her bir pencerelenmiş parçaya (Bölüm 3.1) ait frekans bileşenleri ardışıl bir biçimde yanyana getirilerek üç boyutlu bir zaman- frekans-genlik gösterimi elde edilmektedir. Frekans bileşenlerin genlik değerleri renk değişimleri yoluyla ifade edilmektedir.

GSİ yazılımı, farklı özelliklerle kaydedilmiş PCM Ses Dalgası dosyalarını işleyebilmek amacıyla, esnek bir biçimde tasarlanmıştır. Dosyadaki veriye, hangi örnekleme hızında, kaç kanaldan ve kaç bit üzerinden kaydedilirse kaydedilsin, yazılım tarafından ulaşılabilmektedir. Dosyadan okunan işarete ait örnek değerleri, yazılımı oluşturan tüm birimlerin ortak kullanıma açık bir bellek bölgesine yerleştirilmektedir.

GSİ yazılımının içerdiği tüm yapılar, kullanıcı tarafından görsel biçimde denetlenebilir nitelikte tasarlanmıştır. Her bir yapı, varsayılan (default) değerlerin yanı sıra, kullanıcı tanımlı olarak da şartlandırılabilmektedir.

4.1 Dinamik Değer İşlemleri

4.1.1 Sönüm

GSİ yazılımı yoluyla, giriş işareti üzerinde iki farklı biçimde (doğrusal ve üstel) sönüm etkisi oluşturulabilmektedir. Sönüm işleminin başlangıç ve bitiş değerleri, kullanıcı tarafından, görsel kaydırma çubukları aracılığıyla belirlenebilmektedir.

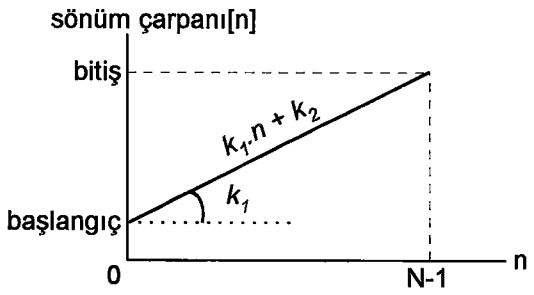
Sönüm işlemini gerçekleyen yapı, N sayıda örnek değerinden oluşmuş bir $x[n]$ sayısal giriş işareti ($n = 0,1,2,\ldots,N-1$) için;

$$y[n] = sönüm\ çarpanı[n].x[n] \qquad n = 0,1,2,\ldots,N-1 \tag{4.1}$$

çıkışını üretmektedir. Sönüm çarpanı, sönüm işleminin karakteristiğine, başlangıç ve bitiş değerlerine ve giriş işaretinin örnek sayısına bağlı olarak bulunmaktadır.

4.1.1.1 Doğrusal sönüm

Şekil 4.1'de, doğrusal sönüm çarpanının giriş işareti uzunluğuna göre değişimi gösterilmiştir. Giriş işareti $x[n]$, N adet örnekten oluşmaktadır.



Şekil 4.1 Doğrusal sönüm çarpanının giriş işareti uzunluğuna bağlı olarak değişimi.

Şekil 4.1 uyarınca, doğrusal sönüm çarpanı,

$$sönüm\ çarpanı[n] = \begin{cases} k_1 n + k_2 & ; 0 \le n \le N-1 \\ 0 & ; diğer \end{cases} \tag{4.2}$$

eşitliğiyle hesaplanmaktadır. (4.2) eşitliğindeki k_1 ve k_2 değerleri;

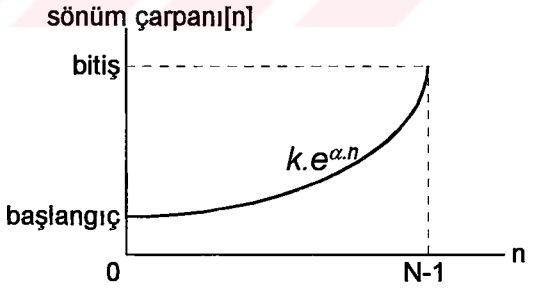
$$k_1 = \frac{bitiş - başlangıç}{N}$$
$$k_2 = başlangıç \tag{4.3}$$

bağıntılarıyla belirlenmektedir.

Sonuç olarak, sönümlendirici çıkışında, N sayıda $(n = 0,1,2,\ldots,N-1)$ örnek değerinden oluşan $y[n]$ işareti, (4.1)'de verilen bağıntı yoluyla üretilmektedir.

4.1.1.2 Üstel sönüm

Şekil 4.2'de, üstel sönüm çarpanının giriş işareti uzunluğuna göre değişimi gösterilmiştir. Giriş işareti $x[n]$, N adet örnekten oluşmaktadır.

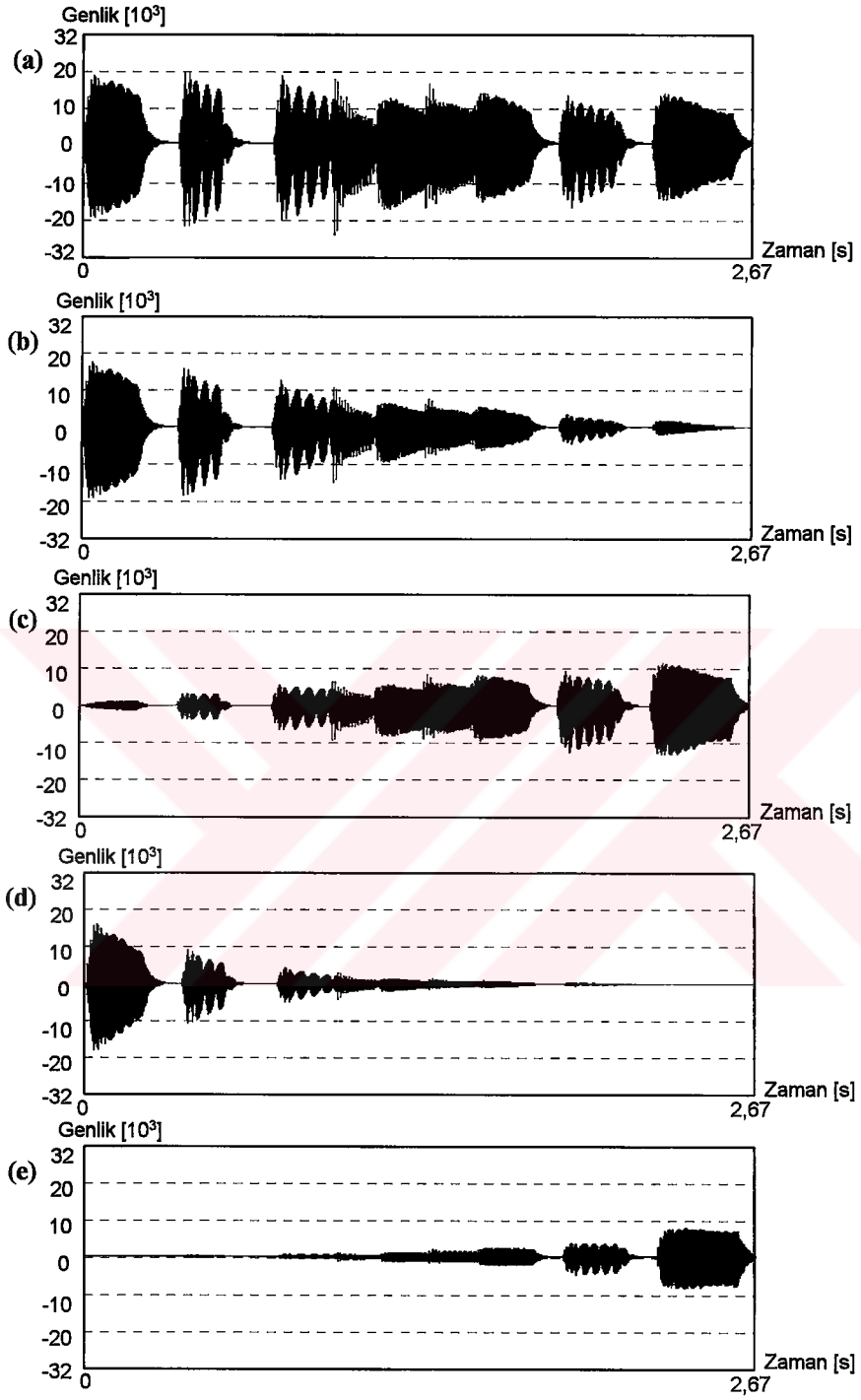


Şekil 4.2 Üstel sönüm çarpanının giriş işareti uzunluğuna bağlı olarak değişimi.

Şekil 4.2 uyarınca, üstel sönüm çarpanı,

$$sönüm\ çarpanı[n] = \begin{cases} k.e^{\alpha n} & ; 0 \le n \le N-1 \\ 0 & ; diğer \end{cases} \tag{4.4}$$

eşitliğiyle hesaplanmaktadır.



Şekil 4.3 Farklı özellikte sönümlendirici çıkışları: (a) özgün işaret; (b) pozitif doğrusal sönüm; (c) negatif doğrusal sönüm; (d) pozitif üstel sönüm; (e) negatif üstel sönüm.

(4.4) eşitliğindeki k ve α değerleri;

$$k = başlangıç$$
$$\alpha = \frac{\ln(bitiş/başlangıç)}{N} \tag{4.5}$$

bağıntılarıyla belirlenmektedir.

Sonuç olarak, sönümlendirici çıkışında, N sayıda $(n = 0,1,2,\ldots,N-1)$ örnek değerinden oluşan $y[n]$ işareti, (4.1)'de verilen bağıntı yoluyla üretilmektedir. Şekil 4.3'de, doğrusal ve üstel karakteristikler için, GSİ yazılımı yoluyla elde edilmiş pozitif ve negatif sönümlü işaretler gösterilmiştir.

4.1.2 Gürültü bastırıcı

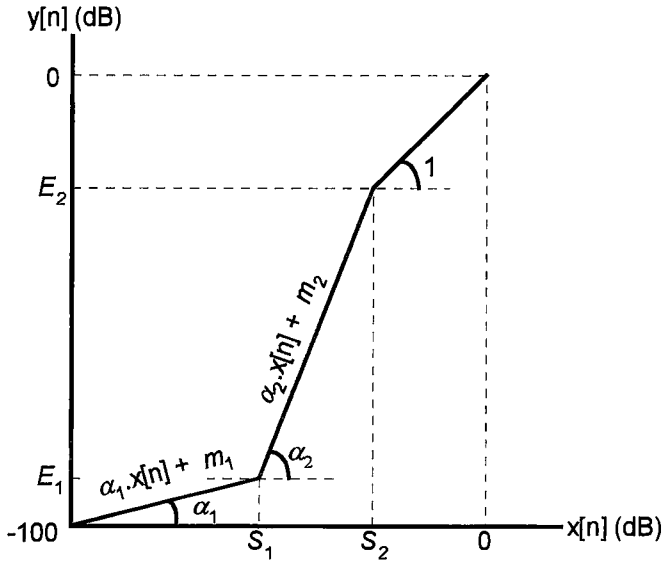
GSİ yazılımında, gürültü bastırma işlemi için, kareköksel ortalama değer (KOD) algılayıcı bir yapı kullanılmaktadır (Bölüm 2.4.1). KOD algılayıcı yapı sayesinde, ani düşüşler ve kesikliklerden arındırılmış, yumuşak geçişli bir çıkış işareti elde edilmektedir. Gürültü bastırma işlemi kullanıcı tanımlı bir biçimde, GSİ üzerinde bulunan *"bastırma oranı"*, *"alt sınır"* ve *"üst sınır"* isimli üç kaydırma çubuğu aracılığıyla gerçekleştirilmektedir. Şekil 4.4'de, gürültü bastırıcıya ait geçiş fonksiyonu gösterilmiştir. Geçiş fonksiyonunda, $x[n]$ giriş ve $y[n]$ çıkış işaretlerine ait örnek değerleri, desibel cinsinden;

$$örnek\ değeri\ [dB] = 20\log_{10}(örnek\ değeri/V_M) \tag{4.6}$$

şeklinde ifade edilmektedir. Giriş veya çıkış işaretlerinin olası en büyük değeri V_M olarak tanımlanmıştır. Bu değer, işaretleri oluşturan örneklerin kaçar bit uzunluklu olduğuna bağlı olarak belirlenmektedir (Ek 1). (4.6)'dan açıkça görüldüğü gibi, V_M desibel cinsinden sıfıra eşit olmaktadır.

Geçiş fonksiyonunda *alt sınır* ve *üst sınır* değerlerinin desibel cinsinden karşılıkları, sırasıyla, S_1 ve S_2 ile gösterilmiştir. S_1 ile belirlenen seviye, düşük seviyeli bir gürültünün sınır değeri olarak algılanmakta ve bu seviyenin altında kalan giriş işareti örnek değerleri

bastırılmaktadır. S_1 ile S_2 seviyesi arasında ise, örnek değerleri genişletilerek gürültü eşiğinden uzaklaştırılmaktadır.



Şekil 4.4 Gürültü bastırıcı geçiş fonksiyonu.

Geçiş fonksiyonu karakterisiği;

$$
\begin{aligned}
&\alpha_1 = \textit{bastırma oranı} \\
&m_1 = -100(1-\alpha_1) \\
&E_1 = \alpha_1 S_1 + m_1 \\
&E_2 = S_2 \\
&\alpha_2 = (E_2 - E_1)/(S_2 - S_1) \\
&m_2 = S_2(1-\alpha_2)
\end{aligned}
\qquad (4.7)
$$

değerleri ile belirlenmektedir.

Bölüm 4'de anlatıldığı gibi, N uzunluklu $x[n]$ giriş işareti $(n = 0,1,2,\ldots,N-1)$, GSİ yazılımında yer alan tüm birimlerin kullanımına açık bir bellek bölgesine yerleştirilmiş durumdadır. Gürültü bastırma işlemi sırasında, bellekteki bu N uzunluklu diziden M uzunluklu ($M < N$) paketler alınmakta ve her bir pakete ait KOD değeri;

$$KOD[p]=\left(\sum_{i=p.M}^{(p+1)M-1} x[i]^2/M\right)^{\frac{1}{2}} \qquad p=0,1,2,\ldots,(paket\ sayısı-1) \tag{4.8}$$

bağıntısı ile hesaplanmaktadır.

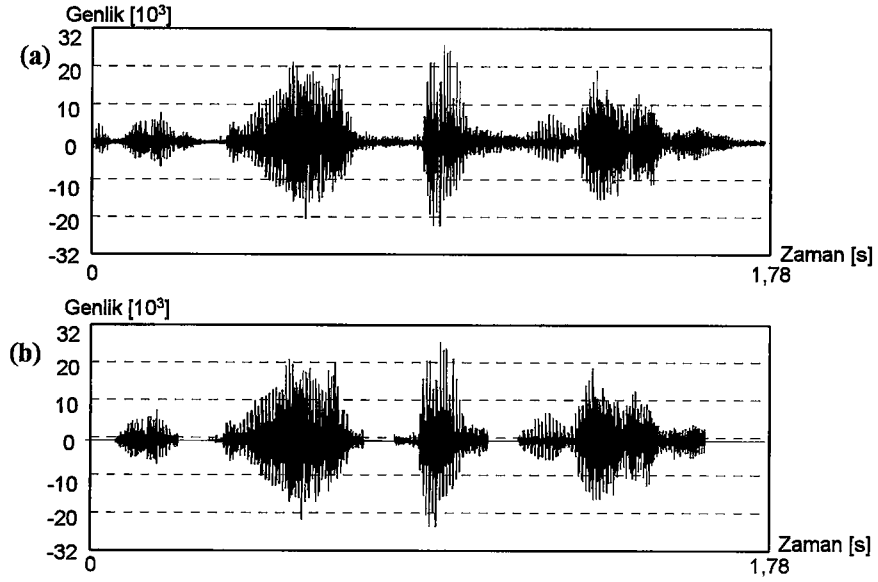
Geçiş fonksiyonu uyarınca, $x[n]$ giriş dizisinden alınan her bir paket için, gürültü bastırıcı çıkışı;

$$y[n]=\begin{cases} x[n] & ;KOD[p]\geq S_2 \ \ veya \ \ x[n]=0 \\ V_M.10^{\left(\alpha_2 20\log_{10}\left(\frac{x[n]}{V_M}\right)+m_2\right)/20} & ;S_1\leq KOD[p]<S_2 \ \ ve \ \ x[n]>0 \\ -V_M.10^{\left(\alpha_2 20\log_{10}\left(\frac{-x[n]}{V_M}\right)+m_2\right)/20} & ;S_1\leq KOD[p]<S_2 \ \ ve \ \ x[n]<0 \\ V_M.10^{\left(\alpha_1 20\log_{10}\left(\frac{x[n]}{V_M}\right)+m_1\right)/20} & ;0\leq KOD[p]<S_1 \ \ ve \ \ x[n]>0 \\ -V_M.10^{\left(\alpha_1 20\log_{10}\left(\frac{-x[n]}{V_M}\right)+m_1\right)/20} & ;0\leq KOD[p]<S_1 \ \ ve \ \ x[n]<0 \end{cases} \tag{4.9}$$

değerlerini almaktadır.

(4.9)'da verildiği gibi, giriş işaretine ait herhangi bir örnek için nasıl bir işlem yapılacağına, o örneğin değeri yerine, örneğin içinde bulunduğu paketin KOD değeri üzerinden karar verilmektedir. Bir başka deyişle, herhangi bir örneğin gürültü eşiğini aşıp aşmadığının analizi, yalnızca o an işleme giren örneğin değerine göre değil, o anki örnekten önceki ve sonraki örneklerin değerleri de göz önünde tutularak gerçekleştirilmektedir. KOD algılayıcılı yapı ile, yalnızca belirli bir süre boyunca gürültü eşiğinin altında değerler alan örnekler bastırılmakta, giriş işaretindeki anlık değişimler üzerinde herhangi bir değişim yaratılmamaktadır. Bu sayede, özellikle insan sesi gibi ani iniş çıkışlar içeren işaretlerle yapılan uygulamalarda, gürültü bastırıcı çıkışında süreksizlikler ve atlamalar içeren bir işaretin oluşmasının önüne geçilmektedir.

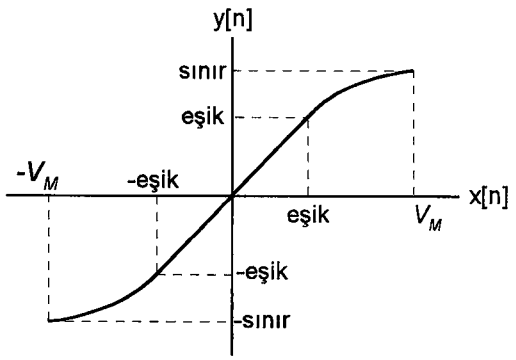
Şekil 4.5'de, GSİ yazılımı yoluyla gerçeklenen bir gürültü bastırma işlemi sonucunda elde edilen çıkış işareti gösterilmiştir.



Şekil 4.5 Gürültü bastırıcı çıkışı: (a) özgün işaret; (b) bastırılmış işaret.

4.1.3 Sıkıştırma

GSİ yazılımında, sıkıştırma işlemi için anlık değer algılayıcı bir yapı kullanılmaktadır (Bölüm 2.4.2). Şekil 4.6'da, sıkıştırıcıya ait geçiş fonksiyonu gösterilmiştir. Geçiş fonksiyonunda, giriş veya çıkış işaretlerinin olası en büyük değeri V_M olarak tanımlanmıştır (Bölüm 4.1.2).



Şekil 4.6 Sıkıştırıcı geçiş fonksiyonu.

Geçiş fonksiyonu uyarınca, N uzunluklu $x[n]$ giriş işareti ($n = 0,1,2,\ldots,N-1$) için sıkıştırıcı çıkışı;

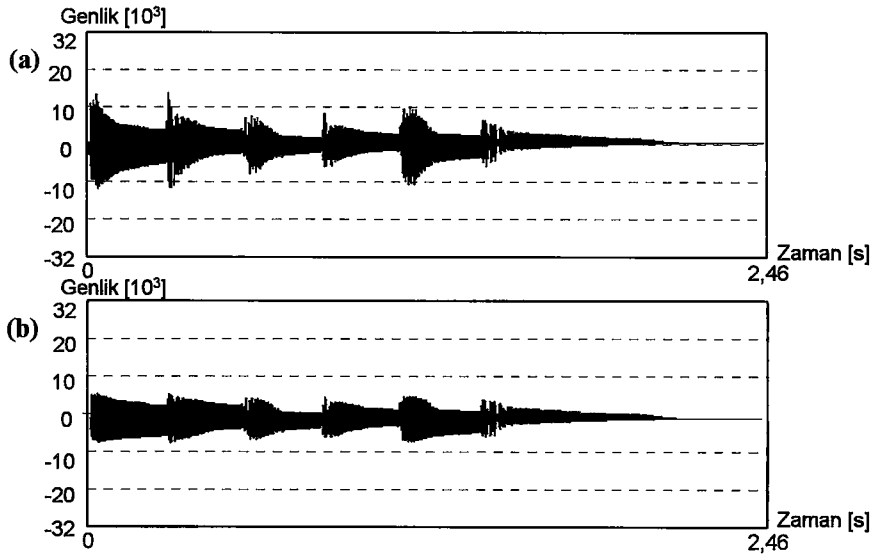
$$y[n] = \begin{cases} x[n] & ; -eşik < x[n] < eşik \\ k\ln(x[n]) + m & ; x[n] \geq eşik \\ -(k\ln(-x[n]) + m) & ; x[n] \leq -eşik \end{cases} \tag{4.10}$$

değerlerini almaktadır. k ve m sabitleri;

$$k = \frac{eşik - sınır}{\ln(eşik) - \ln(V_M)}$$
$$m = sınır - k\ln(V_M) \tag{4.11}$$

eşitlikleriyle hesaplanmaktadır.

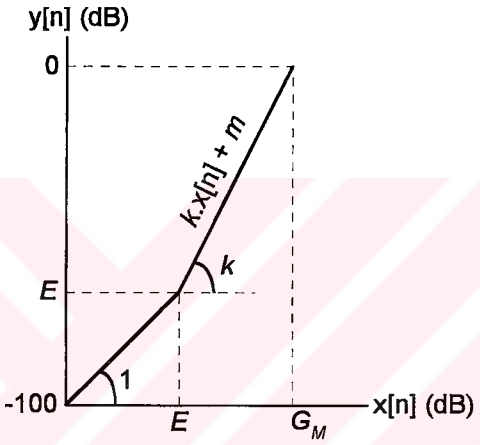
Sıkıştırıcının eşik ve sınır değerleri, kullanıcı tarafından GSİ üzerinde bulunan kaydırma çubukları aracılığıyla belirlenebilmektedir. Bu sayede, giriş işareti üzerinde kullanıcı tanımlı bir sıkıştırma etkisi yaratılabilmektedir. Şekil 4.7'de, GSİ yazılımı yoluyla gerçeklenen bir sıkıştırma işlemi sonucunda elde edilen çıkış işareti gösterilmiştir.



Şekil 4.7 Sıkıştırıcı çıkışı: (a) özgün işaret; (b) sıkıştırılmış işaret.

4.1.4 Genişletme

GSİ yazılımında, genişletme işlemi için anlık değer algılayıcı bir yapı kullanılmaktadır (Bölüm 2.4.3). Genişletme işlemi kullanıcı tanımlı bir biçimde, GSİ üzerinde bulunan *"genişletme oranı"* ve *"eşik değeri"* isimli iki kaydırma çubuğu aracılığıyla gerçekleştirilmektedir. Şekil 4.8'de, genişleticiye ait geçiş fonksiyonu gösterilmiştir. Geçiş fonksiyonunda, $x[n]$ giriş ve $y[n]$ çıkış işaretlerine ait örnek değerleri desibel cinsinden ifade edilmektedir. Giriş veya çıkış işaretlerinin alabileceği en büyük değer olarak tanımlanan V_M, desibel cinsinden sıfıra eşit olmaktadır (Bölüm 4.1.2).



Şekil 4.8 Genişletici geçiş fonksiyonu.

Geçiş fonksiyonunda, eğimin birden büyük değerler aldığı genişletme bölgesinin karakteristiği *k* ve *m* parametrelerine bağlı olarak;

$$\begin{aligned} k &= \textit{genişletme oranı} \\ m &= E(1-k) \end{aligned} \tag{4.12}$$

eşitlikleri ile belirlenmektedir. Eşik değerinin desibel cinsinden karşılığı E ile gösterilmiştir.

G_M, $x[n]$ işareti genişletme işleminden geçirildiğinde, olası en büyük çıkışı üreten örnek değeri g_M'nin desibel cinsinden karşılığını göstermektedir. g_M değeri;

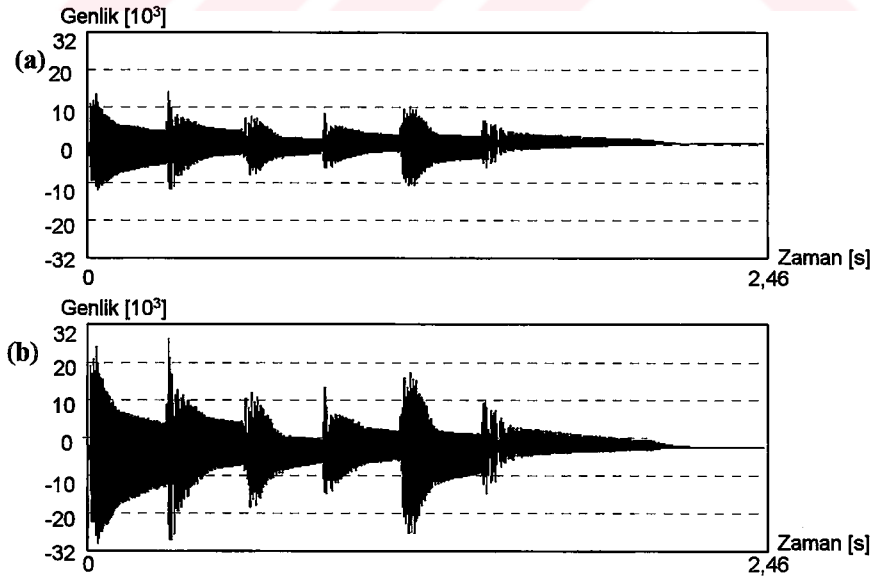
$$k.G_M + m = 0$$
$$20\log_{10}\left(\frac{g_M}{V_M}\right) = G_M = \frac{-m}{k} \tag{4.13}$$
$$g_M = V_M.10^{-m/20k}$$

şeklinde hesaplanmaktadır.

$x[n]$ giriş işaretinin, g_M'den büyük değerde bir örnek içermesi durumunda, genişletici çıkışında V_M'den büyük bir örnek değeri elde edilmektedir. Böyle bir çıkış sayısal bir taşmaya yol açmaktadır. Bu nedenle, en büyük örnek değeri $x[n]_M$ terimi ile ifade edilen $x[n]$ girişinin, genişletme işleminden önce bir küçültme işleminden geçirilerek en büyük değeri g_M olacak şekilde düzenlenmesi gerekmektedir. Küçültme işlemi, $x[n]$ işaretini oluşturan her bir örneğin;

$$sr = \frac{g_M}{x[n]_M} \tag{4.14}$$

değerli bir sabitle çarpılması yoluyla gerçekleştirilmektedir.



Şekil 4.9 Genişletici çıkışı: (a) özgün işaret; (b) genişletilmiş işaret.

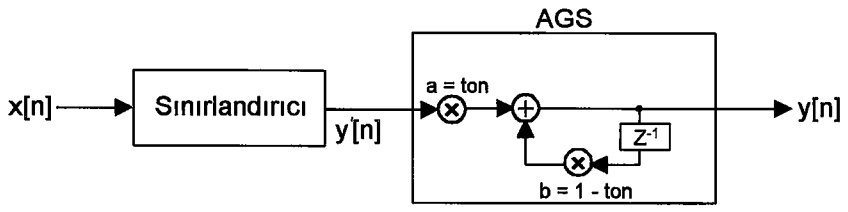
Geçiş fonksiyonu uyarınca, N uzunluklu $x[n]$ giriş işareti $(n = 0,1,2,\ldots,N-1)$ için genişletici çıkışı;

$$y[n] = \begin{cases} sr.x[n] & ; -sr.eşik \le sr.x[n] \le sr.eşik \\ V_M.10^{\left(k20\log_{10}\left(\frac{sr.x[n]}{V_M}\right)+m\right)/20} & ; sr.x[n] > sr.eşik \\ -V_M.10^{\left(k20\log_{10}\left(\frac{-sr.x[n]}{V_M}\right)+m\right)/20} & ; sr.x[n] < -sr.eşik \end{cases} \quad (4.15)$$

değerlerini almaktadır. Şekil 4.9'da, GSİ yazılımı yoluyla gerçeklenen bir genişletme işlemi sonucunda elde edilen çıkış işareti gösterilmiştir.

4.1.5 Kırpma

GSİ yazılımı yoluyla, giriş işareti üzerinde iki farklı biçimde (doğrusal ve logaritmik) kırpma etkisi oluşturulabilmektedir. Kırpma iki aşamalı bir işlemdir. Birinci aşamada, giriş işaretinin zarfi doğrusal veya logaritmik bir geçiş fonksiyonuna sahip anlık değer algılayıcılı bir sınırlandırıcı (Bölüm 2.4.2) kullanılarak şekillendirilmektedir. İstenilen kırpma etkisi bu aşamada yaratılmaktadır. İkinci aşamada ise, müzikal işaretler üzerinde yapılan uygulamalarda yaygın olarak kullanılan "ton ayarı" işlemi gerçekleştirilmektedir. Zarf şekillendiriciden alınan işaret alçak geçiren bir süzgeçten geçirilerek çıkış işareti elde edilmektedir. Ton ayarı için, tek kutuplu, basit bir yinelemeli AGS (Bölüm 2.1.4) kullanılmıştır. GSİ yazılımında kullanılan kırpıcı yapı Şekil 4.10'da gösterilmiştir.



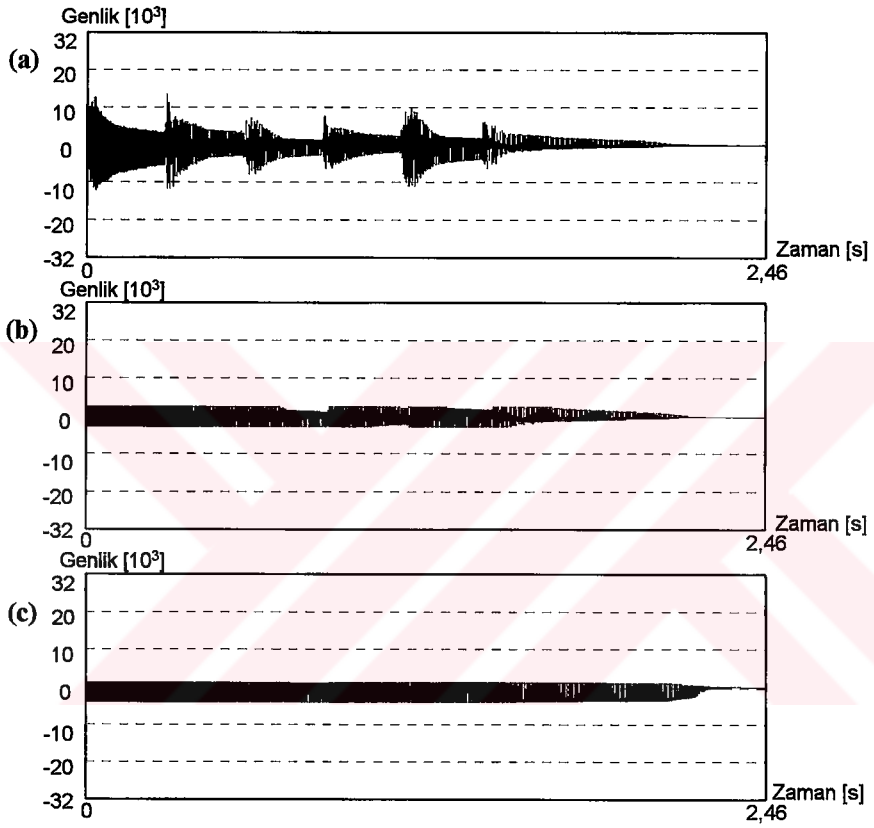
Şekil 4.10 Kırpıcı yapı.

Sınırlandırının eşik değeri ve süzgeç katsayıları, kullanıcı tarafından GSİ üzerinde bulunan *"kırpma oranı"* ve *"ton"* isimli kaydırma çubukları aracılığıyla belirlenebilmektedir.

N uzunluklu bir $x[n]$ giriş işareti ($n = 0,1,2,\ldots,N-1$) için sınırlandırıcı eşik değeri;

$$\textit{eşik değeri} = x[n]_M\,(1 - \textit{kırpma oranı}) \tag{4.16}$$

bağıntısıyla hesaplanmaktadır. AGS süzgeç katsayıları ise, *"ton"* kaydırma çubuğu ile seçilen değere bağlı olarak değişmektedir.



Şekil 4.13 Farklı özellikte kırpıcı çıkışları: (a) özgün işaret; (b) doğrusal kırpma; (c) logaritmik kırpma.

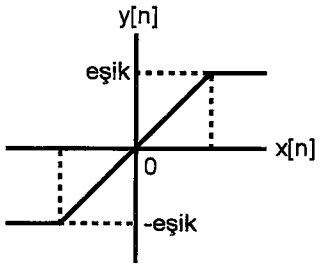
4.1.5.1 Doğrusal kırpma

Şekil 4.11'de, doğrusal kırpma işleminde kullanılan sınırlandırıya ait geçiş fonksiyonu gösterilmiştir.

N uzunluklu bir $x[n]$ giriş işareti ($n = 0,1,2,\ldots,N-1$) için sınırlandırıcı çıkışı;

$$y'[n]=\begin{cases}-eşik & ; x[n]\le -eşik\\ eşik & ; x[n]\ge eşik\\ x[n] & ; diğer\end{cases} \tag{4.17}$$

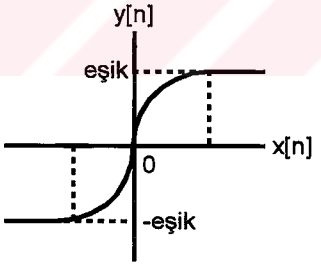
değerlerini almaktadır.



Şekil 4.11 Doğrusal kırpıcı geçiş fonksiyonu.

4.1.5.2 Logaritmik kırpma

Şekil 4.12'de, logaritmik kırpma işleminde kullanılan sınırlandırıya ait geçiş fonksiyonu gösterilmiştir.



Şekil 4.12 Logaritmik kırpıcı geçiş fonksiyonu.

N uzunluklu bir $x[n]$ giriş işareti ($n = 0,1,2,\ldots,N-1$) için sınırlandırıcı çıkışı;

$$y'[n]=\begin{cases}-eşik & ; x[n]\le -eşik\\ -k\ln(-x[n]) & ; -eşik < x[n] < 0\\ k\ln(x[n]) & ; 0 \le x[n] < eşik\\ eşik & ; x[n]\ge eşik\end{cases} \tag{4.18}$$

değerlerini almaktadır. k çarpanı;

$$k = \frac{eşik}{\ln(eşik)} \tag{4.19}$$

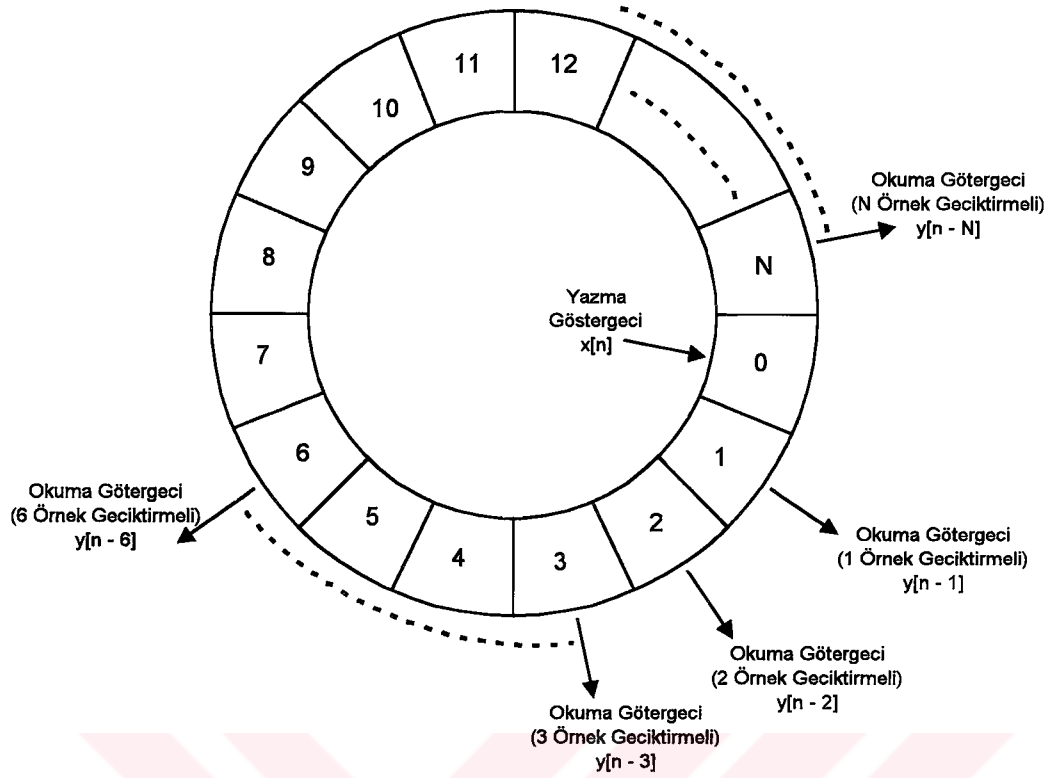
eşitliğiyle bulunmaktadır.

Şekil 4.13'de, GSİ yazılımı yoluyla gerçeklenen doğrusal ve logaritmik kırpma işlemleri sonucunda elde edilen çıkış işaretleri gösterilmiştir.

4.2 Geciktirme Etkileri

4.2.1 Çok çıkışlı sayısal geciktirici

Şekil 4.14'de, GSİ yazılımında kullanılan sayısal geciktirici (Bölüm 2.2.1) gösterilmiştir. Şekilde görüldüğü gibi, sayısal geciktirici çevrimsel bir yapıda ve birden fazla sayıda çıkış alınabilecek şekilde tasarlanmıştır. Geciktiricinin kaç örneklik bir N uzunluğunda olacağı ve her bir çıkışın geciktirme süresi yazılımsal olarak belirlenebilmektedir. Uygulamada, çok çıkışlı sayısal geciktirici (ÇÇSG) için bellekte N uzunluklu bir alan dinamik olarak ayrılmakta ve giriş çıkış işlemleri yazılımsal olarak göstericiler (pointers) yoluyla gerçekleştirilmektedir. ÇÇSG'nin her bir çıkışına ait geciktirme süresi istenildiğinde yeni bir değere şartlandırılabilmektedir. Bir başka deyişle, ÇÇSG çıkışlarından istenilenleri sabit zamanlı, istenilenleri değişken zamanlı geciktirilmiş işaretler üretecek şekilde kullanılabilmektedir. ÇÇSG'nin çıkışlarına atanan geciktirme süresi ilk (initial) değerleri, uygulama boyunca belirli bir bellek bölgesinde kalıcı olarak saklanmaktadır. Bu yolla, herhangi bir çıkışın geciktirme süresinde ilk değerine bağımlı düzenli bir değişim oluşturulabilmektedir. GSİ yazılımında geciktirme etkileri ÇÇSG'nin kullanıldığı genel amaçlı bir yapı yoluyla gerçeklenmektedir. ÇÇSG'nin esnek kullanım özellikleri sayesinde, genel amaçlı yapıya ait kimi parametrelerin değerleri değiştirilerek farklı karakteristikte sabit zaman (Bölüm 2.2.3) ve değişken zaman (Bölüm 2.2.4) geciktirme etkileri elde edilebilmektedir.



Şekil 4.14 Çok çıkışlı sayısal geciktirici.

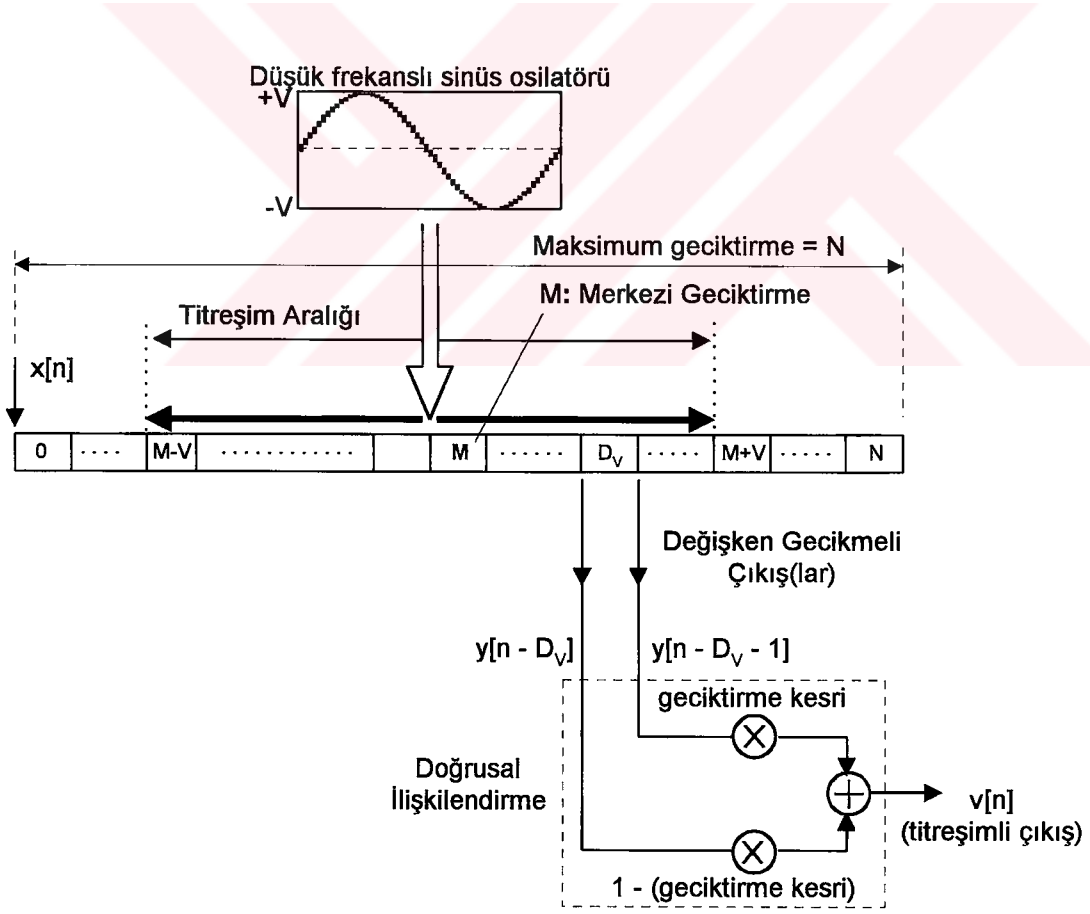
4.2.2 Değişken geciktirmeli çıkışlar ve doğrusal ilişkilendirme

ÇÇSG'den alınan herhangi bir çıkışın geciktirme süresi, örnekleme peryoduna eşit aralıklarla, dinamik bir biçimde değiştirilebilmektedir. Bu işlem düşük frekanslı bir sinüs osilatörü (DFO) aracılığıyla gerçekleştirilmekte ve geciktirici çıkışında, üzerinde titreşim etkisi oluşturulmuş bir işaret elde edilmektedir. Titreşimli işaretin özgün işaretle belirli oranlarda karıştırılması yoluyla, giriş işareti üzerinde dalgalandırma, koro, çiftleme, yankı gibi farklı karakteristikte geciktirme etkileri yaratılabilmektedir. Belirli bir merkezi geciktirme değeri etrafında oluşturulan bu salınım, geciktiriciden alınan çıkış işaretinde süreksizlikler oluşmaktadır. Geciktirici ilişkilendirmesi yoluyla, geciktirme süresinin tamsayı örnek değerlerine ek olarak kesirli bir değeri de içermesi sağlanarak çıkışta oluşan bu süreksizliklerin önüne geçilebilmektedir. GSİ yazılımında doğrusal ilişkilendirme (linear interpolation) yönteminden yararlanılmıştır.

Şekil 4.15'de, sayısal geciktiriciden doğrusal ilişkilendirme yoluyla titreşimli bir çıkışın nasıl elde edildiği gösterilmiştir. Şekilde görüldüğü gibi, N uzunluklu geciktiriciden M merkez geciktirmeli değişken bir çıkış alınmaktadır. Çıkışın geciktirme süresi D_V, sinüzoidal bir DFO ile $M \pm V$ titreşim aralığında, sürekli bir biçimde değiştirilmektedir ($M+V < N-1$). Yazılımsal olarak geciktirme süresinin değiştirilmesi, çıkışın alındığı göstericinin geciktiriciyi oluşturan bellek bölgesinin farklı bir gözüne konumlandırılması anlamına gelmektedir. Geciktirme süresine yeni değerler verilmesi işlemi, osilatörün ürettiği,

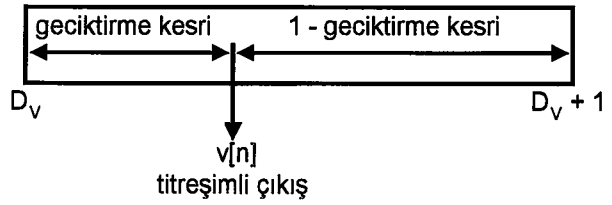
$$DFO[n] = V.\sin\left(\frac{2\pi f_V n}{F_s} + \phi\right) \tag{4.20}$$

ayrık zaman işareti kullanılarak örnekleme peryoduna eşit aralıklarla gerçekleştirilmektedir. (4.20)'de, örnekleme hızı F_s olarak tanımlanmıştır.



Şekil 4.15 Değişken zamanlı geciktiriciden titreşimli çıkış alınması.

Geciktirme süresinin peryodik bir biçimde değiştirilmesinin neden olduğu süreksizliklerin önüne geçmek için, titreşimli işaret doğrusal ilişkilendirme yoluyla elde edilmektedir. Şekil 4.16'da, geciktirici ilişkilendirmesi kullanılarak titreşimli çıkış üretilmesi işlemi gösterilmiştir. Şekilde görüldüğü gibi, çıkış işareti geciktirici üzerindeki ardışıl iki örneğin arasından alınmaktadır. Bir başka deyişle, geciktirme süresi, tamsayı örnek değeri ile kesirli bir değerin toplamından elde edilmektedir.



Şekil 4.16 Doğrusal ilişkilendirme ile titreşimli çıkışın üretilmesi.

Osilatörden alınan $DFO[n]$ kesirli değerinin aşağı yuvarlanmasıyla (*floor*) bulunan tamsayı değeri $T_{DFO}[n]$ olarak alındığında, geciktirme kesri;

$$geciktirme\ kesri = DFO[n] - T_{DFO}[n] \tag{4.21}$$

bağıntısı ile hesaplanmaktadır. Bu durumda titreşimli çıkış $v[n]$, doğrusal ilişkilendirme yoluyla,

$$v[n] = (1 - \ geciktirme\ kesri)\ .\ y[n - D_V] + geciktirme\ kesri\ .\ y[n - D_V - 1] \tag{4.22}$$

eşitliği ile elde edilmektedir. Titreşimli işaret elde edildikten sonra, çıkış geciktirme süresi,

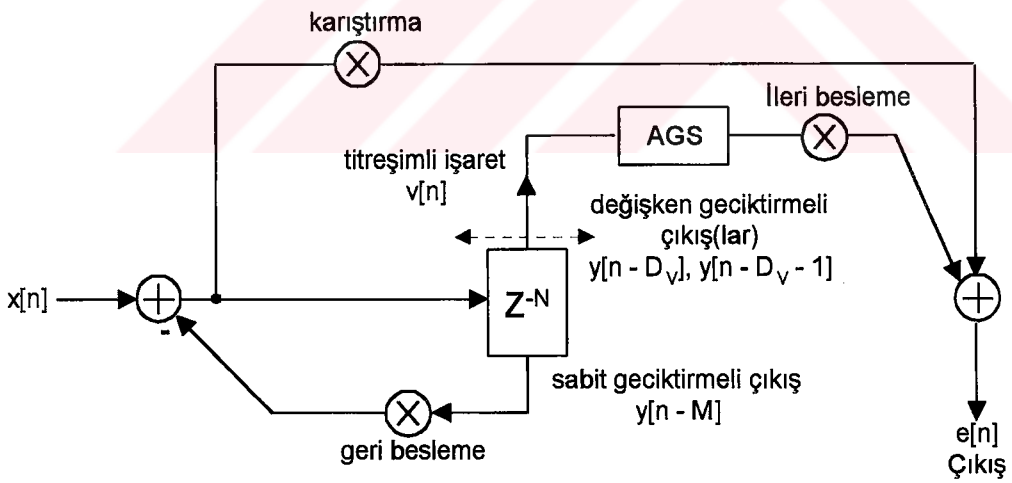
$$D_V = M + T_{DFO}[n] \tag{4.23}$$

bağıntısıyla, osilatörden alınan değere bağlı olarak değiştirilmektedir. Bu şekilde, doğrusal ilişkilendirme yoluyla titreşimli işaret üretilmesi ve M merkez değerli D_V geciktirme

süresinin güncelleştirilmesi işlemleri, çıkıştan alınan her bir örnek için, örnekleme peryoduna eşit aralıklarla, sürekli bir biçimde yinelenmektedir.

4.2.3 Genel amaçlı geciktirme etkisi üreteci

Şekil 4.17'de, GSİ yazılımında kullanılan genel amaçlı etki üreteci (GEÜ) gösterilmiştir. Genel amaçlı yapı Dattorro'nun (1997) tasarımları temel alınarak oluşturulmuştur. GEÜ çok çıkışlı bir sayısal geciktirici (Bölüm 4.2.1) içermektedir. Şekilde görüldüğü gibi, GEÜ çıkışı özgün işaret ile titreşimli işaretin toplamından elde edilmektedir. Titreşimli işaret $v[n]$, ÇÇSG'den alınan değişken geciktirmeli bir çıkış yoluyla üretilmektedir (Bölüm 4.2.2). Titreşimli çıkışa ait merkezi geciktirme süresinin değeri ile geciktirme süresinin değiştirilmesinde kullanılan DFO'nun tepe genliği, titreşim frekansı ve fazı yazılımsal olarak belirlenebilmektedir. GEÜ yapısında ÇÇSG'den alınan sabit zamanlı bir çıkış üzerinden gerçekleştirilen bir negatif geri besleme yolu da bulunmaktadır. Karıştırma (özgün işaret), ileri besleme (titreşimli işaret) ve geri besleme katsayılarına uygun değerler verilerek farklı türden geciktirme etkileri elde edilebilmektedir.



Şekil 4.17 Geciktirme etkisi üreteci.

Özgün işaret ile titreşimli işaretin toplanması, giriş işareti spektrumu üzerinde bir tarak süzgeç etkisi yaratmaktadır (Bölüm 2.1.7). Tarak süzgeç etkisi, giriş işareti spektrumunda düzenli aralıklarla iniş çıkışlar (tepe ve sıfır noktaları) oluşmasına yol açmaktadır. Bu tür bir

çıkış işaretinin elde edilmesi, dalgalandırma (Bölüm 2.2.4.1) uygulamalarında istenilen bir durumdur. Çünkü dalgalandırma, giriş işareti üzerinde kısa geciktirme süreli bir tarak süzgeç etkisine benzer bir değişim oluşturmaktadır. Bu nedenle, çıkış işareti spektrumundaki tepe ve sıfır noktaları arasındaki geçişler ne kadar derinleştirilirse, o kadar belirgin bir dalgalandırma etkisi elde edilmektedir. Koro etkisi (Bölüm 2.1.8) ise dalgalandırmaya göre daha uzun geciktirme süreleri kullanılarak gerçeklenmektedir. Koro etkisi oluşturma, giriş işaretini bir dizi paralel bağlı dalgalandırıcıdan geçirmeye benzetilebilir. Koro etkisinin giriş işareti spektrumunda yarattığı değişim dalgalandırmaya göre oldukça yumuşak karakteristiklidir. Bu nedenle, özgün işaret ile titreşimli işaretin toplanmasının giriş işareti spektrumunda oluşturduğu iniş çıkışlar koro etkisi için istenilmeyen bir durumdur. Standart koro etkisi uygulamalarında, özgün işaret ile titreşimli işaretin farklı oranlarda karıştırılması yoluyla bu istenilmeyen durumun ortadan kaldırılmasına çalışılmaktadır.

GEÜ ile koro etkisi oluşturulurken, negatif geri besleme kullanılarak giriş işareti spektrumunda oluşan iniş çıkışların dengelenmesi sağlanmaktadır. Geri besleme, ÇÇSG'den alınan sabit geciktirme zamanlı bir çıkış üzerinden gerçekleştirilmektedir. Sabit geciktirme süresi, titreşimli işaretin alındığı değişken zamanlı çıkışın merkezi geciktirme süresine eşittir. Değişken zamanlı çıkıştan alınan işaret perde ötelemeli olduğu için geri beslemede kullanılmamıştır. Geri beslemeli yapı, katsayılar uygun değerlerde seçildiği ve değişken çıkış geciktirme süresi D_V'nin sabit çıkış geciktirme süresi M'ye yakın değerler aldığı durumlarda, giriş işareti üzerinde bir tüm geçiren süzgeç (Bölüm 2.1.8) etkisi yaratmaktadır. GEÜ ile dalgalandırma etkisi oluşturulurken de geri beslemeden yararlanılmaktadır. Koro etkisinden farklı olarak, dalgalandırma etkisi için pozitif geri besleme kullanılmaktadır. Bu sayede, giriş işareti üzerinde daha derin iniş çıkışların oluşması ve daha belirgin bir dalgalandırma etkisinin yaratılması sağlanmaktadır.

Çizelge 4.1'de, farklı türden geciktirme etkilerine ait GEÜ katsayı değerleri verilmiştir. Katsayılara uygun değerlerin atanması ve merkezi geciktirme süresinin Çizelge 4.2'de verilen aralıklarda seçilmesi yoluyla, adı geçen tüm geciktirme etkileri tek bir genel amaçlı yapı aracılığıyla oluşturulabilmektedir.

Çizelge 4.1 GEÜ katsayı değerleri.

Geciktirme Etkisi	Karıştırma	İleri Besleme	Geri Besleme
Titreşim	0,0	1,0	0,0
Dalgalandırma	0,7071	0,7071	−0,071
Standart Koro	1,0	0,7071	0,0
Koro	0,7071	1,0	0,7071
Çiftleme	0,7071	0,7071	0,0
Yankı	1,0	≤ 1,0	< 1,0

GEÜ yapısının geçiş fonksiyonu,

$$H_{GEÜ}(z) = \frac{karıştırma + ileri\ besleme.z^{-D_V}}{1 + geri\ besleme\ .z^{-M}} \tag{4.24}$$

eşitliği ile ifade edilmektedir. Tüm geçiren bir süzgeç (Bölüm 2.1.8) ise,

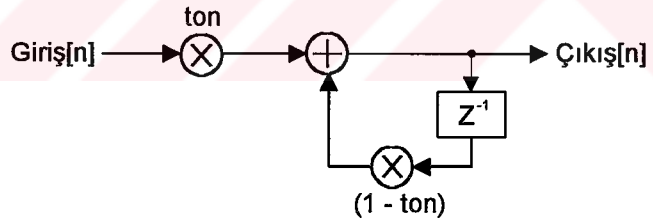
$$H(z) = \frac{g + z^{-m}}{1 + g.z^{-m}} \tag{4.25}$$

eşitliği ile belirtilen bir geçiş fonksiyonuna sahiptir. Çizelge 4.1'de koro etkisi için verilen katsayı değerleri, yapının davranışını tüm geçiren süzgece olabildiğince yaklaştıracak şekilde seçilmiştir. (4.24)'de görüldüğü gibi, ileri besleme katsayısının 1,0 değerinde tutulması ve geri besleme ile karıştırma katsayılarının birbirine eşit değerlerde seçilmesi, değişken geciktirme süresi D_V 'nin sabit çıkış geciktirme süresi M'ye yakın değerler aldığı durumlarda, GEÜ geçiş fonksiyonunu tüm geçiren süzgeç geçiş fonksiyonuna (4.25) yakınlaştırmaktadır. Sonuç olarak, koro etkisi değerleri kullanılarak giriş işareti üzerinde bir tüm geçiren süzgeç etkisi yaratılmaktadır. GEÜ yapısının dalgalandırıcı olarak kullanımında ise, ileri besleme ve karıştırma katsayıları birbirine eşit değerlerde seçilerek işaret spektrumu üzerinde olabildiğince derin iniş çıkışların oluşturulması sağlanmaktadır.

Çizelge 4.2 Farklı türden geciktirme etkilerine ait merkezi geciktirme, titreşim aralığı ve titreşim frekansı değerleri.

Geciktirme Etkisi	M $[ms]$	V $[\times M]$	f_V $[Hz]$
Titreşim	1 - 5	0,4 - 0,7	2 - 8
Dalgalandırma	1,5 - 9	0,8 - 0,999	0,25 - 1,1
Standart Koro	10 - 25	0 - 0,007	2 - 8
Koro	10 - 25	0 - 0,007	2 - 8
Çiftleme	15 - 80	0,4 - 0,6995	0,14 - 0,16
Yankı	80 - 2000	0,0498 - 0,0499	0,14 - 0,16

Şekil 4.17'de görüldüğü gibi, GEÜ yapısında, titreşimli işaret $v[n]$ ileri besleme katsayısı ile çarpılıp özgün işaret ile karıştırılmadan önce, basit bir yinelemeli alçak geçiren süzgeçten (Bölüm 2.1.4) geçirilmektedir. Bu sayede, hem ilişkilendirmeye rağmen titreşimli işarette etkisini göstermeye devam eden süreksizlikler yumuşatılmakta, hem de çıkışta elde edilen işaretin tonu üzerinde ayarlama yapma olanağı elde edilmektedir. Şekil 4.18'de, GEÜ yapısında kullanılan AGS gösterilmiştir. Alçak geçiren süzgeç katsayıları yazılımsal olarak değiştirilebilmektedir.



Şekil 4.18 Tek kutuplu süzgeç.

GSİ yazılımında, GEÜ ile oluşturulan her bir etki, kullanıcı tarafından görsel kaydırma çubukları aracılığıyla denetlenebilmektedir. Kaydırma çubukları yardımıyla, M merkezi geciktirme süresi, $M \pm V$ titreşim aralığı ve f_V titreşim frekansı değerleri, her bir etki için kullanıcı tarafından belirlenebilmektedir. Bunlardan başka, yapıdaki AGS katsayılarını değiştirerek, elde edilen işaret üzerinde ton ayarı yapmaya yarayan bir kaydırma çubuğu da bulunmaktadır. Tüm etkiler için, *"ton"* kaydırma çubuğu $[0,2-0,7]$ kapalı aralığında

değerler almaktadır. Çizelge 4.2'de, farklı türden geciktirme etkileri için GSİ yazılımında kullanılan M, V, ve f_V değer aralıkları verilmiştir.

Titreşim etkisi oluşturulurken, karıştırma ve geri besleme katsayılarına 0,0 değeri verilmektedir. İleri besleme katsayısı ise 1,0 değerinde seçilmektedir. Bu sayede, GEÜ çıkışında yalnızca ÇÇSG'den alınan titreşimli işaretin elde edilmesi sağlanmaktadır.

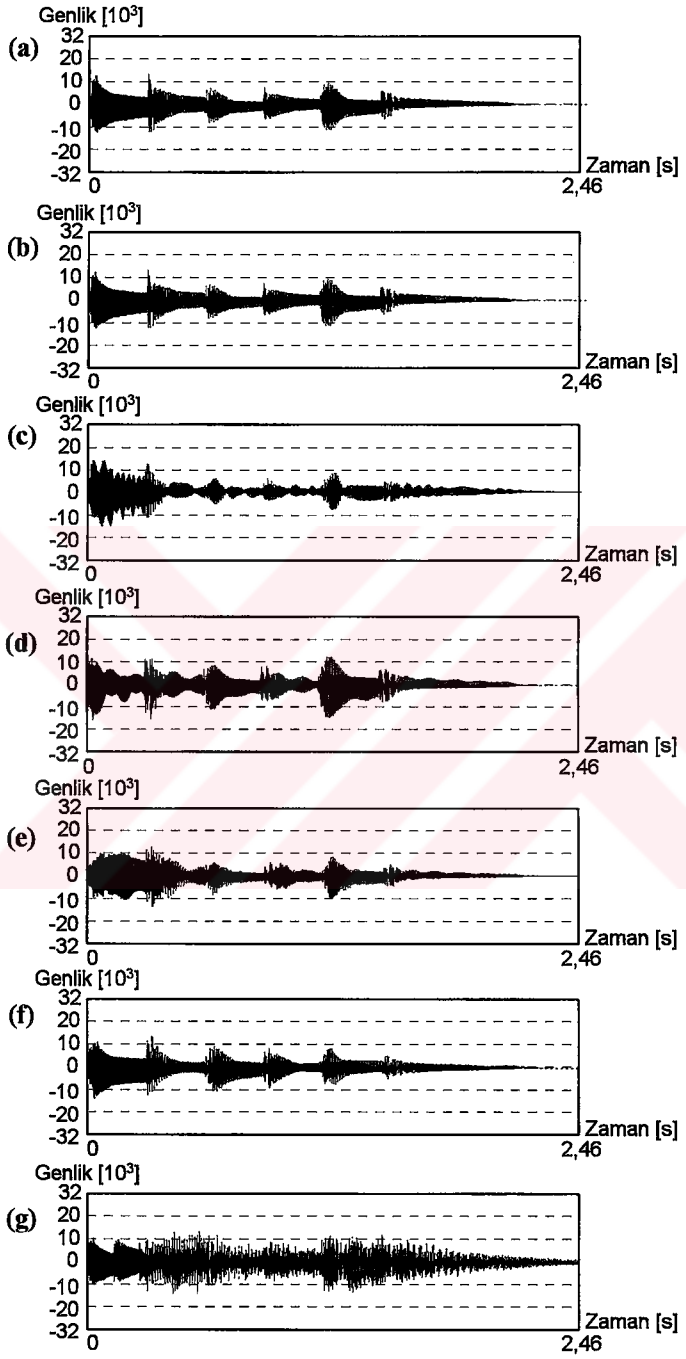
Dalgalandırma etkisinin giriş işareti spektrumunda derin iniş çıkışlar oluşturması gerekmektedir. Bu nedenle, titreşimli işaret ile özgün işaret eşit oranlarda karıştırılmakta ve pozitif bir geri besleme kullanılmaktadır. Titreşim aralığı ve titreşim frekansı arttırıldıkça daha belirgin bir dalgalandırma etkisi elde edilmektedir. Çıkışta etkin bir tarak süzgeç etkisi oluşturabilmek için merkezi geciktirme süresi kısa tutulmuştur. Dalgalandırma, özgün işarete sabit zamanlı geciktiriciden alınan yankıları eklemekte ve bu yankılar değişken zamanlarda geciktirilmektedir. İşlem sonucunda, sesin, üretildiği kaynak hareket ettiriliyormuş gibi algılanması sağlanmaktadır.

Standart koro etkisinde geri besleme kullanılmamaktadır. Özgün işaret ile titreşimli işaret, giriş işareti spektrumunda oluşan istenilmeyen iniş çıkışların dengelenmesi amacıyla farklı oranlarda karıştırılmaktadır. Koro etkisinde ise negatif geri beslemeden yararlanılmıştır. Uygun katsayı değerlerinin seçimi ve titreşimin dar bir aralıkta tutulması yoluyla, yapının giriş işareti üzerinde bir tüm geçiren süzgeç etkisi oluşturması sağlanmaktadır. Her iki koro etkisinde de, gene aynı nedenlerden dolayı, dalgalandırma etkisine oranla daha uzun süreli bir merkezi geciktirme kullanılmaktadır.

Çiftleme etkisinde de, standart koro etkisinde olduğu gibi geri besleme kullanılmamaktadır. Bununla birlikte, çifleme etkisinde, standart korodan farklı olarak, merkezi geciktirme süresi daha uzun seçilmekte, titreşimli işaret ile özgün işaret eşit oranlarda karıştırılmakta ve çok daha düşük değerde bir titreşim frekansı kullanılmaktadır. Sonuç olarak, GEÜ çıkışında, özgün işarete ait her bir örnek değerinin belirli bir süre sonra tekrarlanması sağlanmaktadır.

Yankı etkisinde, çiftleme etkisine göre çok daha uzun süreli bir merkezi geciktirme kullanılmaktadır. Titreşimli işaret ile özgün işaret, eşit veya daha düşük oranlarda

karıştırılmaktadır. Yankı etkisi de, çiftleme etkisi gibi sabit zamanlı bir karakteristik taşıdığından (Bölüm 2.2.3), titreşim aralığı dar tutulmakta ve titreşim frekansı düşük bir değerde seçilmektedir.



Şekil 4.19 GEÜ ile gerçeklenebilen geciktirme etkileri: (a) özgün işaret; (b) titreşim işaret; (c) dalgalandırma; (d) standart koro; (e) koro; (f) çiftleme; (g) yankı.

Şekil 4.19'da, GSİ yazılımında GEÜ kullanılarak, üzerlerinde farklı türden geciktirme etkileri oluşturulmuş işaretler gösterilmiştir.

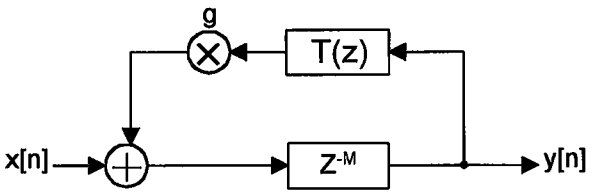
4.3 Çınlatma

GSİ yazılımında kullanılan tümleşik çınlatıcı yapı (TÇY) Moorer'ın (1979) tasarımları temel alınarak oluşturulmuştur. Schroeder'in (1961, 1962) özgün çınlatıcı yapıları (Bölüm 2.3.7) ile gerçeklenen uygulamalarda, aşağıda sıralanan sorunlarla karşılaşılmaktadır;

- Yoğun çınlama bölgesi (Bölüm 2.3.5), gerçekçi bir biçimde elde edilememektedir. Çınlama bölgesi başlangıcında üretilen yankı işaretleri gereken sıklıkta birbirini izlememekte ve üstel bir biçimde sönümlenmemektedir.
- Çınlama bölgesinin karakteristiği, süzgeç katsayıları ve geciktirici uzunlukları gibi çınlatıcıya ait parametre değerlerine aşırı hassas bir biçimde bağımlılık göstermektedir. Herhangi bir geciktiricide yapılan bir iki örneklik bir değişim, sönümlenme eğimini önemli ölçüde etkilemekte ve çınlatıcı çıkışında elde edilen işarette büyük farklılıkların oluşmasına neden olmaktadır.
- Çınlama süresi (Bölüm 2.3.3) attırıldığında, frekans bağımlı geciktirmeler nedeniyle ötme etkisi oluşmaktadır. Ötme etkisi, çınlatıcı yapı çıkışında metalik bir ses işaretinin elde edilmesine yol açmaktadır.

Moorer'ın (1979) tasarımında Schroeder'in (1970) ileri çınlatıcı yapısı (Bölüm 2.3.8, Şekil 2.27) temel alınmakla birlikte, çınlatma birimlerinin yapısal özellikleri ve geciktirici kullanımları üzerinde önemli geliştirmeler yapılmıştır.

Bölüm 2.3.6.1 ve 2.3.6.2'de anlatıldığı gibi, çınlatma birimlerinin tasarımında yinelemeli tarak süzgeç veya tüm geçiren süzgeçlerden yararlanılmaktadır. Bir kaynaktan yayılan bir ses işaretine ait yüksek frekans bileşenlerinden bir kısmı, havada ilerlerken ve yüzeylere çarpıp yansırken bastırılmaktadır. Çınlatma birimlerindeki katsayıların (kazanç terimlerinin) yerine alçak geçiren karakteristikte süzgeçler kullanılarak, bu olay yapay olarak gerçeklenmeye çalışılmaktadır.



Şekil 4.20 Geri besleme yoluna alçak geçiren karakteristikli bir diğer süzgeç yerleştirilmiş tarak süzgeç.

Şekil 4.20'de, bu tür bir tarak süzgeç çınlatma birimi gösterilmiştir. Şekilde görüldüğü gibi, çınlatma biriminin geri besleme yoluna geçiş fonksiyonu $T(z)$ ile ifade edilen bir diğer süzgeç yerleştirilmiştir. Çınlatma biriminin geçiş fonksiyonu,

$$G(z)=\frac{z^{-M}}{1-gT(z)z^{-M}} \tag{4.26}$$

bağıntısı ile verilmektedir. Geçiş fonksiyonu birim çember üzerine taşındığında,

$$\left|G\left(e^{j.\omega.P_S}\right)\right|^2=\frac{1}{1+g^2|T|^2-2g|T|\cos(\theta(\omega)-M\omega P_S)} \tag{4.27}$$

eşitliği elde edilmektedir. Eşitlikte, $T\left(e^{j\omega P_S}\right)$ karmaşık büyüklüğünün genliği $|T|$, açısı ise $\theta(\omega)$ terimi ile gösterilmiştir. Örnekleme peryotu, P_S olarak tanımlanmıştır. Çınlatma biriminin kararlı durumda kalması için,

$$g<1/\max_{0\le\omega\le 2\pi}|T| \tag{4.28}$$

gerek ve yeter şartının sağlanması gerekmektedir.

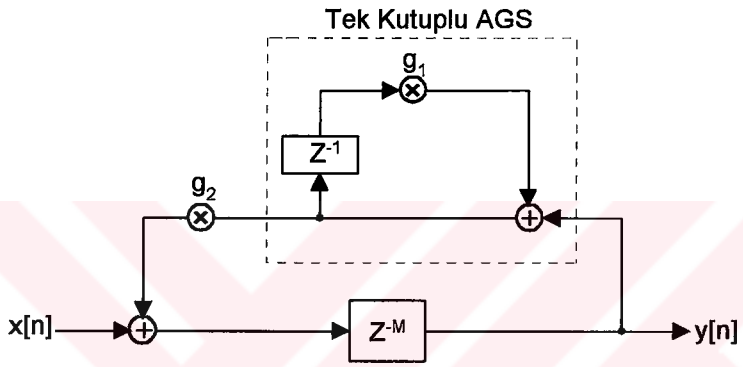
Şekil 4.21'de, GSİ yazılımında kullanılan tarak süzgeç çınlatma birimi gösterilmiştir. Şekilde görüldüğü gibi, çınlatma biriminin geri besleme çevrimine birinci dereceden bir yinelemeli AGS yerleştirilmiştir. AGS geçiş fonksiyonu,

$$T(z)=\frac{1}{1-g_1 z^{-1}} \tag{4.29}$$

bağıntısı ile ifade edilmekte ve $\omega = 0$ için,

$$|T|_{\max} = T(1)=\frac{1}{1-g_1} \tag{4.30}$$

en büyük değerini almaktadır.



Şekil 4.21 Geri besleme yoluna tek kutuplu AGS yerleştirilmiş tarak süzgeç.

(4.28) uyarınca, çınlatma biriminin kararlılığı,

$$\frac{g_2}{1-g_1}<1 \tag{4.31}$$

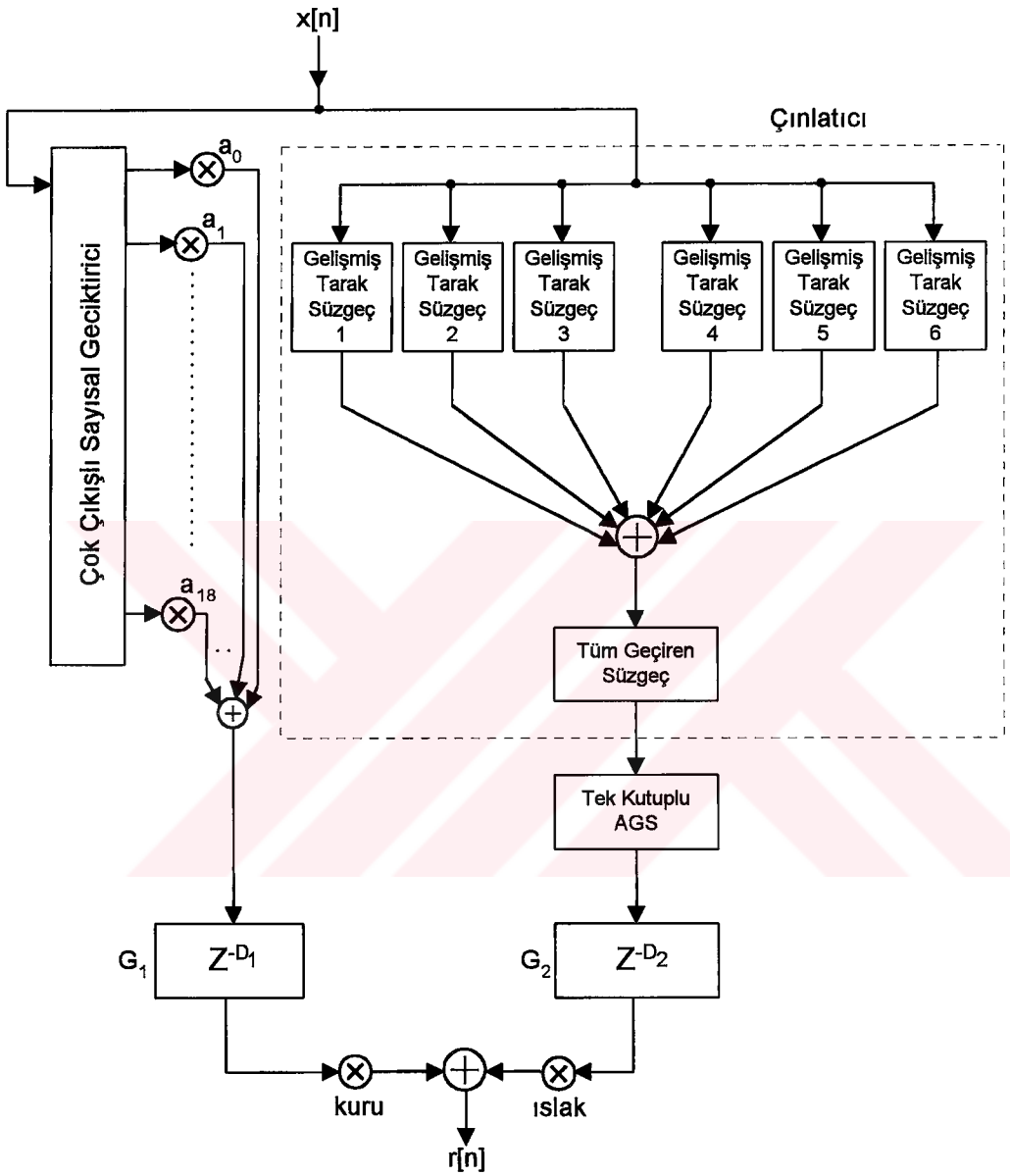
gerek ve yeter şartının sağlanmasını zorunlu kılmaktadır. Daha genel bir ifade ile, çınlatma biriminin kazancı g olarak alındığında, kararlılığın sağlanması için g_1 ve g_2 parametrelerinin,

$$g_2 = g(1-g_1) \tag{4.32}$$

koşulunu gerçekleyen değerlerde seçilmeleri gerekmektedir. Çınlatma birimi kazancı,

$$0 < g < 1 \tag{4.33}$$

aralığında değerler almaktadır.



Şekil 4.22 Tümleşik çınlatıcı yapı.

Şekil 4.22'de, GSİ yazılımında kullanılan TÇY gösterilmiştir. TÇY, yoğun çınlama bölgesini altı adet seri tarak süzgeç ve bir adet paralel tüm geçiren süzgeçten oluşan bir çınlatıcı yoluyla gerçeklemektedir. Tarak süzgeç çınlatma birimlerinin her birinin geri besleme yoluna birinci dereceden bir AGS yerleştirilmiştir (Şekil 4.21). İlk ulaşan yansımaların (Bölüm 2.3.5

ve 2.3.8) gerçeklenmesinde ise, çok çıkışlı bir sayısal geciktiriciden (Bölüm 4.2.1) yararlanılmıştır.

TÇY'da, tarak süzgeç çınlatma birimlerinin tümü için ortak bir g kazancı kullanılmaktadır. Bu ortak kazanç değeri, tümleşik yapının bütününe ait bir çınlama süresi (Bölüm 2.3.3) tanımlamaktadır. Sistemin çınlama süresi,

$$\textit{çınlama süresi}\,[s] = \frac{0{,}366}{1-g} \tag{4.34}$$

bağıntısı yoluyla hesaplanmaktadır. GSİ yazılımında bulunan *"çınlatma kazancı"* isimli görsel bir kaydırma çubuğu aracılığıyla, g sistem kazancı kullanıcı tarafından yazılımsal olarak denetlenebilmekte ve 0,75 ila 0,95 arasında değerler alabilmektedir.

Çizelge 4.3 Tarak süzgeç geciktirme süreleri ve katsayı değerleri.

	Geciktirme $[ms]$	g_1
Tarak Süzgeç 1	53	0,405
Tarak Süzgeç 2	59	0,423
Tarak Süzgeç 3	63	0,441
Tarak Süzgeç 4	69	0,458
Tarak Süzgeç 5	73	0,529
Tarak Süzgeç 6	79	0,564

Çizelge 4.3'de, TÇY'da kullanılan tarak süzgeç çınlatma birimlerine ait geciktirme süreleri ve 44100 Hz örnekleme hızı için, g_1 kazanç değerleri verilmiştir. Geciktirme süreleri 1 : 1,5 doğrusal oranı üzerinden dağıtılmıştır. Tarak süzgeçlerin ürettiği yankı işaretlerinin birbirleri ile çakışarak çıkışta sayısal bir taşmaya yol açmasını önlemek amacıyla, geciktirme süreleri asal değerlerden seçilmiştir. GSİ yazılımında bulunan *"çınlatma derinliği"* isimli görsel bir kaydırma çubuğu aracılığıyla, geciktirme süreleri kullanıcı tarafından oransal olarak değiştirilebilmektedir. Tarak süzgeçlere ait g_1 kazanç değerleri, örnekleme hızına bağlı olarak değişim göstermektedir. Farklı örnekleme hızları ile yapılan uygulamalarda, doğrusal

orantılama yoluyla yeni g_1 kazanç değerlerinin belirlenmesi gerekmektedir. Tarak süzgeçlere ait g_2 kazanç değerleri ise, g ve g_1 değerlerine bağlı olarak (4.32) bağıntısı yoluyla belirlenmektedir. TÇY'da yer alan tüm geçiren süzgeç çınlatma birimi için, 6 ms süreli bir geciktirme kullanılmış ve süzgeç kazancı 0,7 değerinde seçilmiştir.

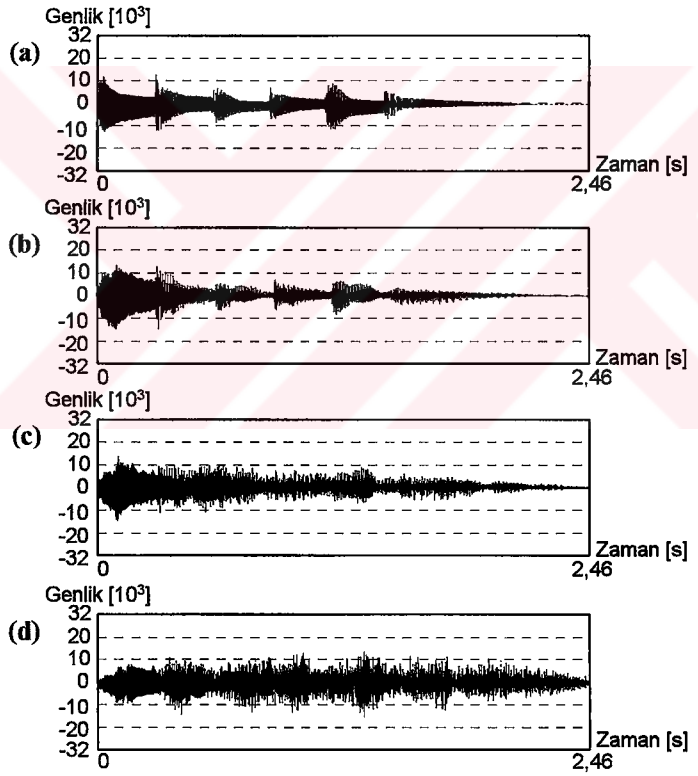
TÇY'da, ilk ulaşan yansımaların gerçeklenmesi için 19 çıkışlı bir sayısal geciktiriciden yararlanılmaktadır. Çizelge 4.4'de, geciktirici çıkışlarına ait geciktirme süreleri ve kazanç değerleri verilmiştir.

Çizelge 4.4 ÇÇSG katsayı değerleri ve geciktirme süreleri.

Geciktirici Çıkışı Sıra No.	Geciktirme Süresi $[s]$	Kazanç
0	0	1,00
1	0,0043	0,841
2	0,0215	0,504
3	0,0225	0,491
4	0,0268	0,379
5	0,0270	0,380
6	0,0298	0,346
7	0,0458	0,289
8	0,0485	0,272
9	0,0572	0,192
10	0,0587	0,193
11	0,0595	0,217
12	0,0612	0,181
13	0,0707	0,180
14	0,0708	0,181
15	0,0726	0,176
16	0,0741	0,142
17	0,0753	0,167
18	0,0797	0,134

Şekil 4.22'de görüldüğü gibi, TÇY'da çınlatıcı tarafından üretilen işaret basit bir yinelemeli alçak geçiren süzgeçten (Bölüm 4.2.3, Şekil 4.18) geçirilmektedir. Bu sayede, hem kaçınılmaz geciktirici çakışmaları nedeniyle oluşan ani sıçramaların etkisi yumuşatılmakta, hem de çıkışta elde edilen işaretin tonu üzerinde ayarlama yapma olanağı elde edilmektedir. AGS katsayılarının değerleri, GSİ yazılımında bulunan *"ton"* isimli görsel kaydırma çubuğu aracılığıyla kullanıcı tarafından belirlenebilmektedir.

Uygulama aşamasında, TÇY'da yer alan G_1 ve G_2 geciktiricilerine ait D_1 ve D_2 geciktirme sürelerine (Şekil 4.22) uygun değerler verilerek, yoğun çınlama bölgesinin çok çıkışlı sayısal geciktirici tarafından en son üretilen yankı işaretinin hemen ardından başlaması sağlanmaktadır.



Şekil 4.23 Farklı derecelerde çınlatma işlemleri: (a) özgün işaret; (b) kazanç = 0,75, derinlik = 1,0, ıslak/kuru oranı = 1,0; (c) kazanç = 0,83, derinlik = 1, ıslak/kuru oranı =2,0; (d) kazanç = 0,95, derinlik = 2, ıslak/kuru oranı = 3,0.

GSİ yazılımında *"ıslak / kuru oranı"* isimli görsel bir kaydırma çubuğu bulunmaktadır

(Bölüm 2.3.5). Kaydırma çubuğu yoluyla TÇY'da yer alan *ıslak* ve *kuru* çarpanlarının (Şekil 4.22) değerleri belirlenmektedir. Bu sayede, çınlatma işlemine sokulmuş işaret ile sayısal geciktiriciden alınan, çınlatıcıdan geçirilmemiş işaret kullanıcı tarafından seçilen bir oranda karıştırılabilmektedir. Sonuç olarak, kullanıcı, TÇY tarafından üretilen işaretteki çınlatma etkisinin derecesini yazılımsal olarak değiştirebilmektedir.

Şekil 4.23'de, GSİ yazılımında TÇY kullanılarak sayısal bir giriş işareti üzerinde oluşturulmuş farklı derecelerde çınlatma etkileri gösterilmiştir.

4.4 Süzgeçleme

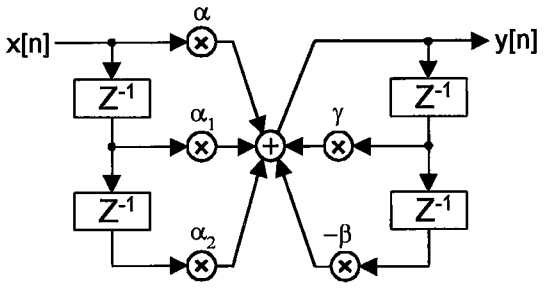
Müzikal işaretler üzerinde yapılan uygulamalarda genellikle ikinci dereceden süzgeçlerin kullanımı tercih edilmektedir. Daha dar bir iletim bandının elde edilmesini gerektiren durumlarda ise, ikinci dereceden süzgeçlerin kaskad bağlanması ile oluşturulmuş daha yüksek dereceli yapılardan yararlanılmaktadır. Bu yöntem sayesinde, süzgeç karakteristiğinin katsayılara aşırı hassas bir biçimde bağımlı olması engellenmekte ve sayısal taşmaların yol açabileceği bozulmaların önüne geçilmektedir.

GSİ yazılımında, süzgeçleme işlemi için *"basit parametrik süzgeç"* ve *"müzikal süzgeç"* adı altında iki farklı yinelemeli süzgeç (Bölüm 2.1.5) seçeneği bulunmaktadır. Süzgeçler, belirli sayıda parametre ile denetlenen genel amaçlı yapılar biçimindedir. Süzgeç parametrelerinin değerleri kapalı form bağıntılar üzerinden belirlenmektedir. Yapıların frekans yanıtları, parametrelere atanan değerler yoluyla, kullanıcı tanımlı bir biçimde değiştirilebilmektedir. Bu sayede, tek bir genel amaçlı yapı ile farklı süzgeçleme karakteristikleri elde edilebilmektedir. Süzgeç yapılarının gerçeklenmesinde Lane vd. (1997) ile Dattorro'nun (1997) tasarımlarından yararlanılmıştır.

4.4.1 Basit parametrik süzgeç

Şekil 4.24'de, basit parametrik süzgeç (BPS) yapısı gösterilmiştir. BPS genel amaçlı bir yapı biçiminde tasarlanmıştır. Alçak geçiren, yüksek geçiren, bant geçiren ve bant sönümlendiren

karakteristiklerde frekans yanıtları, süzgeç parametrelerine uygun değerlerin atanması yoluyla elde edilebilmektedir.



Şekil 4.24 Basit parametrik süzgeç yapısı.

Çizelge 4.5'de, farklı süzgeç karakteristikleri için, BPS parametrelerinin elde edilmesinde kullanılan kapalı form bağıntılar verilmiştir (Lane, 1990).

Çizelge 4.5 Farklı karakteristikler için BPS parametre değerleri.

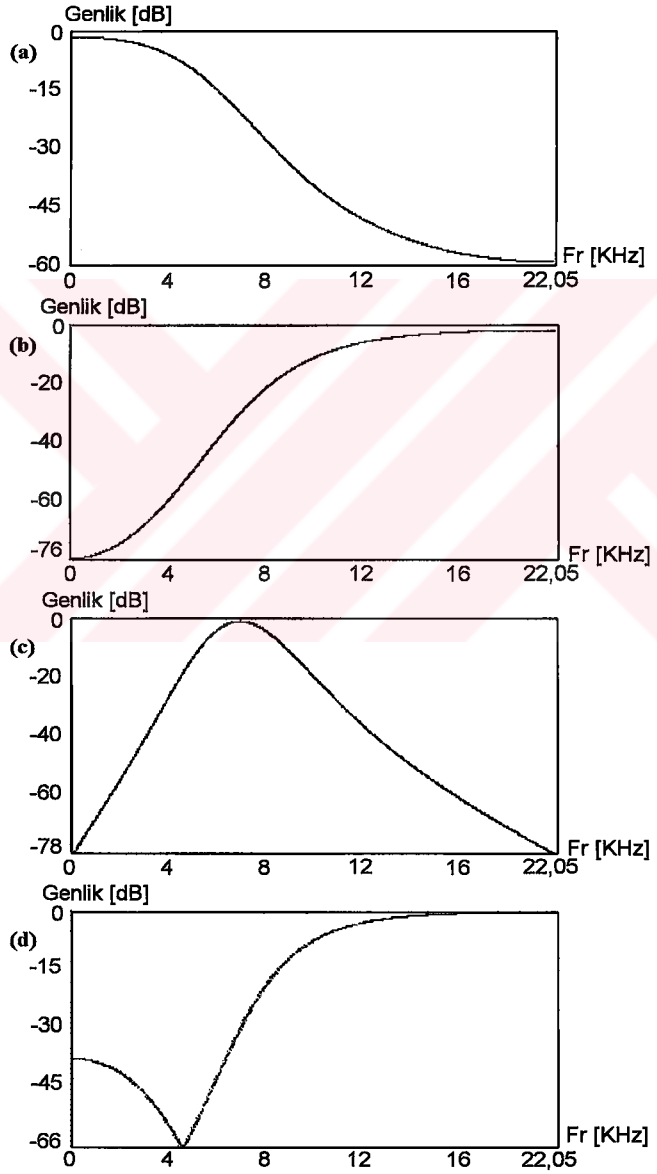
Katsayı	Alçak Geçiren	Yüksek Geçiren	Bant Geçiren	Bant Sönümlendiren
α	$\frac{1}{4}\left(\frac{1}{2}+\beta-\gamma\right)$	$\frac{1}{4}\left(\frac{1}{2}+\beta+\gamma\right)$	$\frac{1}{2}\left(\frac{1}{2}-\beta\right)$	$\frac{1}{2}\left(\frac{1}{2}+\beta\right)$
α_1	2α	-2α	0	$-\gamma$
α_2	α	α	$-\alpha$	α
β	$\frac{1}{2}\cdot\frac{1-\frac{d}{2}\sin\theta_c}{1+\frac{d}{2}\sin\theta_c}$	$\frac{1}{2}\cdot\frac{1-\frac{d}{2}\sin\theta_c}{1+\frac{d}{2}\sin\theta_c}$	$\frac{1}{2}\cdot\frac{1-\tan\left(\frac{\theta_0}{2Q}\right)}{1+\tan\left(\frac{\theta_0}{2Q}\right)}$	$\frac{1}{2}\cdot\frac{1-\tan\left(\frac{\theta_0}{2Q}\right)}{1+\tan\left(\frac{\theta_0}{2Q}\right)}$
γ	$\left(\frac{1}{2}+\beta\right)\cos\theta_c$	$\left(\frac{1}{2}+\beta\right)\cos\theta_c$	$\left(\frac{1}{2}+\beta\right)\cos\theta_0$	$\left(\frac{1}{2}+\beta\right)\cos\theta_0$

Bağıntılarda, alçak geçiren ve yüksek geçiren karakteristikler için, kesim frekansı ve sönüm faktörü, sırasıyla, f_c ve d terimleri ile ifade edilmiştir. Bant geçiren ve bant sönümlendiren karakteristikler için ise, merkez frekansı ve kalite faktörünün tanımlanmasında, sırasıyla, f_o

ve Q terimleri kullanılmıştır. Normalize frekans değerleri θ_c ve θ_0 terimleri ile gösterilmekte ve,

$$\theta = 2\pi f / F_s \tag{4.35}$$

bağıntısı ile hesaplanmaktadır. Eşitlikte yer alan F_s terimi, örnekleme hızını ifade etmektedir.



Şekil 4.25 Farklı karakteristikler için BPS frekans yanıtları: (a) alçak geçiren; (b) yüksek geçiren; (c) bant geçiren; (d) bant sönümlendiren.

GSİ yazılımında, genel amaçlı yapıya ait her bir süzgeç seçeneği iki ayrı görsel kaydırma çubuğu kullanılarak denetlenmektedir. Kaydırma çubukları aracılığı ile, kesim (veya merkez) frekansı ile sönüm (veya kalite) faktörü değerleri, kullanıcı tarafından, istenilen değerlerde seçilebilmektedir.

Şekil 4.25'de, 44100 Hz örnekleme hızı için, GSİ yazılımında BPS parametre değerleri değiştirilerek elde edilen farklı karakteristikte süzgeçleme işlemlerine ait frekans yanıtları gösterilmiştir. Frekans yanıtlarının çiziminde, kesim ve merkez frekansları 5000 Hz, sönüm ve kalite faktörleri ise 1,414 değerlerinde seçilmiştir.

4.4.2 Müzikal süzgeç

Müzikal süzgeç (MS), temel anlamıyla, bant geçiren ve bant sönümlendiren karakteristiklerde frekans yanıtlarının elde edilebildiği genel amaçlı bir sayısal yapıdır. Genel amaçlı yapı ikinci dereceden bir tüm geçiren süzgeç kullanılarak tasarlanmıştır ve belirli sayıda parametre üzerinden kullanıcı tanımlı olarak denetlenebilmektedir. Parametre değerleri kapalı form bağıntılar yoluyla hesaplanmaktadır. MS'in bant geçiren ve bant sönümlendiren seçenekleri ile, oldukça dar bir iletim bandına sahip, yüksek seçicilikte geçirme ve sönümlendirme işlemleri gerçeklenebilmektedir. Bu nedenle, bant sönümlendiren karakteristikli süzgeç seçeneği kesici süzgeç, bant geçiren karakteristikli süzgeç seçeneği ise rezonatör olarak adlandırılmıştır.

4.4.2.1 Q kalite faktörü

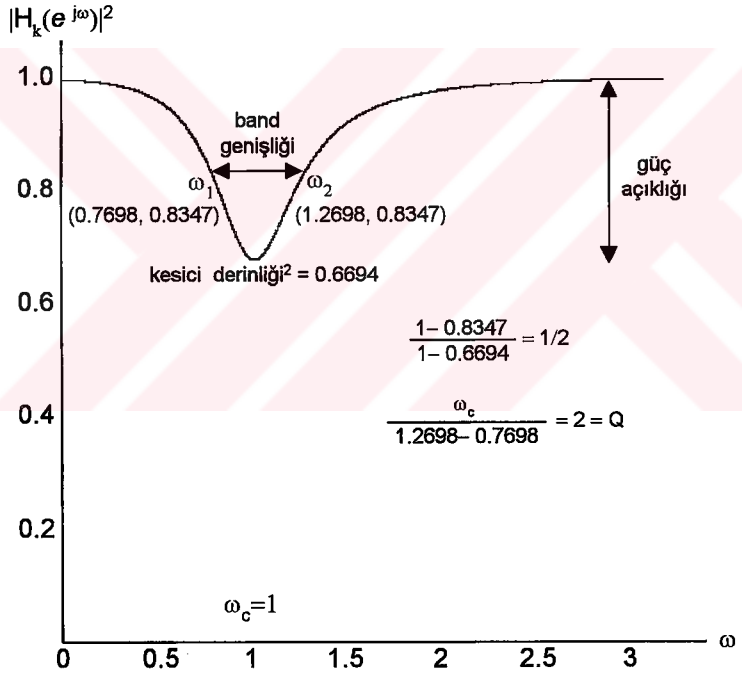
Bant geçiren veya bant sönümlendiren karakteristikte bir süzgece ait kalite faktörü, merkez frekansının, bant genişliğine bölümünden hesaplanmaktadır (4.36).

$$Q = \frac{\omega_c}{\omega_2 - \omega_1} \tag{4.36}$$

(4.36) eşitliğinde, frekans değerleri,

$$\omega = 2\pi f \tag{4.37}$$

bağıntısı uyarınca, açısal biçimde ifade edilmiştir. Kesim frekansları ω_1 ve ω_2, genel olarak, süzgece ait güç spektrumunda, gücün, tepe değerinin yarısına $(10\log_{10}(1/2) = -3{,}01\,\text{dB}$ altına) eşit olduğu noktalardaki frekans değerleri olarak tanımlanmaktadır. Bununla birlikte, müzikal işaret işleme amaçlı süzgeçlere ait güç spektrumlarında, gücün, tepe değerinin yarısına kadar inmediği durumlarla sıklıkla karşılaşılmaktadır. Bu nedenle, bu türden, dar bir güç açıklığına sahip süzgeçler için kesim frekanslarının farklı bir mantığa göre hesaplanması gerekmektedir. Güç spektrumunda, gücün tepe değeri ile en düşük seviyesi arasındaki fark güç açıklığı olarak adlandırılmaktadır. Müzikal bir süzgecin kesim frekansları, gücün, güç açıklığının orta noktasına eşit olduğu frekans değerleri olarak tanımlanmaktadır.



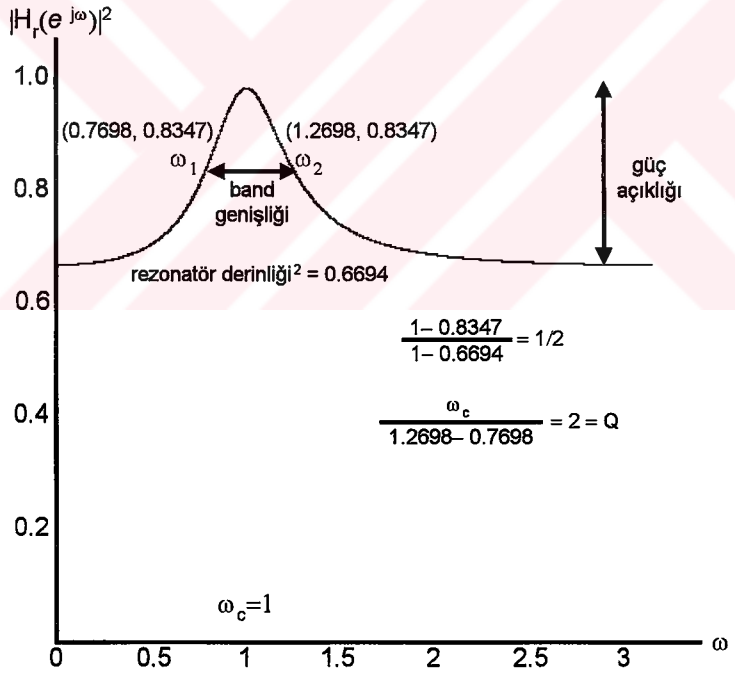
Şekil 4.26 Kesici süzgeç güç spektrumu.

Şekil 4.26'da, kesici bir süzgece ait güç spektrumu gösterilmiştir. Güç spektrumu, tepe değerine $z = \pm 1$ ($\omega = 0$ ve $\omega = \pi$) noktalarında asimptotik olarak yaklaşmaktadır. Merkez frekansında ise, güç spektrumu en düşük değerini almaktadır. Süzgecin ω_c merkez frekansı

1,0 değerinde seçilmiştir. Kesim frekansları, müzikal süzgeçler için yapılan tanımlama uyarınca,

$$\frac{1-\left|H_k\left(e^{j\omega}\right)\right|^2}{1-\left|H_k\left(e^{j\omega_c}\right)\right|^2}=\frac{1}{2} \tag{4.38}$$

bağıntısı üzerinden hesaplanmaktadır. Eşitliği sağlayan ω değerleri, ω_1 ve ω_2 kesim frekanslarını vermektedir. Süzgece ait kalite faktörü (4.36) 2,0 değerine eşittir. Şekil 4.26'da görüldüğü gibi, $\left|H_k\left(e^{j\omega_c}\right)\right|^2=0$ değerini aldığında, kesici süzgeç güç spektrumu, çentik (notch) süzgeç güç spektrumu ile tam bir benzerlik göstermektedir. Bu durumda, kesim frekanslarını hesaplamakta kullanılan bağıntı (4.38) ile geleneksel bağıntı arasındaki fark ortadan kalkmaktadır.



Şekil 4.27 Rezonatör güç spektrumu.

Yapılan tanımlama, güç spektrumu Şekil 4.27'de gösterilen rezonatör karakteristikli müzikal süzgeç için de geçerlidir. Rezonatör, kesici süzgecin aksine, $z=\pm 1$ noktalarında en düşük

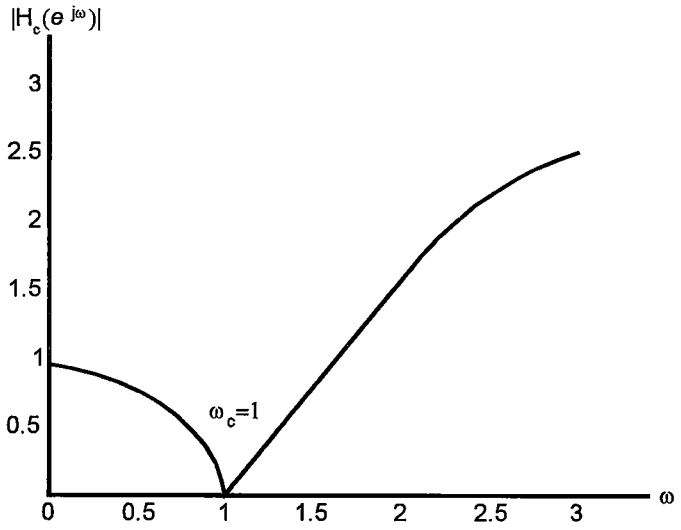
güç seviyesine asimptotik olarak yaklaşmakta, ω_c merkez frekansında ise tepe değerini almaktadır. Bu nedenle, rezonatör kesim frekansları ω_1 ve ω_2,

$$\frac{1-\left|H_r\left(e^{j\omega}\right)\right|^2}{1-\left|H_r\left(\pm 1\right)\right|^2}=\frac{1}{2} \tag{4.39}$$

eşitliği üzerinden hesaplanmaktadır. $\left|H_r\left(\pm 1\right)\right|^2 = 0$ değerini aldığında, mükemmel rezonatör karakteristiği elde edilmektedir. Bu durumda, kesici süzgeçten çentik süzgece geçişte olduğu gibi, kesim frekanslarını hesaplamakta kullanılan bağıntı (4.39) ile geleneksel bağıntı arasındaki fark ortadan kalkmaktadır.

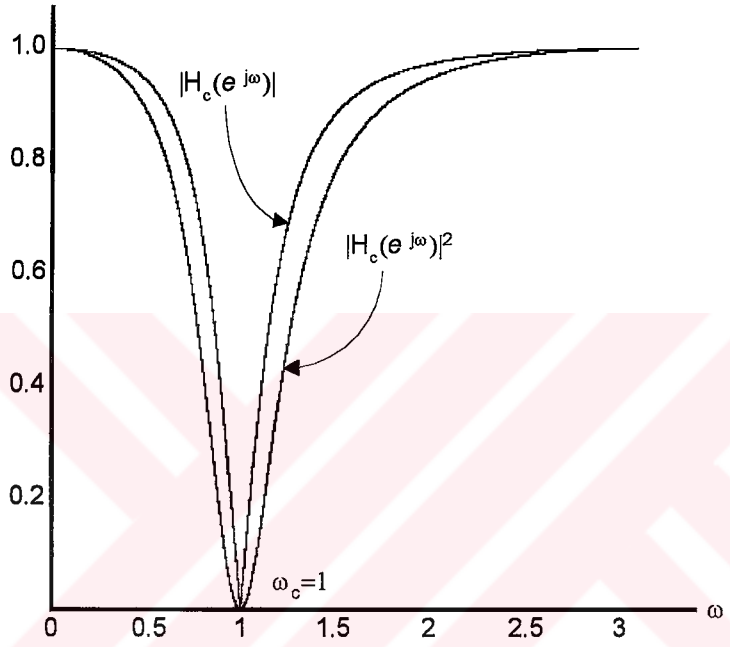
4.4.2.2 Çentik süzgeç

Çentik süzgeç frekans yanıtında, genliğin belirli bir frekans değeri için sıfıra eşit olması, diğer tüm frekans değerlerinde ise sabit bir seviyede kalması istenmektedir. Yalnızca sıfırları olan (hiç kutbu olmayan) bir geçiş fonksiyonuna sahip bir süzgeç ile, frekans yanıtında genliğin belirli bir frekans değeri için sıfıra çekilmesi sağlanmakla birlikte, diğer tüm frekans değerleri için genlik sabit bir seviyede tutulamamaktadır.



Şekil 4.28 Basit yapıda bir çentik süzgecin frekans yanıtı.

Şekil 4.28'de, bu türden bir çentik süzgecin frekans yanıtı gösterilmiştir. Süzgecin, $z = 0$ noktasında çift katlı, keyfi olarak seçilmiş bir kutbu ve $\omega = 1$ noktasında bir adet sıfırı bulunmaktadır. Şekilde görüldüğü gibi, yüksek frekanslarda kazanç aşırı miktarda artmaktadır. Bu durum, süzgeç çıkışında üretilen işaret üzerinde bozucu bir etki yaratmaktadır. Ayrıca, süzgeç sıfırının farklı bir değere taşınması frekans yanıtını düzensiz bir biçimde değiştirmektedir. Sonuç olarak, bu yapıda bir süzgeç müzikal işaret işleme uygulamalarında doyurucu bir sonuç vermemektedir.



Şekil 4.29 Çentik süzgeç frekans ve güç spektrumları.

Belirli bir frekans bileşeninin yüksek bir seçicilikle bastırılması ve diğer tüm frekanslarda spektrumun sabit bir genlikte tutulması için, çentik süzgeç geçiş fonksiyonuna keyfi olmayan, bilinçli olarak seçilmiş kutupların eklenmesi gerekmektedir (Regalia ve Mitra, 1987). Bu tür bir çentik süzgecin geçiş fonksiyonu,

$$H_c(z) = \frac{1}{2}(1+\beta)\frac{1+2\gamma z^{-1}+z^{-2}}{1+\gamma(1+\beta)z^{-1}+\beta z^{-2}} \tag{4.40}$$

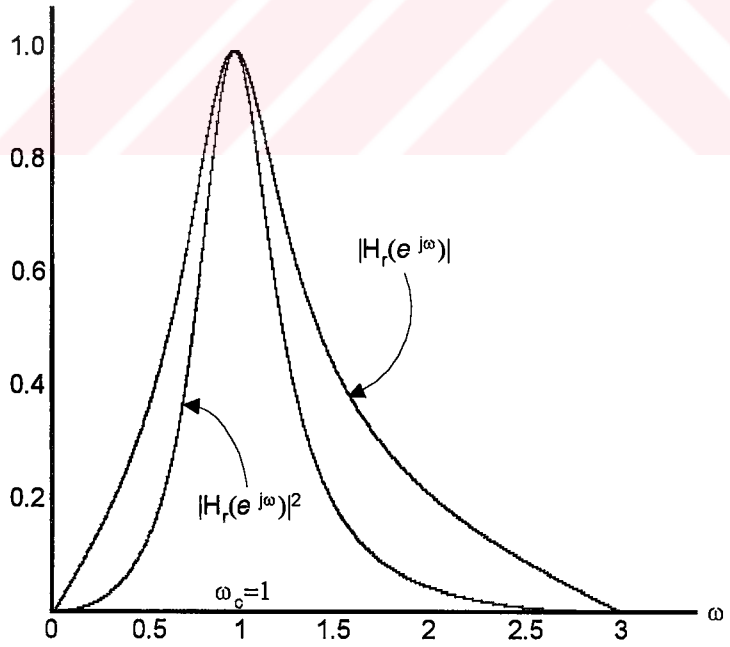
eşitliği ile verilmektedir. Süzgece ait frekans ve güç spektrumları Şekil 4.29'da

gösterilmiştir. Şekilde görüldüğü gibi, genlik (veya güç), ω_c merkez frekansında sıfıra eşit olmakta ve diğer tüm frekanslarda, merkez frekansının değerinden bağımsız olarak sabit birim seviyede kalmaktadır.

Kesici süzgeç frekans yanıtında genlik, çentik süzgeçten farklı olarak, merkez frekansında sıfırdan büyük bir seviyeye çekilmektedir. Bir başka deyişle, kesici süzgeç çentik süzgece göre daha dar bir güç açıklığına sahiptir. MS yapısında, kesici derinliği (Şekil 4.26) belirli sayıda parametre üzerinden, kullanıcı tanımlı olarak belirlenebilmektedir.

4.4.2.3 Rezonatör

Basit bir rezonatör yapı, yalnızca kutupları olan (hiç sıfırı olmayan) bir geçiş fonksiyonuna sahip bir süzgeç kullanılarak oluşturulabilmektedir. Fakat bu tür bir tasarımda, yalnızca sıfırları olan bir çentik süzgeç ile gerçekleştirilen uygulamalarda karşılaşılanlara benzer sorunlar ortaya çıkmaktadır. Bu sorunların önüne geçmek için, rezonatör geçiş fonksiyonuna $z = \pm 1$ noktalarına yerleştirilmiş iki adet sıfır eklenmektedir (Jackson, 1996).



Şekil 4.30 Rezonatör frekans ve güç spektrumları.

Rezonatör geçiş fonksiyonu,

$$H_r(z) = \frac{1}{2}(1-\beta)\frac{1-z^{-2}}{1+\gamma(1+\beta)z^{-1}+\beta z^{-2}} \tag{4.41}$$

eşitliği ile ifade edilmektedir. Süzgece ait frekans ve güç spektrumları Şekil 4.30'da gösterilmiştir. Şekilde görüldüğü gibi, genlik (veya güç), ω_c merkez frekansında tepe değerine ulaşmaktadır. Tepe değeri, merkez frekansından bağımsız olarak daima birim seviyeye eşit olmaktadır. Geçiş fonksiyonundaki sıfırlar nedeniyle, spektrum $z = \pm 1$ ($\omega = 0$ ve $\omega = \pi$) noktalarında sıfır değerine ulaşmaktadır.

Frekans yanıtında, genliğin $\omega = 0$ ve $\omega = \pi$ frekanslarında sıfıra eşit olduğu durum, mükemmel rezonatör durumu olarak adlandırılmaktadır. Rezonatör frekans yanıtında ise, genlik, sınır frekanslarında sıfırdan büyük bir seviyeye çekilmektedir. Bir başka deyişle, rezonatör, mükemmel rezonatöre göre daha dar bir güç açıklığına sahiptir. MS yapısında, rezonatör derinliği (Şekil 4.27) belirli sayıda parametre üzerinden, kullanıcı tanımlı olarak belirlenebilmektedir.

4.4.2.4 Müzikal süzgeç yapısı

Sırasıyla (4.40) ve (4.41) eşitlikleri ile verilen çentik süzgeç ve rezonatör geçiş fonksiyonları ortak özellikler taşımaktadır. Eşitliklerde görüldüğü gibi, fonksiyon paydaları (kutupları) özdeştir. Ayrıca, geçiş fonksiyonlarının her ikisinde de, ω_c ve Q müzikal süzgeç parametreleri ile β ve γ katsayıları arasında basit bir ilişki kurulabilmektedir. Bu özellikler, her iki süzgeç karakteristiğinin de tek bir genel amaçlı yapı kullanılarak elde edilmesini olası hale getirmektedir. MS yapısı bu düşünce doğrultusunda geliştirilmiştir. Yapıya ait parametrelere uygun değerler verilerek farklı karakteristiklerde süzgeçleme işlemleri gerçekleştirilebilmektedir.

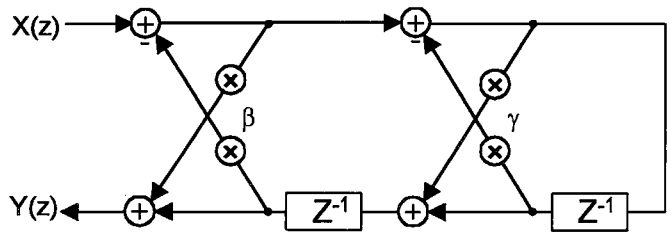
Şekil 4.31'de, ikinci dereceden bir tüm geçiren süzgeç yapısı gösterilmiştir. Süzgecin geçiş fonksiyonu,

$$T(z)=\frac{Y(z)}{X(z)}=\frac{\beta+\gamma(1+\beta)z^{-1}+z^{-2}}{1+\gamma(1+\beta)z^{-1}+\beta z^{-2}} \tag{4.42}$$

eşitliği ile verilmektedir. Süzgeç karakteristiğinin temel özellikleri ise,

$$\left|T\left(e^{j\omega}\right)\right|=1, \qquad T(\pm 1)=1, \qquad T\left(e^{j\omega_c}\right)=-1 \tag{4.43}$$

bağıntıları ile ifade edilmektedir.



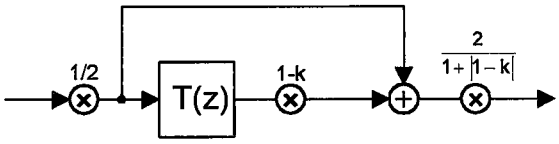
Şekil 4.31 İkinci dereceden tüm geçiren süzgeç yapısı.

(4.42) eşitliği incelendiğinde, tüm geçiren süzgeç geçiş fonksiyonunun da, çentik süzgeç (4.40) ve rezonatör (4.41) geçiş fonksiyonları ile özdeş bir paydaya (kutuplara) sahip olduğu görülmektedir. Çentik süzgeç ve rezonator geçiş fonksiyonları,

$$H_c(z)=\frac{1+T(z)}{2} \tag{4.44}$$

$$H_r(z)=\frac{1-T(z)}{2} \tag{4.45}$$

eşitlikleri yoluyla tüm geçiren süzgeç geçiş fonksiyonundan türetilebilmektedir. Bu özellikten yararlanılarak, ikinci dereceden bir tüm geçiren süzgecin kullanıldığı genel amaçlı bir yapı aracılığıyla, hem çentik süzgeç, hem de rezonatör karakteristikleri elde edilebilmektedir. Şekil 4.32'de, GSİ yazılımında kullanılan MS genel amaçlı yapısı gösterilmiştir.



Şekil 4.32 Müzikal süzgeç yapısı.

MS yapısında yer alan k katsayısına uygun değerler verilerek farklı karakteristiklerde süzgeçleme işlemleri gerçekleştirilebilmektedir. GSİ yazılımında, k katsayısına görsel bir kaydırma çubuğu aracılığıyla kullanıcı tanımlı olarak değer atanabilmektedir. Çizelge 4.6'da, farklı karakteristikte süzgeçleme işlemleri için k katsayısının alması gereken değerler verilmiştir.

Çizelge 4.6 Müzikal süzgeç k katsayısı değerleri.

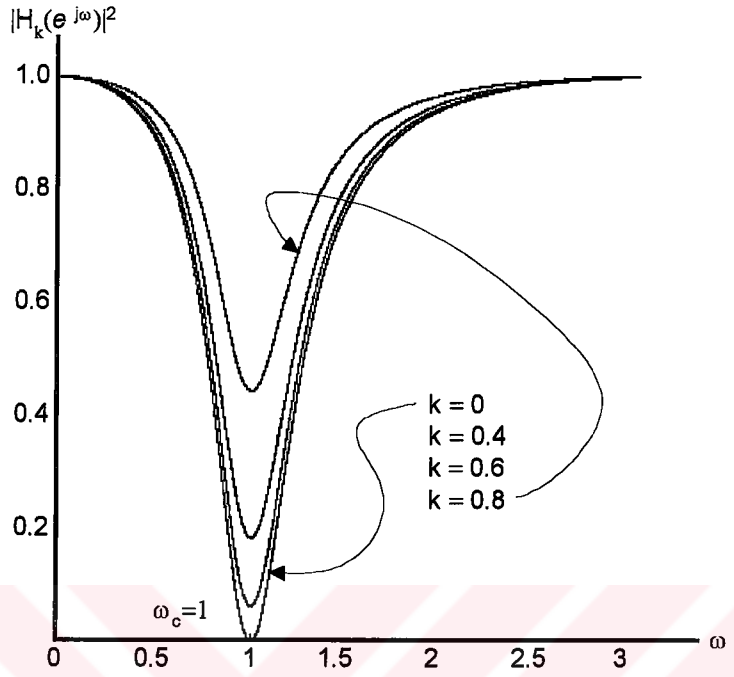
k katsayısı	MS karakteristiği
$k = 0$	Çentik süzgeç
$0 < k < 1$	Kesici süzgeç
$k = 1$	Giriş işareti (süzgeçleme yok)
$1 < k < 2$	Rezonatör
$k = 2$	Mükemmel rezonatör

Çizelge 4.6'da verildiği gibi, k katsayısının 0 ile 1 aralığında bir değer alması durumunda kesici süzgeç karakteristiği elde edilmektedir. Kesici derinliği (Şekil 4.26), (4.43) ve (4.46) eşitliklerinden çıkarılan,

$$|\textit{kesici derinliği}| = \frac{1-(1-k)}{1+|1-k|} = \frac{k}{2-k} \qquad ; 0 \le k < 1 \tag{4.48}$$

bağıntısı üzerinden, k katsayısınının değerine göre, merkez frekansından bağımsız olarak belirlenebilmektedir. (4.48) eşitliğinde açıkça görüldüğü gibi, $k = 0$ seçilmesi durumunda, kesici derinliği de sıfır değerini almakta ve kesici süzgeç karakteristiği çentik süzgeç karakteristiğine dönüşmektedir. Şekil 4.33'de, farklı k katsayı değerleri için, GSİ yazılımında

MS genel amaçlı yapısı kullanılarak elde edilen kesici süzgeç karakteristiklerine ilişkin güç spektrumları gösterilmiştir.

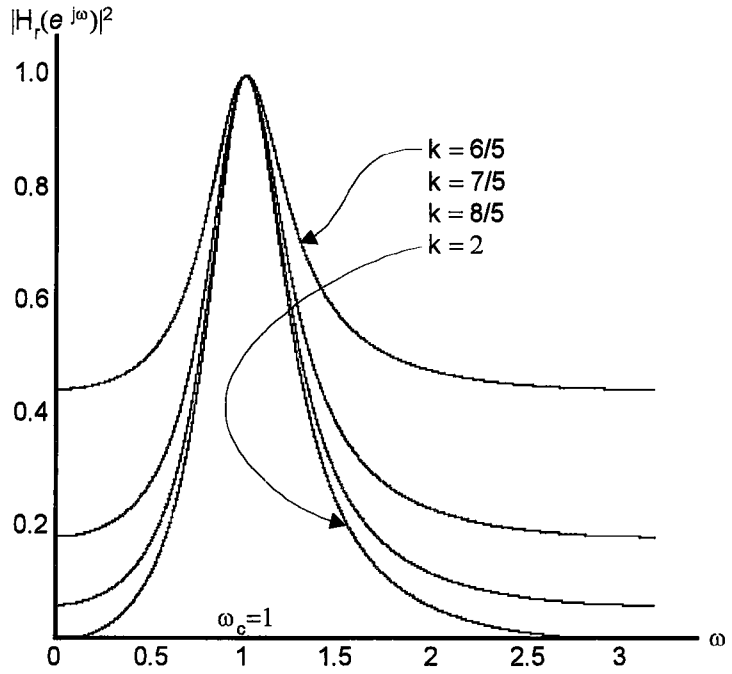


Şekil 4.33 Farklı k değerleri için kesici süzgeç güç spektrumları; $Q = 2$.

MS yapısında yer alan k katsayısının 1 ile 2 aralığında bir değer alması durumunda ise rezonatör karakteristiği elde edilmektedir. Rezonatör derinliği (Şekil 4.27), (4.43) ve (4.46) eşitliklerinden çıkarılan,

$$\left|\textit{rezonatör derinliği}\right| = \frac{1+(1-k)}{1+|1-k|} = \frac{2-k}{k} \qquad ;1 < k \le 2 \tag{4.49}$$

bağıntısı üzerinden, k katsayısınının değerine göre, merkez frekansından bağımsız olarak belirlenebilmektedir. (4.49) eşitliğinde açıkça görüldüğü gibi, $k = 2$ seçilmesi durumunda rezonatör derinliği sıfır değerini almakta ve rezonatör karakteristiği mükemmel rezonatör karakteristiğine dönüşmektedir. Şekil 4.34'de, farklı k katsayı değerleri için, GSİ yazılımında MS genel amaçlı yapısı kullanılarak elde edilen rezonatör karakteristiklerine ilişkin güç spektrumları gösterilmiştir.



Şekil 4.34 Farklı k değerleri için rezonatör güç spektrumları; $Q = 2$.

Tüm süzgeç karakteristikleri için, ω_c merkez frekansı değeri, (4.43) eşitliğinden yararlanılarak,

$$\omega_c = \arccos(-\gamma) \tag{4.46}$$

bağıntısı yoluyla hesaplanmaktadır. GSİ yazılımında, merkez frekansı değeri, görsel bir kaydırma çubuğu aracılığıyla kullanıcı tanımlı olarak seçilmekte ve tüm geçiren süzgeç γ katsayısı, istenilen merkez frekansına bağlı olarak,

$$\gamma = -\cos(\omega_c) \tag{4.47}$$

eşitliği üzerinden hesaplanmaktadır.

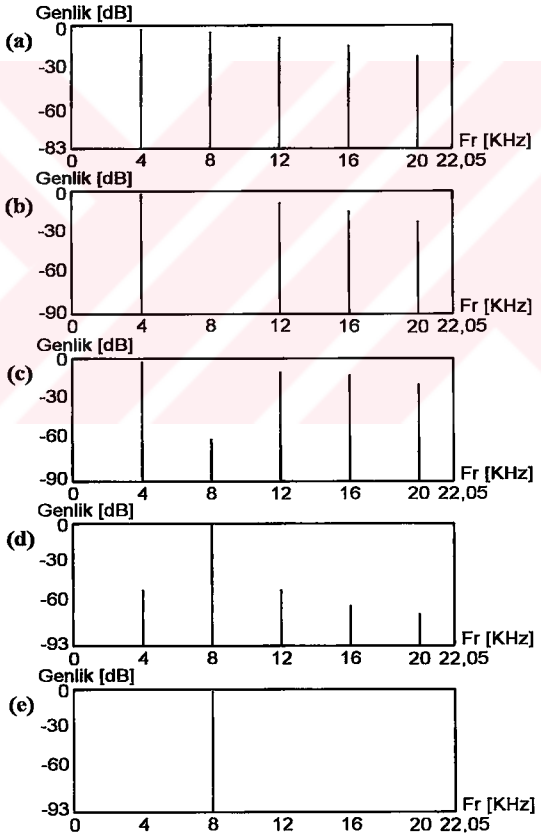
ω_1 ve ω_2 müzikal kesim frekanslarının değerleri, MS yapısı ile gerçeklenebilen tüm süzgeç karakteristikleri için,

$$\cos(\omega_{2,1}) = \frac{(1+\beta)^2 \cos(\omega_c) +,- (\beta-1)\sqrt{2(1+\beta^2) - (1+\beta)^2 \cos^2(\omega_c)}}{2(1+\beta^2)} \tag{4.50}$$

eşitliği yoluyla hesaplanmaktadır. Tüm geçiren süzgeç β katsayısının değeri ise, ω_c merkez frekansı ve Q kalite faktörüne (4.36) bağlı olarak,

$$\beta = \frac{1-\tan(\omega_c/(2Q))}{1+\tan(\omega_c/(2Q))} = \frac{1-\tan(\Delta\omega/2)}{1+\tan(\Delta\omega/2)} \tag{4.51}$$

bağıntısı ile belirlenmektedir. GSİ yazılımında, görsel bir kaydırma çubuğu aracılığıyla, Q kalite faktörünün değeri kullanıcı tanımlı olarak seçilebilmektedir.

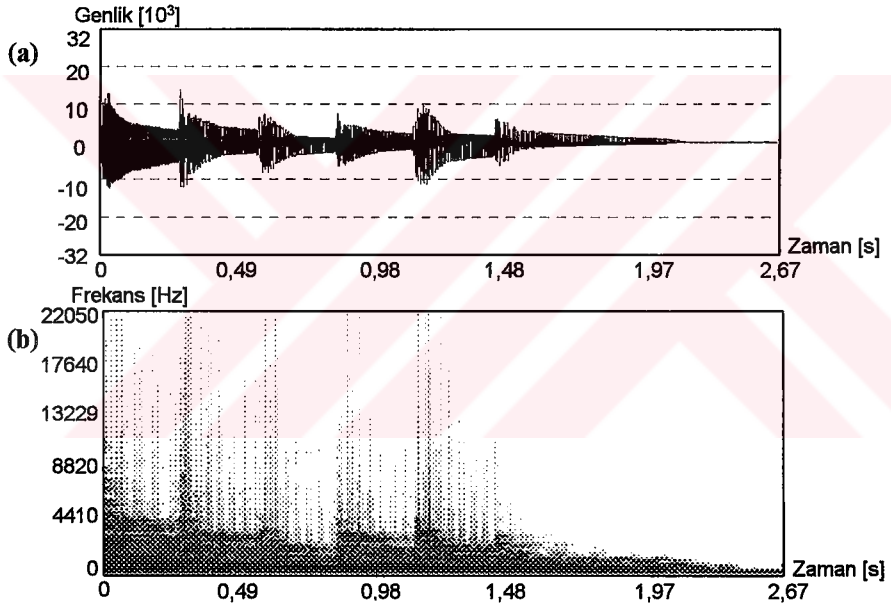


Şekil 4.35 MS kullanılarak geçekleştirilen farklı karakteristiklerde süzgeçleme işlemleri: (a) özgün işaret; (b) çentik süzgeç çıkışı, $Q = 20$, $k = 0$; (c) kesici süzgeç çıkışı, $Q = 2{,}5$, $k = 0{,}5$; (d) rezonatör çıkışı, $Q = 2{,}5$, $k = 1{,}5$; (e) mükemmel rezonatör çıkışı, $Q = 20$, $k = 2$.

Şekil 4.35'de, GSİ yazılımında MS aracılığıyla, sayısal bir giriş işareti üzerinde gerçekleştirilen farklı karakteristiklerde süzgeçleme işlemleri gösterilmiştir. Giriş işareti, 4 KHz, 8KHz, 12 KHz, 16 KHz ve 20 KHz frekanslı sinüzoidal bileşenlerin toplamından oluşmaktadır. Tüm süzgeçleme işlemleri için, merkez frekansı 8 KHz değerinde seçilmiştir.

4.5 Gösterim

Şekil 4.36'da verilen örnekte görüldüğü gibi, GSİ yazılımında, PCM ses dalgası dosyasından okunan sayısal işaret kullanıcı tarafından iki farklı biçimde görüntülenebilmektedir. Her iki gösterim seçeneği de, işaretin kanal sayısı, örnekleme hızı vb. özelliklerine esnek bir biçimde uyum sağlayacak şekilde gerçekleştirilmektedir.



Şekil 4.36 (a) Zamanda gösterim; (b) spektral gösterim, örnekleme hızı = 44100 Hz.

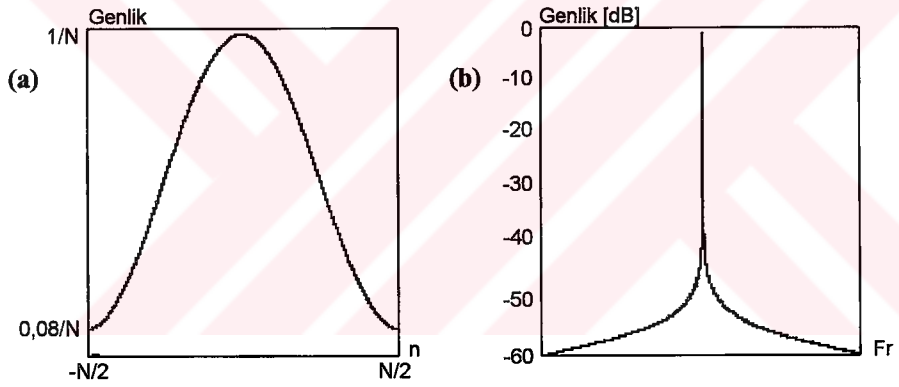
Zamanda gösterim, sayısal ses işareti örnek değerlerinin zaman ekseninde değişimini vermektedir. Spektral gösterimde ise, ses işareti KSFD (Bölüm 3) işleminden geçirilerek, işarete ait frekans bileşenlerinin zamana bağlı değişimi elde edilmektedir (Şekil 4.36).

KSFD işlemi pencereleme (Bölüm 3.1) aşamasında, Hamming standart penceresi

kullanılmaktadır (Şekil 4.37). GSİ arabiriminde yer alan gösterge (display) yüksekliğine bağlı olarak, 256 örnek uzunluklu (Bölüm 3.1.1) normalize bir pencere ile işlem yapılmaktadır. Sayısal giriş işareti $x[n]$, 256 örneklik paketler halinde pencereleme işlemine sokulmaktadır. Her bir paket için, pencereleme işlemi sonucunda elde edilen pencerelenmiş parçalara ait örnek değerleri,

$$p[n] = \frac{0{,}54 + 0{,}46\cos(2\pi(n-128)/256)}{256} x[n] \qquad n = 0,1,2,\ldots 255 \qquad (4.52)$$

eşitliği uyarınca hesaplanmaktadır. KSFD'nün ikinci aşamasında, her bir pencerelenmiş parçaya ait frekans bileşenleri HFD (Bölüm 3.2) yoluyla bulunmaktadır. HFD hesabı, kelebek yapının (Bölüm 3.2.2) kullanıldığı, yinelemeli bir algoritma aracılığıyla gerçekleştirilmektedir.



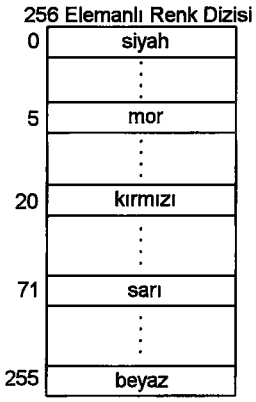
Şekil 4.37 (a) $N = 256$ örnek uzunluklu normalize Hamming penceresi; (b) pencereye ait frekans spektrumu.

KSFD işleminde amaç, HFD ile elde edilen çerçeveleri yan yana dizerek, işarete ait frekans bileşenlerinin zamanda değişimini görüntülemektir. GSİ yazılımında, her bir frekans bileşeninin genliğine, uygun bir renk değeri karşılık düşürülmektedir. Bu sayede, zaman, frekans, genlik eksenlerinde, üç boyutlu bir spektral gösterim oluşturulmaktadır.

Frekans bileşenleri genliklerinin farklı renk değerleri ile eşleştirilmesi aşamasında, 256 elemanlı bir renk dizisinden yararlanılmaktadır (Şekil 4.38). Eşleştirme işleminden önce, frekans bileşenlerinin 0 ila 255 arasında genlik değerleri alması için, her bir genlik değeri,

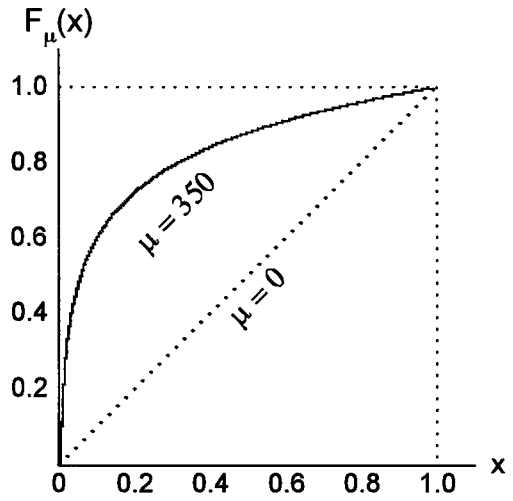
$$k = 256/V_M \tag{4.53}$$

katsayısı ile çarpılmaktadır. Eşitlikte yer alan V_M terimi, giriş işaretine ait olası en büyük örnek değerini ifade etmektedir. Bu değer, işareti oluşturan örneklerin kaçar bit uzunluklu olduğuna bağlı olarak belirlenmektedir (Ek 1).



Şekil 4.38 Spektral gösterimde kullanılan 256 elemanlı renk dizisi

Genlik değerlerinin, 0 ile 255 arasında düzgün (uniform) olmayan bir dağılım gösterdiği saptanmıştır. Düzgün olmayan bu dağılım, gösterimde kesiklikler ve atlamalar oluşmasına yol açarak, genlik değerlerinin farklı renk seviyeleri ile ifade edilmesinde sorunlar yaratmaktadır.



Şekil 4.39 μ eğrisi.

Bu sorunların önüne geçmek için, frekans bileşenlerine ait genlik değerleri μ eğrisi ile çarpılarak,

$$F_{\mu}(x) = Sgn(x)\frac{1+\ln(\mu|x|)}{\ln(1+\mu)} \tag{4.54}$$

eşitliği uyarınca (Kayran vd., 1992), düzgün olmayan kuantalama işleminden geçirilmektedir (Şekil 4.39).

5. SONUÇLAR

Bölüm 4'de, GSİ yazılımına ait her bir işlev için, gerçekleştirilen örnek bir uygulama sonucunda elde edilen çıkış işaretleri gösterilmiştir. Farklı nitelikte sayısal ses işaretleri üzerinde yapılan denemeler sonucunda, tasarlanan yazılımın, aşağıda sıralanan bazı özel durumlar dışında görevini sorunsuz bir biçimde yerine getirdiği saptanmıştır.

- Yüksek sıkıştırma oranları seçilerek gerçekleştirilen sıkıştırma (Bölüm 4.1.3) işlemlerinde, sayısal ses işareti üzerinde bir kırpma (Bölüm 4.1.5) yan etkisi ortaya çıkmaktadır.
- GEÜ genel amaçlı yapısı (Bölüm 4.2.3), dalgalandırma ve yankı etkileri oluşturulurken pozitif geri beslemeli olarak kullanılmaktadır. Bu durum, yüksek dinamik değere veya ani geçişlere sahip işaretlerle yapılan kimi uygulamalarda, çıkışta sayısal bir taşma oluşmasına yol açabilmektedir.
- GEÜ ile gerçekleştirilen koro etkisi uygulamalarında, uygun katsayı ve geciktirme süresi değerleri seçilerek yapının davranışının tüm geçiren süzgece benzetilmesine çalışılmaktadır. Bu yolla, sayısal ses işaretine ait farklı frekans bileşenlerinin farklı değerlerde geciktirilmesi sağlanmaktadır. Harmonik işaretlerle yapılan kimi uygulamalarda, GEÜ parametreleri hassas biçimde ayarlanmadığı takdirde işaret frekans bileşenleri üzerinde aşırı bir öteleme oluşturulabilmektedir. Bu durum, işaret üzerinde koro etkisi sınırlarının dışında, abartılı bir değişim yaratılmasına yol açabilmektedir.
- Çınlatma etkisi (Bölüm 4.3) işlemlerinde, çınlama süresi (Bölüm 2.3.3) attırıldığında, frekans bağımlı geciktirmeler nedeniyle ötme etkisi oluşmaktadır. Ötme etkisi, çınlatıcı yapı çıkışında metalik bir ses işaretinin elde edilmesine yol açmaktadır. Benzer durum, yüksek bir dinamik değere veya ani geçişlere sahip sayısal işaretler üzerinde gerçekleştirilen çınlatma etkisi uygulamalarında da ortaya çıkabilmektedir.
- Çok sayıda örnek değerinden oluşmuş sayısal işaretler için, KSFD (Bölüm 3.3) aracılığıyla gerçekleştirilen spektral gösterim (Bölüm 4.5) işlemi oldukça uzun bir sürede tamamlanabilmektedir. Bu durum, gösterimde kullanılan sistem bağımlı standart çizim fonksiyonlarının yavaşlığından kaynaklanmaktadır. KSFD işleminde kullanılan yöntem ve algoritmanın, ortaya çıkan yavaşlık sorunu ile herhangi bir ilgisi bulunmamaktadır.

Sonuç olarak, gerçek-zaman uygulamaları dışında, yüksek maliyetli özel bir elektronik cihaza gereksinim duymadan, farklı türde sayısal ses işlemleri tasarlanan yazılım aracılığıyla gerçekleştirilebilmektedir. GSİ yazılımında kullanılan temel yapı ve algoritmalara ait program dökümleri Ek 3'de verilmiştir. Sistem erişimi ve donanım özellikleri üzerinde yapılacak bir çalışma ile, yazılımın gerçek-zaman uygulamalarını da desteklemesi sağlanabilir. Ayrıca, çizim fonksiyonlarını içeren özel bir kütüphanenin tasarlanması yoluyla, grafik arabirim görselliğinin güçlendirilmesi ve hızlandırılması da olasıdır.

KAYNAKLAR

Allen, J. B. ve Rabiner, L. R., (1977), "A Unified Approach to Short-Time Fourier Analysis and Synthesis", Proceedings of the IEEE, 65: 1558-1564.

Chamberlin, H., (1985), Musical Applications of Microprocessors, Hayden Books, New Jersey.

Cooley, J. ve Tukey, J., (1965), "An Algorithm for the Machine Computation of Complex Fourier Series", Mathematical Computation, 19: 297-301.

Dattorro, J., (1997), "Effect Design, Part 1: Reverberator and Other Filters", Journal of the Audio Engineering Society, 45(9): 660-684.

Deer, J., Bloom, P. ve Preis, D., (1985), "Perception of Phase Distortion in Allpass Filters", Journal of the Audio Engineering Society, 33(10): 782-786.

Flanagan, J. L., (1972), Speech Analysis, Synthesis, and Perception, Springer-Verlag, New York.

Harris, F., (1978), "On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform", Proceedings of the IEEE, 66(1): 51-83.

Jackson, L. B., (1996), Digital Filters and Signal Processing, Norwell, MA.

Jaffe, D., (1987), "Spectrum Analysis Tutorial, Part 2: Properties and Applications of the Discrete Fourier Transform", Computer Music Journal, 11(3): 17-35.

Kayran, A. H., Panayırcı, E. ve Aygölü, Ü., (1992), Sayısal Haberleşme, Sistem Yayıncılık ve Matbaacılık, İstanbul.

Lane, J. E., (1990), "Pitch Detection Using a Tunable IIR Filter", Computer Music Journal, 14(3): 46-59.

Lane, J. E., Hoory, D., Martinez, E. ve Wang, P., (1997), "Modeling Analog Synthesis with DSPs", Computer Music Journal, 21(4): 23-41.

McNally, G., (1984), "Dynamic Range Control of Digital Audio Signals", Journal of the Audio Engineering Society, 32(5): 316-327.

Moore, F. R., (1990), Elements of Computer Music, Englewood Cliffs, New Jersey.

Moorer, J. A., (1977), "Signal Processing Aspects of Computer Music", Proceedings of the IEEE, 65(8): 1108-1137.

Moorer, J. A., (1979), "About this Reverberation Business", Computer Music Journal, 3(2): 13-28.

Nuttal, A., (1981), "Some Windows with Very Good Sidelobe Behavior", IEEE Transactions on Acoustics, Speech, and Signal Processing, ASSP-29(1): 84-91.

Preis, D., (1982), "Phase Distortion and Phase Equalization in Audio Signal Processing-A Tutorial Review", Journal of the Audio Engineering Society, 30(11): 774-794.

Rabiner, L. R., Cooley, J., Helms, H., Jackson, L., Kaiser, J., Rader, C., Schafer, R., Steiglitz, K. ve Weinstein, C., (1972), "Terminology in Digital Signal Processing", IEEE Transactions on Audio and Acoustics, AU-20: 322-337.

Regalia, P. A. ve Mitra, S. K., (1987), "Tunable Digital Frequency Response Equalization Filters", IEEE Transactions on Acoustics, Speech, and Signal Processing, ASSP-35: 118-120.

Roads, C., (1996), Computer Music Tutorial, MIT Press, MA.

Schafer, R. ve Rabiner, L. R., (1973), "Design and Simulation of a Speech Analysis-Synthesis System Based on Short-Time Fourier Analysis", IEEE Transactions on Audio and Electroacoustics, AU-21: 165-174.

Schroeder, M., (1961), "Improved Quasi-Stereophony and Colorless Artificial Reverberation", Journal of the Acoustical Society of America, 33: 1061.

Schroeder, M., (1962), "Natural Sounding Artificial Reverberation", Journal of the Audio Engineering Society, 10(3): 219-223.

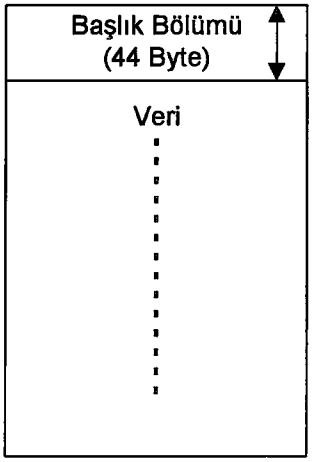
Schroeder, M. ve Atal, B. S., (1962), "Generalized Short-Time Power Spectra and Autocorrelation Functions", Journal of the Acoustical Society of America, 34: 1679-1683.

Schroeder, M., (1970), "Digital Simulation of Sound Transmission in Reverberant Spaces", Journal of the Acoustical Society of America, 47(2): 424-431.

EKLER

Ek 1 PCM Ses Dalgası Dosya Biçimi

Windows PCM ses dalgası (Wave File) dosyaları, bir başlık (header) ve veri bölümlerinden oluşmaktadır (Şekil E.1).



Şekil E.1 Ses dalgası dosya yapısı.

Dosyanın ilk 44 Byte'ında yer alan başlık bölümü, ses verisine ait kanal sayısı, örnekleme hızı, veri uzunluğu vb. bilgileri içermektedir. Başlık bölümü bileşenleri ve uzunlukları Çizelge E.1'de verilmiştir.

Çizelge E.1 PCM ses dalgası dosyası başlık bölümü.

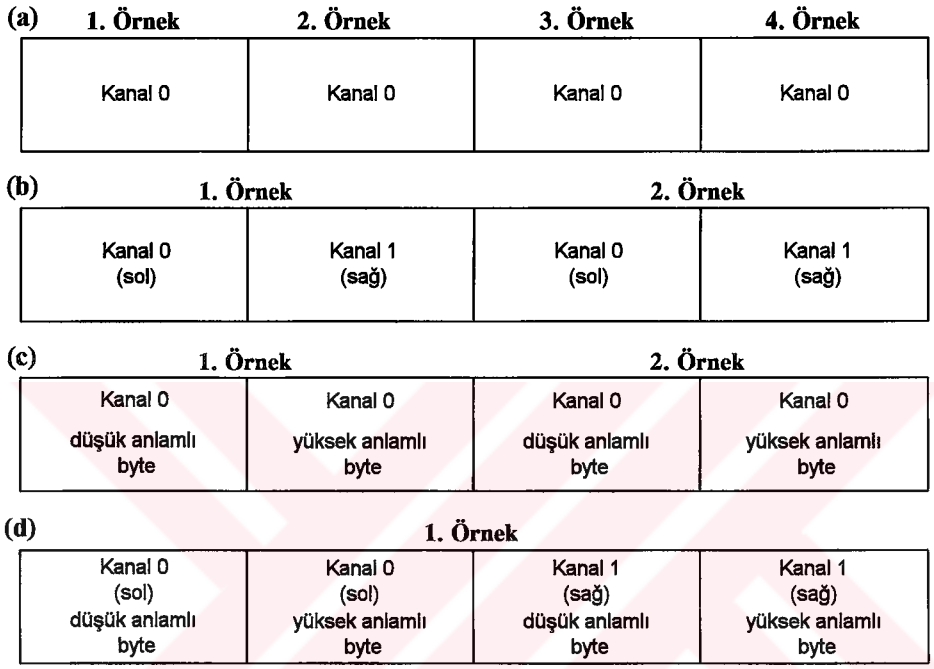
Başlık Bölümü Bileşeni	Boyut $[Byte]$
"RIFF"	4
$riff\ uzunluğu = 36 + (kanal\ sayısı \times bps \times veri\ uzunluğu/8)$	4
"WAVE"	4
"fmt "	4
$f = 16$	4
$format\ belirteci = 1$	2
kanal sayısı	2
örnekleme hızı	4
$saniyedeki\ ort.\ byte\ sayısı = (kanal\ sayısı \times örnek.\ hızı \times bps)/8$	4
$blok\ hizalama = (kanal\ sayısı \times bps)/8$	2
örnek başına düşen bit sayısı (bps)	2
"data"	4
veri uzunluğu	4

Örnek başına düşen bit sayısına göre tanımlanan dosya biçimleri ve değer aralıkları Çizelge E.2'de verilmiştir.

Çizelge E.2 Ses dalgası dosya biçimleri ve örnek değer aralıkları.

Biçim	En Büyük Değer	En Küçük Değer	Orta Değer
8-bit PCM	255 (0xFF)	0	128 (0x80)
16-bit PCM	32767 (0x7FFF)	-32768 (-0x8000)	0

Sayısal ses verisini oluşturan örnek değerlerinin, veri bölgesi içinde, kanal sayısı ve örnek başına düşen bit sayısına bağlı olarak belirlenen dizilişleri Şekil E.2'de gösterilmiştir.



Şekil E.2 Ses dalgası dosyası örnek değerlerinin veri bölgesi içindeki dizilişleri: (a) 8-bit tekli; (b) 8-bit çiftli; (c) 16-bit tekli; (d) 16-bit çiftli.

Ek 2 Kullanılan Türkçe Terimlerin İngilizce Karşılıkları

Türkçe	İngilizce
Ayrık Fourier Dönüşümü (AFD):	Discrete Fourier Transform (DFT).
Biçim:	Format.
Bit Çevrimi:	Bit Reversal.
Çentik Süzgeç:	Notch Filter.
Çınlatıcı:	Reverberator.
Çınlatma:	Reverberation.
Çiftleme:	Doubling.
Çiftli:	Stereo.
Dalgalandırıcı:	Flanger.
Dinamik Değer İşleme:	Dynamic Range Processing.
Doğrusal İlişkilendirme:	Linear Interpolation.
Dürtü:	Impulse.
Düşük Frekans Osilatörü (DFO):	Low Frequency Oscillator (LFO).
Etki	Effect.
Faz Giderme:	Phase Cancellation.
Faz Güçlendirme:	Phase Reinforcement.
Geciktirici:	Delay Line.
Genişletici:	Expander.
Gürültü Bastırıcı:	Noise Gate.
Hızlı Fourier Dönüşümü (HFD):	Fast Fourier Transform (FFT).
Kareköksel Ortalama Değer (KOD):	Root Mean Square (RMS).
Kesici Süzgeç:	Cut Filter.
Kırpma:	Distortion.
Kısa Süreli Fourier Dönüşümü (KSFD):	Short-Time Fourier Transform (STFT).
Koro:	Chorus.
Kulakçık:	Lobe.
Negatif Sönüm:	Fade Out.
Ötme:	Ringing.
Pozitif Sönüm:	Fade In.
Ses Dalgası Dosyası:	Wave File.
Sıkıştırıcı:	Compressor.
Sınırlandırıcı:	Limiter.
Sonlu Dürtü Yanıtlı (SDY):	Finite Impulse Response (FIR).
Sonsuz Dürtü Yanıtlı:	Infinite Impulse Response (IIR).
Tarak Süzgeç:	Comb Filter.
Tekli:	Mono.
Titreşim:	Vibrato.
Tüm Geçiren Süzgeç:	Allpass Filter.
Yinelemeli:	Recursive.

Ek 3 GSİ Yazılımında Kullanılan Temel Yapılar ve Algoritmalar

```
/*************************************************************************
ÇOK ÇIKIŞLI SAYISAL GECİKTİRİCİ (ÇÇSG)
*************************************************************************/

//---------------------------------------------------------------
#ifndef _DELAY_H
#define _DELAY_H

#include <stdio.h>

class delayline {
        protected:
                int * buffer;
                long maxlength;
                int ** read;
                int * write;
                long *delay_time_values;
                int output_number;
        public:
                delayline(long bufferlength, int read_pointer_number,
                        long * index = NULL);
                delayline();
                ~delayline();
                void set_delayline(long bufferlength, int read_pointer_number,
                        long *index = NULL);
                void delay_input(int data);
                int delay_output(int output_number);
                int delay_write_read(int delay_in);
                void set_delay_out(int output_number, long new_offset);
                void clear_delayline();
};
#endif
//---------------------------------------------------------------

#include <iostream.h>
#include <stdlib.h>
#include "delay.h"

delayline :: delayline(long bufferlength, int read_pointer_number, long *index)
{
        output_number = read_pointer_number;
        maxlength = bufferlength;
        buffer = (int *)malloc(sizeof(int) * maxlength);
        if(!buffer) {
                cout << "Yetersiz Bellek!!!" << endl;
```

```
                exit(1);
        }
        long i;
        for(i = 0; i < maxlength; ++i)
                buffer[i] = 0;
        write = buffer;
        read = (int **)malloc(sizeof(int *) * read_pointer_number);
        if(!read) {
                cout << "Yetersiz Bellek!!!" << endl;
                exit(1);
        }
        delay_time_values = (long *)malloc(sizeof(long) * read_pointer_number);
        if(!delay_time_values) {
                cout << "Yetersiz Bellek!!!" << endl;
                exit(1);
        }
        if(index == NULL) {
                for(i = 0; i < read_pointer_number; ++i) {
                        read[i] = buffer + maxlength - (i + 1);
                        delay_time_values[i] = i + 1;
                }
        }
        else {
                for(i = 0; i < read_pointer_number; ++i) {
                        if(index[i] >= maxlength) {
                                cout << "Geciktirme Sınırı Aşıldı!!!" << endl;
                                exit(1);
                        }
                delay_time_values[i] = index[i];
                read[i] = buffer + (maxlength - delay_time_values[i]) % maxlength;
                }
        }
}

delayline :: delayline()
{
        maxlength = 2048L;
        buffer = (int *)malloc(sizeof(int) * maxlength);
        if(!buffer) {
                cout << "Yetersiz Bellek!!!" << endl;
                exit(1);
        }
        long i;
        for(i = 0; i < maxlength; ++i)
                buffer[i] = 0;
        write = buffer;
        read = (int **)malloc(sizeof(int *) * 3);
```

```
        if(!read) {
                cout << "Yetersiz Bellek!!!" << endl;
                exit(1);
        }
        delay_time_values = (long *)malloc(sizeof(long) * 3);
        if(!delay_time_values) {
                cout << "Yetersiz Bellek!!!" << endl;
                exit(1);
        }
        for(i = 0; i < 3; ++i) {
                read[i] = buffer + maxlength - (i + 1);
                delay_time_values[i] = i + 1;
        }
}

void delayline :: set_delayline(long bufferlength, int read_pointer_number, long *index)
{
        output_number = read_pointer_number;
        maxlength = bufferlength;
        buffer = (int *)malloc(sizeof(int) * maxlength);
        if(!buffer) {
                cout << "Yetersiz Bellek!!!" << endl;
                exit(1);
        }
        long i;
        for(i = 0; i < maxlength; ++i)
                buffer[i] = 0;
        write = buffer;
        read = (int **)malloc(sizeof(int *) * read_pointer_number);
        if(!read) {
                cout << "Yetersiz Bellek!!!" << endl;
                exit(1);
        }
        delay_time_values = (long *)malloc(sizeof(long) * read_pointer_number);
        if(!delay_time_values) {
                cout << "Yetersiz Bellek!!!" << endl;
                exit(1);
        }
        if(index == NULL) {
                for(i = 0; i < read_pointer_number; ++i) {
                        read[i] = buffer + maxlength - (i + 1);
                        delay_time_values[i] = i + 1;
                }
        }
        else {
                for(i = 0; i < read_pointer_number; ++i) {
                        if(index[i] >= maxlength) {
```

```
                        cout << "Geciktirme Sınırı Aşıldı!!!" << endl;
                        exit(1);
                    }
                    delay_time_values[i] = index[i];
                    read[i] = buffer + (maxlength - delay_time_values[i]) % maxlength;
                }
        }
}

delayline :: ~delayline()
{
        free(buffer);
        free(read);
        free(delay_time_values);
}

void delayline :: delay_input(int data)
{
        *write = data;
        if(write == (buffer + maxlength - 1))
                write = buffer;
        else
                ++write;
}

int delayline :: delay_output(int output_number)
{
        if(read[output_number] == (buffer + maxlength - 1)) {
                read[output_number] = buffer;
                return *(buffer + maxlength - 1);
        }
        else {
                ++read[output_number];
                return *(read[output_number] - 1);
        }
}

int delayline :: delay_write_read(int delay_in)
{
        delay_input(delay_in);
        return delay_output(0);
}

void delayline :: set_delay_out(int output_number, long new_offset)
{
        if(new_offset >= maxlength) {
                cout << "Yeni Geciktirme Değeri Sınırı Aşıyor!!!" << endl;
```

```
                exit(1);
        }
        if(new_offset <= (write - buffer))
                read[output_number] = write - new_offset;
        else
                read[output_number] = buffer + maxlength - (new_offset - (write - buffer));
}

void delayline :: clear_delayline()
{
        long i;
        for(i = 0; i < maxlength; ++i)
                buffer[i] = 0;
        write = buffer;
        for(i = 0; i < output_number; ++i)
                read[i] = buffer + (maxlength - delay_time_values[i]) % maxlength;
}

/*************************************************************************
YUVARLAMA FONKSİYONU
*************************************************************************/

#include <math.h>

double round(double d)
{
        if(d - floor(d) < 0.5)
                return floor(d);
        else
                return ceil(d);
}

/*************************************************************************
DÜŞÜK FREKANS OSİLATÖRÜ (DFO)
*************************************************************************/

//---------------------------------------------------------------
#ifndef lfoH
#define lfoH


class LFO {
        protected:
                double amp;
                double fr;
                double phase;
                double FSK;
```

```
        public:
                LFO(double _amp, double _fr, double _phase,
                        double _FSK = 44100);
                LFO();
                void set_lfo(double _amp, double _fr, double _phase,
                        double _FSK = 44100);
                double LFO_out(unsigned long i);
};

#endif
//---------------------------------------------------------------------

#include <math.h>
#include <iostream.h>
#include "lfo.h"

LFO :: LFO(double _amp, double _fr, double _phase, double _FSK)
{
        amp = _amp;
        fr = _fr;
        phase = _phase;
        FSK = _FSK;
}

LFO :: LFO()
{
        amp = 48.0;
        fr = 6.5;
        phase = 0.0;
        FSK = 44100;
}

double LFO :: LFO_out(unsigned long i)
{
        return amp * sin(2.0 * M_PI * fr * i / FSK + phase);
}

void LFO :: set_lfo(double _amp, double _fr, double _phase,  double _FSK)
{
        amp = _amp;
        fr = _fr;
        phase = _phase;
        FSK = _FSK;
}
```

```
/*****************************************************************************
TEK KUTUPLU SÜZGEÇ
********************************************** **********************/

//---------------------------------------------------------------
#ifndef onepoleH
#define onepoleH

class OnePole {
        protected:
                enum {STATE};
                float band_width;
                int last_out;
   public:
                OnePole(float bwidth);
                OnePole();
                void set_onepole(float bwidth);
                int OnePole_Out(int data);
                int lpf_out(int data);
};
#endif
//---------------------------------------------------------------

#include "onepole.h"

OnePole :: OnePole(float bwidth)
{
        band_width = bwidth;
        last_out = STATE;
}

OnePole :: OnePole()
{
        band_width = 1.0f;
        last_out = STATE;
}

void OnePole :: set_onepole(float bwidth)
{
        band_width = bwidth;
        last_out = STATE;
}

int OnePole :: OnePole_Out(int data)
{
        int pole_out = last_out;
        last_out = (int)((band_width * data) + ((1 - band_width) * last_out));
```

```
        return pole_out;
}

int OnePole :: lpf_out(int data)
{
        int filter_out = last_out;
        last_out = (int) (data + band_width * last_out);
        return filter_out;
}

/*************************************************************************
GENEL AMAÇLI GECİKTİRME ETKİSİ ÜRETECİ (GEÜ)
************************************************************ ************/

//---------------------------------------------------------------
#ifndef chorusH
#define chorusH

#define M_VIBRATO 1
#define N_VIBRATO 2
#define E_VIBRATO 3
#define M_FLANGE 4
#define N_FLANGE 5
#define E_FLANGE 6
#define M_CHORUS 7
#define N_CHORUS 8
#define E_CHORUS 9
#define M_WCHORUS 10
#define N_WCHORUS 11
#define E_WCHORUS 12
#define M_DOUBLING 13
#define N_DOUBLING 14
#define E_DOUBLING 15
#define M_ECHO 16
#define N_ECHO 17
#define E_ECHO 18

class chorus : public delayline, public LFO, public OnePole {
        protected:
                float feedback;
                float feedforward;
                float blend;
                long sampling_rate;

        public:
                chorus( float fforward, float fback, float blnd, long samp_rate = 44100);
                chorus();
```

```
                void set_chorus_parameters(float _feedback, float _feedforward,
                        float _blend, long samp_rate = 44100);
                double linvibrato(int vibrato_in, unsigned long lfo_samp_var);
                void set_user_defined_effect(float fback, float fwrd,float blnd,
                        long samp_rate, float brightness, float delay_range,
                        float amp_coeff, double _fr, double _phase, double _FSK);
                int generic_effect_func(int flange_input, long flange_index);
                void effect_data_base(int index_number, long srate);
};
#endif
//---------------------------------------------------------------

#include "delay.h"
#include "lfo.h"
#include "onepole.h"
#include "chorus.h"
#include <math.h>
#include <iostream.h>
#include <stdlib.h>

double round (double);

chorus :: chorus(float fforward, float fback, float blnd, long samp_rate)
{
        feedback = fback;
        feedforward = fforward;
        blend = blnd;
        sampling_rate = samp_rate;
}

chorus :: chorus()
{
        feedback = 0.0f;
        feedforward = 0.0f;
        blend = 0.0f;
        sampling_rate = 44100;
}

void chorus :: set_chorus_parameters(float _feedback, float _feedforward, float _blend,
long samp_rate)
{
        feedback = _feedback;
        feedforward = _feedforward;
        blend = _blend;
        sampling_rate = samp_rate;
}
```

```
double chorus :: linvibrato(int vibrato_in, unsigned long lfo_samp_var)
{
        delay_input(vibrato_in);
        double x = LFO_out(lfo_samp_var);
        double y = floor(x);
        double z = x - y;
        double temp = round((delay_output(0) * (1.0 - z)) + (delay_output(1) * z));
        temp = OnePole_Out(temp);
        set_delay_out(0, (long) (delay_time_values[0] + y));
        set_delay_out(1, (long) (delay_time_values[1] + y));
        return temp;
}

int chorus :: generic_effect_func(int flange_input, long flange_index)
{
        float temp = flange_input - delay_output(2) * feedback;
        return (int) (temp * blend + linvibrato((int) temp, flange_index) * feedforward);
}

void chorus :: set_user_defined_effect(float fback, float fwrd, float blnd, long samp_rate,
float brightness, float delay_range, float amp_coeff, double _fr, double _phase,
double _FSK)
{
        set_chorus_parameters(fback, fwrd, blnd, samp_rate);
        set_onepole(brightness);
        double temp = samp_rate / 1000.0 * delay_range;
        maxlength = 2L + (long)(temp * (1.0 + amp_coeff));
        double last_amp = (double)(temp * amp_coeff);
        long middle_point = (long) temp;
        long delay_array[] = {middle_point, middle_point + 1, middle_point};
        set_delayline(maxlength, 3, delay_array);
        set_lfo(last_amp, _fr, _phase, _FSK);
}

void chorus :: effect_data_base(int index_number, long srate)
{
        float sratio = srate / 44100.0;
        float temp;
        if(index_number == 1) {
                temp = 44L + sratio;
                long mild_vib[] = {temp, temp + 1L, temp};
                set_delayline(3 * temp + 3L, 3, mild_vib);
                set_lfo(26.46001 * sratio, 2.5, 1.49547, srate);
                set_onepole(0.5);
                set_chorus_parameters(0.0, 1.0, 0.005, srate);
        }
        if(index_number == 2) {
```

```
        temp = 57L + sratio;
        long nom_vib[] = {temp, temp + 1L, temp};
        set_delayline(3 * temp + 3L, 3, nom_vib);
        set_lfo(40.130998 * sratio, 3.5, 1.49547, srate);
        set_onepole(0.5);
        set_chorus_parameters(0.0, 1.0, 0.005, srate);
}
if(index_number == 3) {
        temp = 180L + sratio;
        long exag_vib[] = {temp, temp + 1L, temp};
        set_delayline(3 * temp + 3L, 3, exag_vib);
        set_lfo(72.323999 * sratio, 7.5, 1.49547, srate);
        set_onepole(0.5);
        set_chorus_parameters(0.0, 1.0, 0.005, srate);
}
if(index_number == 4) {
        temp = 66L + sratio;
        long mild_flange[] = {temp, temp + 1L, temp};
        set_delayline(3 * temp + 3L, 3, mild_flange);
        set_lfo(56.630182 * sratio, 0.2582, 1.49547, srate);
        set_onepole(0.4);
        set_chorus_parameters(-0.7071, 0.7071, 0.7071, srate);
}
if(index_number == 5) {
        temp = 89L + sratio;
        long nom_flange[] = {temp, temp + 1L, temp};
        set_delayline(3 * temp + 3L, 3, nom_flange);
        set_lfo(88.1922 * sratio, 0.55, 1.49547, srate);
        set_onepole(0.4);
        set_chorus_parameters(-0.7071, 0.7071, 0.7071, srate);
}
if(index_number == 6) {
        temp = 397L + sratio;
        long exag_flange[] = {temp, temp + 1L, temp};
        set_delayline(3 * temp + 3L, 3, exag_flange);
        set_lfo(396.8571 * sratio, 1.1, 1.49547, srate);
        set_onepole(0.4);
        set_chorus_parameters(-0.7071, 0.7071, 0.7071, srate);
}
if(index_number == 7) {
        temp = 441L + sratio;
        long mild_chorus[] = {temp, temp + 1L, temp};
        set_delayline(3 * temp + 3L, 3, mild_chorus);
        set_lfo(30.87 * sratio, 2.0, 1.49547, srate);
        set_onepole(0.5);
        set_chorus_parameters(0.0005, 0.7071, 1.0, srate);
}
```

```
if(index_number == 8) {
        temp = 661L + sratio;
        long nom_chorus[] = {temp, temp + 1L, temp};
        set_delayline(3 * temp + 3L, 3, nom_chorus);
        set_lfo(46.305 * sratio, 3.2, 1.49547, srate);
        set_onepole(0.5);
        set_chorus_parameters(0.0005, 0.7071, 1.0, srate);
}

if(index_number == 9) {
        temp = 1102L + sratio;
        long exag_chorus[] = {temp, temp + 1L, temp};
        set_delayline(3 * temp + 3L, 3, exag_chorus);
        set_lfo(11.025 * sratio, 8.0, 1.49547, srate);
        set_onepole(0.5);
        set_chorus_parameters(0.0005, 0.7071, 1.0 , srate);
}

if(index_number == 10) {
        temp = 441L + sratio;
        long mild_white_chorus[] = {temp, temp + 1L, temp};
        set_delayline(3 * temp + 3L, 3, mild_white_chorus);
        set_lfo(30.87 * sratio, 2.0, 1.49547, srate);
        set_onepole(0.4);
        set_chorus_parameters(0.7071, 1.0, 0.7071 , srate);
}

if(index_number == 11) {
        temp = 661L + sratio;
        long nom_white_chorus[] = {temp, temp + 1L, temp};
        set_delayline(3 * temp + 3L, 3, nom_white_chorus);
        set_lfo(46.305 * sratio, 3.2, 1.49547, srate);
        set_onepole(0.4);
        set_chorus_parameters(0.7071, 1.0, 0.7071 , srate);
 }

if(index_number == 12) {
        temp = 1102L + sratio;
        long exag_white_chorus[] = {temp, temp + 1L, temp};
        set_delayline(3 * temp + 3L, 3, exag_white_chorus);
        set_lfo(11.025 * sratio, 8.0, 1.49547, srate);
        set_onepole(0.4);
        set_chorus_parameters(0.7071, 1.0, 0.7071 , srate);
}

if(index_number == 13) {
        temp = 661L + sratio;
```

```
            long mild_doubling[] = {temp, temp + 1L, temp};
            set_delayline(3 * temp + 3L, 3, mild_doubling);
            set_lfo(297.6749 * sratio, 0.15, 1.49547, srate);
            set_onepole(0.5);
            set_chorus_parameters(0.0005, 0.7071, 0.7071, srate);
        }
        if(index_number == 14) {
            temp = 882L + sratio;
            long nom_doubling[] = {temp, temp + 1L, temp};
            set_delayline(3 * temp + 3L, 3, nom_doubling);
            set_lfo(441 * sratio, 0.15, 1.49547, srate);
            set_onepole(0.5);
            set_chorus_parameters(0.0005, 0.7071, 0.7071, srate);
        }
        if(index_number == 15) {
            temp = 1764L + sratio;
            long exag_doubling[] = {temp, temp + 1L, temp};
            set_delayline(3 * temp + 3L, 3, exag_doubling);
            set_lfo(1234.7999 * sratio, 0.15, 1.49547, srate);
            set_onepole(0.5);
            set_chorus_parameters(0.0005, 0.7071, 0.7071, srate);
        }
        if(index_number == 16) {
            temp = 3528L + sratio;
            long mild_echo[] = {temp, temp + 1L, temp};
            set_delayline(3 * temp + 3L, 3, mild_echo);
            set_lfo(176.4 * sratio, 0.15, 1.49547, srate);
            set_onepole(0.5);
            set_chorus_parameters(0.7071, 0.80, 1.0, srate);
        }
        if(index_number == 17) {
            temp = 5292L + sratio;
            long nom_echo[] = {temp, temp + 1L, temp};
            set_delayline(3 * temp + 3L, 3, nom_echo);
            set_lfo(264.6 * sratio, 0.15, 1.49547, srate);
            set_onepole(0.5);
            set_chorus_parameters(0.7071, 0.80, 1.0, srate);
        }
        if(index_number == 18) {
            temp = 22050L + sratio;
            long exag_echo[] = {temp, temp + 1L, temp};
            set_delayline(3 * temp + 3L, 3, exag_echo);
            set_lfo(1102.5 * sratio, 0.15, 1.49547, srate);
            set_onepole(0.5);
            set_chorus_parameters(0.7071, 0.80, 1.0, srate);
        }
}
```

```
/*************************************************************************
LATTICE TÜM GEÇİREN SÜZGEÇ
*************************************************************************/

//---------------------------------------------------------------

#ifndef latticeH
#define latticeH

class lattice : public delayline, public LFO {

        protected:
                float lattice_coeff;
                int lattice_buff;
                int delay_input_buff;

        public:
                lattice(float _coeff);
                lattice();
                void set_lattice_coeff(float _coeff);
                void set_lattice(float _coeff, int delay_out_number, long *delay_start_offset,
                        double lfo_amp = 0.0, double lfo_fr = 0.0, double lfo_phase = 0.0,
                        double lfo_FSK = 0.0);
                int lattice_simple_out(int lattice_input);
                int lattice_var_out(int lattice_input, long lfo_samp_var);
};

#endif
//---------------------------------------------------------------

#include "delay.h"
#include "lfo.h"
#include <math.h>
#include <iostream.h>
#include <stdlib.h>
#include "lattice.h"

double round(double);

lattice :: lattice(float _coeff)
{
        lattice_coeff = _coeff;
        delay_input_buff = 0;
}
```

```
lattice :: lattice()
{
        lattice_coeff = 1.0f;
        delay_input_buff = 0;
}

void lattice :: set_lattice_coeff(float _coeff)
{
        lattice_coeff = _coeff;
}

void lattice :: set_lattice(float _coeff, int delay_out_number, long * delay_start_offset,
double lfo_amp, double lfo_fr, double lfo_phase, double lfo_FSK)
{
        lattice_coeff = _coeff;
        if(lfo_amp == 0.0f)
        set_delayline(delay_start_offset[0] + 2L, delay_out_number, delay_start_offset);
        else {
                set_delayline(delay_start_offset[0] + 2L + (long) lfo_amp,
                delay_out_number, delay_start_offset);
                set_lfo(lfo_amp, lfo_fr, lfo_phase, lfo_FSK);
        }
}

int lattice :: lattice_simple_out(int lattice_input)
{
        delay_input_buff = delay_output(0);
        lattice_buff = lattice_input * lattice_coeff +
                delay_input_buff * (1 - lattice_coeff * lattice_coeff);
        delay_input(lattice_input + delay_input_buff * -1.0 * lattice_coeff);
        return lattice_buff;
}

int lattice :: lattice_var_out(int lattice_input, long lfo_samp_var)
{
        double x = LFO_out(lfo_samp_var);
        double y = floor(x);
        double z = x - y;
        delay_input(delay_input_buff);
        lattice_buff = (int) (round((delay_output(0) * (1.0 - z)) + (delay_output(1) * z)));
        set_delay_out(0, (long) (delay_time_values[0] + y));
        set_delay_out(1, (long) (delay_time_values[1] + y));
        int last_output = lattice_coeff * delay_input_buff + lattice_buff;
        delay_input_buff = lattice_input - lattice_coeff * lattice_buff;
        return last_output;
}
```

```
/*************************************************************************
TARAK SÜZGEÇ
*********************************************** ***************************/

//---------------------------------------------------------------
#ifndef combH
#define combH

class comb : public delayline, public LFO, public OnePole {
        protected:
                int  comb_buff;
                float comb_coeff;
        public:
                comb(float _comb_coeff, long delay_time);
                comb();
                void set_comb(float _comb_coeff, long delay_time,
                        float onepole_coeff = 1.0f, double lfo_amp = 0.0,
                        double lfo_fr = 0.0, double lfo_phase = 0.0, double lfo_FSK = 0.0);
                int comb_out(int comb_input);
                int osc_comb_out(int comb_input);
};
#endif
//---------------------------------------------------------------

#include "delay.h"
#include "lfo.h"
#include "onepole.h"
#include <math.h>
#include <iostream.h>
#include <stdlib.h>
#include "comb.h"

double round(double);

comb :: comb(float _comb_coeff, long delay_time)
{
        comb_coeff = _comb_coeff;
        long delay_array[1];
        delay_array[0] = delay_time;
        set_delayline(delay_time + 2L, 1, delay_array);
}

comb :: comb()
{
        comb_coeff = 1.0;
        comb_buff =0;
}
```

```
void comb :: set_comb(float _comb_coeff, long delay_time,float onepole_coeff,
double lfo_amp, double lfo_fr, double lfo_phase, double lfo_FSK)
{
        comb_coeff = _comb_coeff;
        if(onepole_coeff != 1.0f)
                set_onepole(onepole_coeff);
        if(lfo_amp == 0.0f) {
                long delay_array[1] = {delay_time};
                set_delayline(delay_time + 2L, 1, delay_array);
        }
        else {
                long delay_array[2] = {delay_time, delay_time + 1};
                set_delayline(delay_time + 2L + (long)lfo_amp, 2, delay_array);
                set_lfo(lfo_amp, lfo_fr, lfo_phase, lfo_FSK);
        }
}

int comb :: comb_out(int comb_input)
{
        comb_buff = delay_output(0);
        delay_input(comb_buff * comb_coeff + comb_input);
        return comb_buff;
}

int comb :: osc_comb_out(int comb_input)
{
        comb_buff = delay_output(0);
        delay_input(lpf_out(comb_buff) * comb_coeff + comb_input);
        return comb_buff;
}

/*************************************************************************
TÜMLEŞİK ÇINLATICI YAPI (TÇY)
*************************************** ******************************/

//---------------------------------------------------------------
#ifndef reverbH
#define reverbH

#include <iostream.h>
#include <stdlib.h>
#include <math.h>
#include "delay.h"
#include "onepole.h"
#include "lfo.h"
#include "comb.h"
#include "lattice.h"
```

```
class reverb {
        protected:
                comb comb1;
                comb comb2;
                comb comb3;
                comb comb4;
                comb comb5;
                comb comb6;
                lattice allpass1;
                delayline reflection;
                delayline reverb_delay;
                OnePole pole;
                float reverb_gain;
                float reverb_distance;
                float onepole_coeff;
                float dry_coeff;
                float wet_coeff;
                int delay_flag;
                long add_delay;
                long samprate;
        public:
                reverb(float rgain, float rdistance, float polecoeff, float dry,
                        float wet, long _samprate);
                void set_reverb(float rgain, float rdistance, float polecoeff,
                        float dry, float wet, long _samprate);
                int reverb_out(int input_data);
                long get_add_delay();
};
#endif
//---------------------------------------------------------------
#include "reverb.h"

reverb :: reverb(float rgain, float rdistance, float polecoeff, float dry,
float wet, long _samprate)
{
        delay_flag = 0;
        reverb_gain = rgain;
        reverb_distance = rdistance;
        onepole_coeff = polecoeff;
        dry_coeff = dry;
        wet_coeff = wet;
        samprate = _samprate;
        float samp_ratio = samprate / 44100.0;
        comb1.set_comb(reverb_gain * (1 - 0.405 * samp_ratio),
                2602 * reverb_distance * samp_ratio, 0.405 * samp_ratio);
        comb2.set_comb(reverb_gain * (1 - 0.423 * samp_ratio),
                2746 * reverb_distance * samp_ratio, 0.423 * samp_ratio);
```

```
        comb3.set_comb(reverb_gain * (1 - 0.441 * samp_ratio),
                2994 * reverb_distance * samp_ratio, 0.441 * samp_ratio);
        comb4.set_comb(reverb_gain * (1 - 0.458 * samp_ratio),
                3046 * reverb_distance * samp_ratio, 0.458 * samp_ratio);
        comb5.set_comb(reverb_gain * (1 - 0.529 * samp_ratio),
                3202 * reverb_distance * samp_ratio, 0.529 * samp_ratio);
        comb6.set_comb(reverb_gain * (1 - 0.564 * samp_ratio),
                3362 * reverb_distance * samp_ratio, 0.564 * samp_ratio);
        long lat_array[1];
        lat_array[0] = 37;
        allpass1.set_lattice(0.7, 1, lat_array);
        long delay_array[] = {0, 190 * samp_ratio, 948 * samp_ratio, 992 * samp_ratio,
                1182 * samp_ratio, 1191 * samp_ratio, 1314 * samp_ratio,
                2019 * samp_ratio, 2139 * samp_ratio, 2523 * samp_ratio,
                2589 * samp_ratio, 2623 * samp_ratio, 2699 * samp_ratio,
                3117 * samp_ratio, 3123 * samp_ratio, 3201 * samp_ratio,
                3267 * samp_ratio, 3321 * samp_ratio, 3515 * samp_ratio};
        reflection.set_delayline((3515L * samp_ratio) + 2L, 19, delay_array);
        long r_delay[1];
        add_delay = 3515 * samp_ratio - ((2602 * reverb_distance * samp_ratio) + 37);
        long delay_add = add_delay;
        if(delay_add >= 0) {
                r_delay[0] = delay_add;
                delay_flag = 1;
        }
        else {
                delay_add *= -1;
                r_delay[0] = delay_add;
        delay_flag = 0;
        }
        reverb_delay.set_delayline(delay_add + 2L, 1, r_delay);
        pole.set_onepole(onepole_coeff);
}

void reverb :: set_reverb(float rgain, float rdistance, float polecoeff, float dry, float wet,
long _samprate)
{
        delay_flag = 0;
        reverb_gain = rgain;
        reverb_distance = rdistance;
        onepole_coeff = polecoeff;
        dry_coeff = dry;
        wet_coeff = wet;
        samprate = _samprate;
        float samp_ratio = samprate / 44100.0;
        comb1.set_comb(reverb_gain * (1 - 0.405 * samp_ratio),
                2602 * reverb_distance * samp_ratio, 0.405 * samp_ratio);
```

```
        comb2.set_comb(reverb_gain * (1 - 0.423 * samp_ratio),
                2746 * reverb_distance * samp_ratio, 0.423 * samp_ratio);
        comb3.set_comb(reverb_gain * (1 - 0.441 * samp_ratio),
                2994 * reverb_distance * samp_ratio, 0.441 * samp_ratio);
        comb4.set_comb(reverb_gain * (1 - 0.458 * samp_ratio),
                3046 * reverb_distance * samp_ratio, 0.458 * samp_ratio);
        comb5.set_comb(reverb_gain * (1 - 0.529 * samp_ratio),
                3202 * reverb_distance * samp_ratio, 0.529 * samp_ratio);
        comb6.set_comb(reverb_gain * (1 - 0.564 * samp_ratio),
                3362 * reverb_distance * samp_ratio, 0.564 * samp_ratio);
        long lat_array[1];
        lat_array[0] = 37; //265
        allpass1.set_lattice(0.7, 1, lat_array);
        long delay_array[] = {0, 190 * samp_ratio, 948 * samp_ratio, 992 * samp_ratio,
                1182 * samp_ratio, 1191 * samp_ratio, 1314 * samp_ratio,
                2019 * samp_ratio, 2139 * samp_ratio, 2523 * samp_ratio,
                2589 * samp_ratio, 2623 * samp_ratio, 2699 * samp_ratio,
                3117 * samp_ratio, 3123 * samp_ratio, 3201 * samp_ratio,
                3267 * samp_ratio, 3321 * samp_ratio, 3515 * samp_ratio};
        reflection.set_delayline((3515L * samp_ratio) + 2L, 19, delay_array);
        long r_delay[1];
        add_delay = 3515 * samp_ratio - ((2602 * reverb_distance * samp_ratio) + 37);
        long delay_add = add_delay;
        if(delay_add >= 0) {
                r_delay[0] = delay_add;
                delay_flag = 1;
        }
        else {
                delay_add *= -1;
                r_delay[0] = delay_add;
                delay_flag = 0;
        }
        reverb_delay.set_delayline(delay_add + 2L, 1, r_delay);
        pole.set_onepole(onepole_coeff);
}

int reverb :: reverb_out(int input_data)
{
        int temp_reverb;
        int temp_delay;
        temp_reverb = (1.0 / 6) * (comb1.osc_comb_out(comb_input) +
                        comb2.osc_comb_out(comb_input) +
                        comb3.osc_comb_out(comb_input) +
                        comb4.osc_comb_out(comb_input) +
                        comb5.osc_comb_out(comb_input) +
                        comb6.osc_comb_out(comb_input));
        temp_reverb = pole.OnePole_Out(allpass1.lattice_simple_out(temp_reverb));
```

```
        reflection.delay_input(input_data);
        temp_delay = dry_coeff * (reflection.delay_output(0) +
                        0.841 * reflection.delay_output(1) +
                        0.504 * reflection.delay_output(2) +
                        0.491 * reflection.delay_output(3) +
                        0.379 * reflection.delay_output(4) +
                        0.380 * reflection.delay_output(5) +
                        0.346 * reflection.delay_output(6) +
                        0.289 * reflection.delay_output(7) +
                        0.272 * reflection.delay_output(8) +
                        0.192 * reflection.delay_output(9) +
                        0.217 * reflection.delay_output(10) +
                        0.181 * reflection.delay_output(11) +
                        0.180 * reflection.delay_output(13) +
                        0.181 * reflection.delay_output(14) +
                        0.176 * reflection.delay_output(15) +
                        0.142 * reflection.delay_output(16) +
                        0.167 * reflection.delay_output(17) +
                        0.134 * reflection.delay_output(18));
        if(delay_flag == 1) {
                reverb_delay.delay_input(temp_reverb);
                return temp_delay + reverb_delay.delay_output(0) * wet_coeff;
        }
        else {
                reverb_delay.delay_input(temp_delay);
                return reverb_delay.delay_output(0) + temp_reverb * wet_coeff;
        }
}

long reverb :: get_add_delay()
{
        return add_delay;
}

/*************************************************************************
SÖNÜMLENDİRİCİ
*************************************************************************/

//---------------------------------------------------------------
#ifndef fadeH
#define fadeH

class fade {
        protected:
                double linear_coeff1;
                double linear_coeff2;
                double alpha;
```

```
                double exp_coeff;
        public:
                fade(double first_amp, double last_amp, unsigned long _data_size);
                int fadelin_in_out(int input_data, unsigned long sample_number);
                int fadelog_in_out(int input_data, unsigned long sample_number);
};
#endif
//---------------------------------------------------------------

#include <math.h>
#include "fade.h"

fade :: fade(double first_amp, double last_amp, unsigned long  _data_size)
{
        unsigned long data_size = _data_size;
        linear_coeff2 = first_amp;
        linear_coeff1 = (last_amp - linear_coeff2) / data_size;
        if(first_amp == 0.0)
                first_amp += 0.000001;
        if(last_amp == 0.0)
                last_amp += 0.000001;
        exp_coeff = first_amp;
        alpha = log(last_amp / exp_coeff) / data_size;
}

int fade :: fadelin_in_out(int input_data, unsigned long sample_number)
{
        return (linear_coeff1 * sample_number + linear_coeff2) * input_data;
}

int fade :: fadelog_in_out(int input_data, unsigned long sample_number)
{
        return (exp_coeff * exp(alpha * sample_number)) * input_data;
}

/*****************************************************************************
GÜRÜLTÜ BASTIRICI
*****************************************************************************/

//---------------------------------------------------------------

#ifndef noiseH
#define noiseH

#define VMAX 32767.0
```

```
class noise {
        protected:
                float alpha1;
                float alpha2;
                float m1;
                float m2;
                int limit1;
                int limit2;
                int packet_size;
        public:
                noise(float ratio, int _limit1, int _limit2, int _psize);
                void noise_out(unsigned long data_address);
};
#endif
//---------------------------------------------------------------------

#include "noise.h"
#include <math.h>

noise :: noise(float ratio, int _limit1, int _limit2, int _psize)
{
        alpha1 = ratio;
        limit1 = _limit1;
        limit2 = _limit2;
        float l1 = 20.0 * log10(limit1 / VMAX);
        float l2 = 20.0 * log10(limit2 / VMAX);
        m1 = -100 * (1 - alpha1);
        float t1 = l1 * alpha1 + m1;
        float t2 = l2;
        alpha2 = (t1 - t2) / (l1 - l2);
        m2 = l2 * (1 - alpha2);
        packet_size = _psize;
}

void noise :: noise_out(unsigned long data_address)
{
        unsigned long i;
        double rms = 0.0;
        for(i = 0; i < channel_flag * packet_size; i += channel_flag)
                rms += pow((double)data_buffer[i + data_address], 2.0);
        rms /= packet_size;
        rms = sqrt(rms);
        for(i = 0; i < channel_flag * packet_size; i += channel_flag) {
                if((rms >= limit1 && rms < limit2) && data_buffer[i + data_address] > 0)
                        data_buffer[i + data_address] = (short int) (VMAX * pow(10.0,
                                (alpha2 * 20.0 * log10(data_buffer[i + data_address] /
                                VMAX) + m2) / 20));
```

```
            if(rms < limit1 && data_buffer[i + data_address] > 0)
                data_buffer[i + data_address] = (short int)(VMAX * pow(10.0,
                    (alpha1 * 20.0 * log10(data_buffer[i + data_address] /
                    VMAX) + m1) / 20));
            if((rms >= limit1 && rms < limit2) && data_buffer[i + data_address] < 0)
                data_buffer[i + data_address] = (short int)(-VMAX * pow(10.0,
                    (alpha2 * 20.0 * log10(-data_buffer[i + data_address] /
                    VMAX) + m2) / 20));
            if(rms < limit1 && data_buffer[i + data_address] < 0)
                data_buffer[i + data_address] = (short int)(-VMAX * pow(10.0,
                    (alpha1 * 20.0 * log10(-data_buffer[i + data_address] /
                    VMAX) + m1) / 20));
        }

        if(channel_flag == 2) {
            double rms = 0.0;
            for(i = 1; i < channel_flag * packet_size; i += channel_flag)
                rms += pow((double)data_buffer[i + data_address], 2.0);
            rms /= packet_size;
            rms = sqrt(rms);
            for(i = 1; i < channel_flag * packet_size; i += channel_flag) {
                if((rms >= limit1 && rms < limit2) &&
                data_buffer[i + data_address] > 0)
                    data_buffer[i + data_address] = (short int) (VMAX *
                    pow(10.0, (alpha2 * 20.0 *
                    log10(data_buffer[i + data_address] / VMAX) + m2) / 20));
                if(rms < limit1 && data_buffer[i + data_address] > 0)
                    data_buffer[i + data_address] = (short int)(VMAX *
                    pow(10.0, (alpha1 * 20.0 *
                    log10(data_buffer[i + data_address] / VMAX) + m1) / 20));
                if((rms >= limit1 && rms < limit2) &&
                data_buffer[i + data_address] < 0)
                    data_buffer[i + data_address] = (short int)(-VMAX *
                    pow(10.0, (alpha2 * 20.0 *
                    log10(-data_buffer[i + data_address] / VMAX) + m2) / 20));
                if(rms < limit1 && data_buffer[i + data_address] < 0)
                    data_buffer[i + data_address] = (short int)(-VMAX *
                    pow(10.0, (alpha1 * 20.0 *
                    log10(-data_buffer[i + data_address] / VMAX) + m1) / 20));
            }
        }
}
```

/ Not: "channel_flag" ve "data_buffer" değişkenleri, external (dışsal) değişkenlerdir. Bu değişkenler, editöre ait ana (main) koddan gürültü bastırıcı yapısına aktarılmaktadır.*/*

```
/*************************************************************************
SIKIŞTIRICI
*************************************************************************/

//---------------------------------------------------------------

#ifndef compressH
#define compressH

#define VMAX 32767.0

class compress {
        protected:
                int treshold;
                int limit;
                float k;
                float m;
        public:
                compress(int _treshold, int _limit);
                int compress_out(int input_data);
};

#endif

//---------------------------------------------------------------

#include "compress.h"
#include <math.h>

compress :: compress(int _treshold, int _limit)
{
        treshold = _treshold;
        limit = _limit;
        k = (treshold - limit) / log(treshold / VMAX);
        m = limit - k * log(VMAX);
}

int compress :: compress_out(int input_data)
{
        if(input_data >= treshold)
                return k * log(input_data) + m;
        if(input_data <= -treshold)
                return -(k * log(-input_data) + m);
        else
                return input_data;
}
```

```
/*************************************************************************
GENİŞLETİCİ
*************************************************************************/

//---------------------------------------------------------------
#ifndef expandH
#define expandH

#define VMAX 32767.0

class expand {
        protected:
                float alpha;
                float m;
                int limit;
                float coeff;
   public:
                expand(float ratio, int _limit, int max);
                int expand_out(int input);
};

#endif

//---------------------------------------------------------------

#include <math.h>
#include "expand.h"

expand :: expand(float ratio, int _limit, int max)
{
        alpha = ratio;
        if(!_limit)
                _limit = 1;
        limit = _limit;
        float dblimit = 20.0 * log10(limit / VMAX);
        m = dblimit * (1 - alpha);
        if(m) {
                float expand_max = VMAX * pow(10.0, (- m / (20 * alpha)));
                coeff = expand_max / max;
        }
        else
                coeff = 1;
}

int expand :: expand_out(int input)
{
        if(input * coeff <= limit * coeff && input * coeff >= -limit * coeff)
```

```
                return input * coeff;
        if(input * coeff > limit * coeff)
                return VMAX * pow(10.0, ((alpha * 20.0 *
                        log10(input * coeff / VMAX) + m) / 20));
        else
                return -32767 * pow(10.0, ((alpha * 20.0 *
                        log10(-input * coeff / VMAX) + m) / 20));
}

/*************************************************************************
KIRPICI
*************************************************************************/

//---------------------------------------------------------------
#ifndef distortH
#define distortH
#include "onepole.h"

class distortion {
        protected:
                short int treshold;
                double k;
                OnePole tone;
        public:
                distortion(double _treshold, double distort_tone, short int max);
                int distortlin_out(int input_data);
                int distortlog_out(int input_data);
};
#endif
//---------------------------------------------------------------

#include <math.h>
#include "distort.h"

distortion :: distortion(double _treshold, double distort_tone, short int max)
{
        treshold = (short int)(max * (1.0 - _treshold));
        k = treshold / log(treshold);
        tone.set_onepole(distort_tone);
}
int distortion :: distortlin_out(int input_data)
{
        if(input_data >= treshold)
                return tone.OnePole_Out(treshold);
        if(input_data <= -treshold)
                return tone.OnePole_Out(-treshold);
        else
```

```
                return tone.OnePole_Out(input_data);
}

int distortion :: distortlog_out(int input_data)
{
        if(input_data >= treshold)
                return tone.OnePole_Out(treshold);
        if(input_data <= -treshold)
                return tone.OnePole_Out(-treshold);
        if(input_data == 0)
                return tone.OnePole_Out(0);
        else
                if(input_data < 0)
                        return tone.OnePole_Out(-k * log(-input_data));
                else
                        return tone.OnePole_Out(k * log(input_data));
}

/*************************************************************************
BASİT PARAMETRİK SÜZGEÇ (BPS)
*************************************************************************/

//---------------------------------------------------------------

#ifndef filter1H
#define filter1H

#include <iostream.h>
#include <stdlib.h>
#include <math.h>
#include "delay.h"

class filter_one {

        protected:
                delayline d1;
                delayline d2;
                double alpha, alpha1, alpha2, beta, gama;

        public:
                filter_one();
                void set_filter_one(double teta, double q, int filter_chr);
                int filter_one_out(int input_data);
};

#endif
//---------------------------------------------------------------
```

```
#include "filter1.h"

filter_one :: filter_one()
{
        long delay_array[2] = {1L, 2L};
        d1.set_delayline(3L, 2, delay_array);
        d2.set_delayline(3L, 2, delay_array);
        alpha = 1.0;
        alpha1 = alpha2 = beta = gama = 0.0;
}

void filter_one :: set_filter_one(double teta, double q, int filter_chr)
{
        long delay_array[2] = {1L, 2L};
        d1.set_delayline(3L, 2, delay_array);
        d2.set_delayline(3L, 2, delay_array);
        double temp1 = 0.5 * q * sin(teta);
        double temp2 = tan(teta / (2.0 * q));
        if(filter_chr == 1) {
                beta = 0.5 * ((1 - temp1) / (1 + temp1));
                gama = (0.5 + beta) * cos(teta);
                alpha = alpha2 = 0.25 * (0.5 + beta - gama);
                alpha1 = 2.0 * alpha;
        }
        else
                if(filter_chr == 2) {
                        beta = 0.5 * ((1 - temp1) / (1 + temp1));
                        gama = (0.5 + beta) * cos(teta);
                        alpha = alpha2 = 0.25 * (0.5 + beta + gama);
                        alpha1 = -2.0 * alpha;
                }
                else
                        if(filter_chr == 3) {
                                beta = 0.5 * ((1 - temp2) / (1 + temp2));
                                gama = (0.5 + beta) * cos(teta);
                                alpha = 0.5 * (0.5 - beta);
                                alpha2 = -alpha;
                                alpha1 = 0.0;
                        }
                        else
                                if(filter_chr == 4) {
                                        beta = 0.5 * ((1 - temp2) / (1 + temp2));
                                        gama = (0.5 + beta) * cos(teta);
                                        alpha = alpha2 = 0.5 * (0.5 + beta);
                                        alpha1 = -gama;
                                }
}
```

```
int filter_one :: filter_one_out(int input_data)
{
        d1.delay_input(input_data);
        int temp = (int)(alpha * input_data + alpha1 * d1.delay_output(0) +
                alpha2 * d1.delay_output(1) + gama * d2.delay_output(0) -
                beta * d2.delay_output(1));
        d2.delay_input(temp);
        return temp;
}

/*****************************************************************************
TÜM GEÇİREN SÜZGEÇ
*****************************************************************************/

//---------------------------------------------------------------

#ifndef allpassH
#define allpassH

class allpass_one {
        protected:
                float allpass_coeff;
                int allpass_buff_in;
   public:
                allpass_one(float _coeff);
                allpass_one();
                void set_allpass_coeff(float _coeff);
                int new_out(int input);
};
#endif
//---------------------------------------------------------------

#include "allpass.h"

allpass_one :: allpass_one(float _coeff)
{
        allpass_coeff = _coeff;
        allpass_buff_in = 0;
}


allpass_one :: allpass_one()
{
        allpass_coeff = 1.0f;
        allpass_buff_in = 0;
}
```

```
void allpass_one :: set_allpass_coeff(float _coeff)
{
        allpass_coeff = _coeff;
        allpass_buff_in = 0;
}

int allpass_one :: new_out(int input)
{
        int out = input * allpass_coeff + allpass_buff_in * (1 - allpass_coeff * allpass_coeff);
        allpass_buff_in = input + allpass_buff_in * -1.0 * allpass_coeff;
        return out;
}

/**************************************************************************
MÜZİKAL SÜZGEÇ (MS)
**************************************************************************/

//---------------------------------------------------------------

#ifndef filter2H
#define filter2H

#include "allpass.h"
#include <math.h>

class filter_two {

        protected:

                double beta;
                double gama;
                double k;
                allpass_one ap1;
                int ap2_buff_in;

        public:

                filter_two(double wc, double q, double _k);
                filter_two();
                int filter_func(int func_in);
                void set_filter_two(double wc, double q, double _k);
                int filter_two_out(int input);
                int filter_two_out2(int input);
};

#endif
```

```
//-------------------------------------------------------------------
#include "filter2.h"
filter_two :: filter_two(double wc, double q, double _k)
{
        gama = -cos(wc);
        double temp = tan(wc / (2 * q));
        beta = (1 - temp) / (1 + temp);
        k = _k;
        ap1.set_allpass_coeff(gama);
        ap2_buff_in = 0;
}

void filter_two :: set_filter_two(double wc, double q, double _k)
{
        gama = -cos(wc);
        double temp = tan(wc / (2 * q));
        beta = (1 - temp) / (1 + temp);
        k = _k;
        ap1.set_allpass_coeff(gama);
        ap2_buff_in = 0;
}

filter_two :: filter_two()
{
        gama = k = beta = 0.0;
        ap1.set_allpass_coeff(0.0);
        ap2_buff_in = 0;
}

int filter_two :: filter_func(int func_in)
{
        int y2 = func_in - ap2_buff_in * beta;
        int out = func_in * beta + ap2_buff_in * (1 - beta * beta);
        ap2_buff_in = ap1.new_out(y2);
        return out;
}

int filter_two :: filter_two_out(int input)
{
        return (0.5 * input + (1 - k) * filter_func(0.5 * input)) * (2.0 / (1 + fabs(1 - k)));
}

int filter_two :: filter_two_out2(int input)
{
        input *= 0.5;
        return (1 - k) * filter_func(input) + (1 + k) * input;
}
```

```
/*************************************************************************
KARMAŞIK SAYI SINIFI
*************************************************************************/

//---------------------------------------------------------------
#ifndef __MYCOMPLEX_H
#define __MYCOMPLEX_H

class komplex {
        protected:
                float real;
                float image;
        public:
                komplex ( float _real, float _image);
                komplex ();
                komplex (const komplex &obj);
                void set_komplex(float _real, float _image);
                void set_real(float _real);
                void set_image(float _image);
                float get_real();
                float get_image();
                komplex& operator =(komplex &obj);
                komplex operator + (komplex &obj);
                komplex operator - (komplex &obj);
                komplex operator * (komplex &obj);
                komplex& exp(double rotator);
                void show();
};
#endif
//---------------------------------------------------------------

#include <math.h>
#include <stdlib.h>
#include "mycomp.h"

komplex :: komplex(float _real, float _image)
{
        real = _real;
        image = _image;
}

komplex :: komplex()
{
        real = 0.0;
        image = 0.0;
}
```

```
komplex ::komplex (const komplex &obj)
{
        real = obj.real;
        image = obj.image;
}

void komplex :: set_komplex(float _real, float _image)
{
        real = _real;
        image = _image;
}

void komplex :: set_real(float _real)
{
        real = _real;
}

void komplex :: set_image(float _image)
{
        image = _image;
}

float komplex :: get_real()
{
        return real;
}

float komplex :: get_image()
{
        return image;
}

komplex& komplex :: operator =(komplex &obj)
{
        real = obj.real;
        image = obj.image;
        return *this;
}

komplex  komplex :: operator + (komplex &obj)
{
        komplex temp;
        temp.real = real + obj.real;
        temp.image = image + obj.image;
        return temp;
}
```

```
komplex  komplex :: operator - (komplex &obj)
{
        komplex temp;
        temp.real = real - obj.real;
        temp.image = image - obj.image;
        return temp;
}

komplex  komplex :: operator * (komplex &obj)
{
        komplex temp;
        temp.real = real * obj.real - image * obj.image;
        temp.image = real * obj.image + image * obj.real;
        return temp;
}

komplex& komplex :: exp(double rotator)
{
        real =  cos(rotator);
        image = -sin(rotator);
        return *this;
}

/*************************************************************************
HIZLI FOURIER DÖNÜŞÜMÜ (HFD)
*************************************************************************/

//---------------------------------------------------------------
#ifndef __FFT_H
#define __FFT_H

class fft {
        protected:
                unsigned int data_length;
                komplex * source;
        public:
                fft(unsigned int _data_length, komplex * _source);
                void bit_reverse();
                void butterfly(unsigned int fft_size);
                void osman_bulent_fft();
};
#endif
//---------------------------------------------------------------

#include "fft.h"
#include "mycomp.h"
#include <conio.h>
```

```
#include <iostream.h>
#include <math.h>
#include <stdlib.h>

fft :: fft(unsigned int _data_length, komplex * _source)
{
        data_length = _data_length;
        source = _source;
}

void fft :: bit_reverse()
{
        float rtemp;
        unsigned int i,j,m;

        for(i = j = 0; i < data_length; i += 1, j += m) {
                if(j > i) {
                        rtemp = source[j].get_real();
                        source[j].set_real(source[i].get_real());
                        source[i].set_real(rtemp);
                }
                for(m = data_length >> 1; m >= 2 && j >= m; m >>= 1)
                        j -= m;
        }
}

void fft :: butterfly(unsigned int fft_size)
{
        unsigned int step_size = fft_size / 2;
        komplex temp;
        komplex coeff;
        for(unsigned int i = 0; i < data_length; i += fft_size)
                for(unsigned int j = 0; j < step_size; ++j) {
                        coeff = coeff.exp(2.0 * M_PI * j / fft_size) *
                                source[i + j + step_size];
                        temp = source[i + j] + coeff;
                        source[i + j + step_size] = source[i + j] - coeff;
                        source[i + j] = temp;
                }
}

void fft :: osman_bulent_fft()
{
        bit_reverse();
        for(unsigned int fftsize = 2; fftsize <= data_length; fftsize *= 2)
                butterfly(fftsize);
}
```

```
/*************************************************************************
 SES DALGASI DOSYASI OLUŞTURMADA KULLANILAN SINIF
*************************************************************************/
//-------------------------------------------------------------
#ifndef __WAVWRITE_H
#define __WAVWRITE_H

class wav_write {
        protected:
                unsigned long size_of_data;
                unsigned long size_of_block;
                unsigned long sampling_rate;
                unsigned short number_of_channel;
                unsigned short number_of_bit_per_sample;
                unsigned char *file_name;
        public:
                wav_write(unsigned short channel, unsigned long samprate,
                        unsigned short bps, unsigned long datasize);
                void write_header(unsigned char *ptr);
                void write_datablock(short int *data_block);
                void write_datablock8(unsigned char *data8_block);
};
#endif
//-------------------------------------------------------------
#include <stdio.h>
#include <math.h>
#include <iostream.h>
#include "wavwri1.h"
#include <stdlib.h>

wav_write :: wav_write(unsigned short channel, unsigned long samprate,
unsigned short bps, unsigned long datasize)
{
        number_of_channel = channel;
        sampling_rate = samprate;
        number_of_bit_per_sample = bps;
        size_of_data = datasize;
        size_of_block = datasize * channel;
}

void wav_write :: write_datablock(short int *data_block)
{
        FILE *fptr;
        if((fptr = fopen(file_name, "ab")) == NULL){
                cout << "Dosya Açılamadı!!!" << endl;
                exit(1);
        }
```

```
        fwrite(data_block, sizeof(short int), size_of_block, fptr);
        fclose(fptr);
}

void wav_write :: write_datablock8(unsigned char *data8_block)
{
        FILE *fptr;
        if((fptr = fopen(file_name, "ab")) == NULL){
                cout << "Dosya Açılamadı!!!" << endl;
                exit(1);
        }
        fwrite(data8_block, sizeof(unsigned char), size_of_block, fptr);
        fclose(fptr);
}

void wav_write :: write_header(unsigned char *ptr)
{
        FILE *fp;
        file_name = ptr;
        if((fp = fopen(file_name, "wb")) == NULL){
                cout << "Dosya Açılamadı!!!" << endl;
                exit(1);
        }
        fwrite("RIFF", sizeof(unsigned char), 4, fp);
        unsigned long riff_size = 36 + ((number_of_channel * number_of_bit_per_sample /
                8) * size_of_data);
        fwrite(&riff_size, sizeof(unsigned long),1,fp);
        fwrite("WAVE",sizeof(unsigned char),4,fp);
        fwrite("fmt ",sizeof(unsigned char),4,fp);
        unsigned long var = 16;
        fwrite(&var, sizeof(unsigned long),1,fp);
        unsigned short format_tag = 1;
        fwrite(&format_tag,sizeof(unsigned short),1,fp);
        fwrite(&number_of_channel,sizeof(unsigned short),1,fp);
        fwrite(&sampling_rate, sizeof(unsigned long),1,fp);
        unsigned long average_byte_per_second = (number_of_channel * sampling_rate *
                number_of_bit_per_sample) / 8;
        fwrite(&average_byte_per_second, sizeof(unsigned long),1,fp);
        unsigned short nblock_align = (number_of_channel *
                number_of_bit_per_sample) / 8;
        fwrite(&nblock_align,sizeof(unsigned short),1,fp);
        fwrite(&number_of_bit_per_sample,sizeof(unsigned short),1,fp);
        fwrite("data",sizeof(unsigned char),4,fp);
        unsigned long data_length = riff_size - 36;
        fwrite(&data_length, sizeof(unsigned long),1,fp);
        fclose(fp);
}
```

ÖZGEÇMİŞ

Doğum tarihi	1.8.1971	
Doğum yeri	İstanbul	
Lise	1985-1988	Kocamustafapaşa Lisesi
Lisans	1989-1994	Yıldız Teknik Üniversitesi Elektrik-Elektronik Fak. Elektronik ve Hab. Müh. Bölümü
Yüksek Lisans	1995-1998	Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü Elektr. ve Hab. Müh. Anabilim Dalı, Haberleşme Programı
Çalıştığı kurum(lar)		
	1997-Devam ediyor	Yıldız Teknik Üniversitesi Elektr. ve Hab. Müh. Bölümü Telekomünikasyon Anabilim Dalı Araştırma Görevlisi