T.C.
YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

KARMAŞIK AYRIK DALGACIK DÖNÜŞÜMÜNÜN YENİDEN YAPILANDIRILABİLİR MİMARİLER ÜZERİNDE GERÇEKLENMESİ

FERHAT CANBAY

DOKTORA TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI
BİLGİSAYAR MÜHENDİSLİĞİ PROGRAMI

DANIŞMAN
PROF. DR. NİZAMETTİN AYDIN

İSTANBUL, 2016

T.C.

YILDIZ TEKNİK ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

KARMAŞIK AYRIK DALGACIK DÖNÜŞÜMÜNÜN YENİDEN YAPILANDIRILABİLİR MİMARİLER ÜZERİNDE GERÇEKLENMESİ

Ferhat CANBAY tarafından hazırlanan tez çalışması 27.04.2016 tarihinde aşağıdaki jüri tarafından Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı'nda **DOKTORA TEZİ** olarak kabul edilmiştir.

Tez Danışmanı

Prof. Dr. Nizamettin AYDIN

Yıldız Teknik Üniversitesi

Eş Danışman

Doç. Dr. Sezer GÖREN UĞURDAĞ

Yeditepe Üniversitesi

Jüri Üyeleri

Prof. Dr. Nizamettin AYDIN

Yıldız Teknik Üniversitesi _____

Prof. Dr. Hasan Hüseyin BALIK

Yıldız Teknik Üniversitesi _____

Prof. Dr. Tamer ÖLMEZ

İstanbul Teknik Üniversitesi _____

Doç. Dr. Songül ALBAYRAK

Yıldız Teknik Üniversitesi _____

Doç. Dr. Hasan Fatih UĞURDAĞ

Özyeğin Üniversitesi _____

Bu çalışma, Yıldız Teknik Üniversitesi Bilimsel Araştırma Projeleri Koordinatörlüğü' nün 2014-04-01-DOP03 numaralı projesi ile desteklenmiştir.

ÖNSÖZ

Lisans ve Lisansüstü eğitimim süresince bana akademik konularda yol gösteren, özellikle doktoraya başlamama vesile olan, sabrını, hoşgörüsünü ve desteğini her zaman bana hissettiren değerli tez danışmanım Nizamettin AYDIN Hocama; akademik kariyere başlamama ve şu an ilgilendiğim alanda yetişmeme vesile olan, verdikleri fikirler ve bakış açıları ile benim için her zaman birer ağabey ve abla olan Fatih UĞURDAĞ ve Sezer GÖREN UĞURDAĞ Hocalarıma; tez çalışmalarım sırasında tıkandığım her an bana destek olan Görkem SERBES dostuma ve teknik kısımlarda bana destek olan öğrencim Vecdi Emre LEVENT'e; benim bugünlere gelmemi, fedakârlıkları ve çabaları ile sağlayan anneme, babama ve kardeşlerime; en önemlisi, hayatımın son dokuz yılında bana en zorlandığım zamanlarda yol arkadaşı olan, sonsuz sevgi ve desteğini esirgemeyen eşime, şükranlarımı sunuyorum.

Bu tezi hayatımın neşeleri, sevgili çocuklarım Ömer Mirza ve Yusuf Kerem'e ithaf ediyorum.

Nisan, 2016

Ferhat CANBAY

İÇİNDEKİLER

Sayfa

KISALTMA LİSTESİ

ADD	Ayrık Dalgacık Dönüşümü
Bİ	Biyomedikal İşaret
ÇAKDD	Çift-Ağaç Dalgacık Dönüşümü
DÇAKDD	Değiştirilmiş Çift-Ağaç Dalgacık Dönüşümü
EEG	Elektroensefalografi
EKG	Elektrokardiyografi
FPGA	Alanda Programlanabilir Kapı Dizileri
KÜY	Kod Üretici Yazılım
KZFD	Kısa Zamanlı Fourier Dönüşümü
LUT	Başvuru Çizelgesi
SDD	Sürekli Dalgacık Dönüşümü
Sİİ	Sayısal İşaret İşleme

ŞEKİL LİSTESİ

Sayfa

ÇİZELGE LİSTESİ

Sayfa

ÖZET

KARMAŞIK AYRIK DALGACIK DÖNÜŞÜMÜNÜN YENİDEN YAPILANDIRILABİLİR MİMARİLER ÜZERİNDE GERÇEKLENMESİ

Ferhat CANBAY

Bilgisayar Mühendisliği Anabilim Dalı

Doktora Tezi

Tez Danışmanı: Prof. Dr. Nizamettin AYDIN

Eş Danışman: Doç. Dr. Sezer GÖREN UĞURDAĞ

Çift-Ağaç Karmaşık Ayrık Dalgacık Dönüşümü (ÇAKDD), işaret işleme uygulamalarında yaygın olarak kullanılan bir yöntemdir. Ayrık Dalgacık Dönüşümünün (ADD) daha gelişmiş bir türü olan ÇAKDD, paralel çalışan iki adet ADD'den oluşmaktadır. ADD ise Sonlu Darbe Cevaplı (Finite Impulse Response - FIR) filtrelerden oluşan bir ikili ağaçtır. Tezde, ÇAKDD'nin düşük güç tüketimli, taşınabilir uygulamalar için yeniden yapılandırılabilir bir platform üzerinde geliştirilmesi hedeflenmiştir. Bu kapsamda ÇAKDD, sonuçların doğrulanabilmesi için kişisel bir bilgisayarda, mikrodenetleyici (PIC) üzerinde, ARM tabanlı bir gömülü sistem üzerinde ve Alanda Programlanabilir Kapı Dizileri (Field Programmable Gate Arrays - FPGA) üzerinde gerçeklenmiştir. FPGA platformunda ise üç farklı donanım mimarisi (her ADD ağacı için bir toplayıcı ve bir çarpıcı, her ÇAKDD kanalı için bir toplayıcı ve bir çarpıcı, tüm girdi kanalları için bir toplayıcı ve bir çarpıcı) gerçekleştirilmiştir. Geliştirilen tüm mimariler çok kanallı olarak çalışabilmektedir. Buna ek olarak, üç mimarinin kanal sayısı ve bit genişlikleri gibi parametrelerinin belirlenerek oluşturulduğu Saklayıcı Transfer Dili (Register Transfer Language - RTL) için kod üretici yazılım geliştirilmiştir. Üretilen ÇAKDD RTL, FPGA hedeflenerek yazılmıştır. Yöntemlerde kullanılan toplayıcı ve çarpıcıların sayısının azalmasıyla alan verimliliği ve gecikme artmaktadır. N-kanal işareti işlemek için, ilk yöntem, ikinci yöntemin iki katı hıza sahipken, ikinci yöntem üçüncü yöntemin N katı kadar hızlıdır. Ethernet kullanılarak bilgisayardan FPGA'ye veriler gönderilip sonuçlar

başarılı bir şekilde alınarak doğrulama yapılmıştır. Ayrıca ÇAKKD'nin daha gelişmiş bir sürümü olan Değiştirilmiş ÇAKKD algoritması aynı mimariler kullanılarak gerçekleştirilmiştir.

Anahtar Kelimeler: İşaret İşleme, Karmaşık Dalgacık Dönüşümü, Biyomedikal, FPGA, Gerçek Zamanlı

ABSTRACT

IMPLEMENTATION OF COMPLEX DISCRETE WAVELET TRANSFORM ON RECONFIGURABLE ARCHITECTURES

Ferhat CANBAY

Department of Computer Engineering

Ph.D. Thesis

Adviser: Prof. Dr. Nizamettin AYDIN

Co-Adviser: Assoc. Prof. Dr. Sezer GÖREN UĞURDAĞ

Dual Tree Complex Wavelet Transform (DTCWT) is widely used in signal processing applications. DTCWT is an enhancement to the Discrete Wavelet Transform (DWT) and composed of two DWTs running in parallel. DWT is formed as a binary tree of Finite Impulse Response (FIR) filters. In this thesis, a low power DTCWT implementation on a reconfigurable platform for portable applications is aimed. In this respect, in order to verify the results DTCWT was implemented on a personal computer, on a microcontroller, on an embedded system based on ARM processor, and on a Field Programmable Gate Arrays (FPGA). Three hardware architectures (an adder and a multiplier for each DWT tree, an adder and a multiplier for each DTCWT, an adder and a multiplier for N-channel DTCWT) were implemented. All these architectures can be implemented as multi-channel applications. Additionally, a code generator that produces RTL codes for desired architecture was implemented. We synthesized the generated DTCWT RTL for the targeted FPGA. Area efficiency increases with decreasing number of adders and multipliers utilized in the proposed architectures with some penalty in latency. For processing N-channel signals, a DTCWT using the first implementation is two times faster than the second implementation, a DTCWT using the second implementation is N times faster than the third implementation. In order

to verify the implementations, data is sent to FPGA from a computer using Ethernet and results are received succesfully in real-time. Also, Modified DTCWT algorithm was implemented with the same architectures.

Keywords: Signal Processing, Complex Discrete Wavelet Transform, Biomedical, FPGA, Real-Time

BÖLÜM 1

GİRİŞ

İşaret, bilgi taşıyan bir fiziksel değişken olarak tanımlanabilir. İşaret işleme, işaretin elde edildiği kaynağın mahiyeti hakkında taşıdığı bilgiyi elde etme ya da bir kaynağa enerji aktarmak için işarete bilgi yükleme işlemlerini kapsar. Bu işlemler analog ya da sayısal olarak yapılabilir. Ancak günümüzde, sayısal bilgisayarlar konusundaki gelişmelere paralel olarak sayısal işaret işleme teknikleri çok daha fazla tercih edilmektedir. Sayısal işaret işleme tekniklerinin yaygın olarak kullanıldığı alanlardan biri de tıptır. Hastalıkların teşhis ve tedavisinde biyomedikal cihazlar önemli bir yer tutmaktadır. Bu cihazlardaki önemli aşamalardan biri işaretin elde edildiği organ ya da organ sistemleri hakkında bilgi taşıyan fizyolojik işaretlerin ölçülmesi ve işlenmesidir. Fizyolojik sistemler zamana bağlı olarak değişkenlik gösterirler. Bundan dolayı fizyolojik işaretlerin istatistiksel nitelikleri zamana bağlı olarak değişir. Fizyolojik işaretleri daha doğru çözümleyebilmek için işaretin zamana bağlı olarak genlik, frekans ve faz değişimlerinin eşzamanlı olarak incelenmesi gerekir. Bu incelemeyi yapabilmek için literatürde önerilen birçok yöntem vardır. Kısa zamanlı (veya pencerelenmiş) Fourier dönüşümü (KZFD, Short Time Fourier Transform) bu yöntemlerden en yaygın olarak kullanılanıdır. İşaretin özelliklerine göre KZFD'nin zaman-frekans çözünürlüğü belirlenir ve çözümleme boyunca sabit tutulur. Ancak KZFD işaretin durağan olduğunu varsayarak işlem yaptığı için durağan olmayan işaretlerin işlenmesinde uygun bir yöntem değildir. Alternatif olarak, ayarlanabilir zaman-frekans çözünürlüğü özelliği olan sürekli dalgacık dönüşümü (SDD) durağan olmayan işaretleri işlemek için sıkça kullanılan bir yöntemdir [5]. SDD, düşük frekanslardaki işaretler için yüksek frekans çözünürlüğü, yüksek frekanstaki işaretler için yüksek zaman çözünürlüğü sağlayarak

işaretin en uygun şekilde çözümlenmesine olanak verir [6]. SDD'de analiz edilen işaretler, dalgacık adı verilen özel bir fonksiyonun kullanılması ile temsil edilir. Bu dalgacıkların, ölçekleri ve zamanda yerleri değiştirilerek üzerinde çalışılan işaretin, düşük ve yüksek frekanslı bileşenlerinin özellikleri yakalanabilir. Ancak, SDD'deki sürekli zaman kayması ve ölçekleme değişikliklerine bağlı olarak, bu işlemler hesaplama maliyetine ve zaman kaybına neden olmaktadır. Anlatılan gerekçeler dolayısıyla, genelde gerçek zamanlı uygulamalarda, bellek ihtiyacını azaltmak ve işlem hızını arttırmak amacıyla, ölçek ve çeviri parametrelerinin ayrıklaştırıldığı Ayrık Dalgacık Dönüşümü (ADD) kullanılır [7]. ADD, literatürde Biyomedikal İşaretler (Bİ) için ön işleme, gürültü giderme ve öznitelik çıkarma basamaklarında sıkça kullanılmaktadır [8]. Ancak ADD, analiz edilen işaretlerdeki faz kaymalarına aşırı duyarlıdır ve bu durum ayrıştırılmış ara-bantlardaki dalgacık katsayılarının enerji dağılımını bozmaktadır. Örnek vermek gerekirse, [9]'daki gibi embolik bir işaretin ön-işleme adımında ADD uygulandığında elde edilen gürbüz olmayan katsayılar gürültü süzme performansını azaltır. Bu olumsuz durumun üstesinden gelmek için, ADD'nin yeni bir gerçekleştirimi olan çift-ağaç karmaşık dalgacık dönüşümü (ÇAKDD) kullanılabilir [10]. ÇAKDD kullanılarak, çözümleme kısmı sonucunda, Bİ'lerden çok daha gürbüz dalgacık katsayıları çıkarılabilir [11,12].

Bazı biyomedikal işaretler (örneğin EEG), birden fazla bölgeden elde edilen kayıtlardan oluşmaktadır. Bu tür (çok-kanallı) işaretlerin eşzamanlı olarak işlenmesi önemlidir. Toplanan Bİ'ler hastalık teşhisi, izlemesi ve tedavilerinde sağlık hizmeti kalitesini arttırmak için kullanılır. Bİ'lerde gerçek zamanlı işlemler çok hayati bir ihtiyaçtır. Çünkü bir saniyeden bile kısa bir gecikme, hatalı bir yorumlama sonucu yanlış teşhise sebebiyet verebilir. Böyle durumlarda ölüm vakalarının görülmesi de ihtimal dâhilindedir. Bu nedenle, gerçek zamanlı gömülü sistemler Bİ'lerin işlenmesinde gittikçe önemli bir rol almaya başlamıştır. Bu sistemler, beyin-bilgisayar etkileşimi [1] veya gerçek zamanlı elektrokardiyogram (EKG) gözlemleme gibi karmaşık problemlerin çözümünde kullanılmaktadır [2].

Vücudun çeşitli bölgelerinden gelen eşzamanlı işaretlerin bir şekilde gerçek zamanlı olarak toplanması ve işlenmesi için çok-kanallı biyo-potansiyelleri ölçebilen taşınabilir cihazlara ihtiyaç vardır. Çok-kanallı taşınabilir sistemlere; giyilebilir, hafif, düşük güç

tüketimli ve girdi kanallarının sayısının ayarlanabilir olması gibi özellikleri nedeniyle ihtiyaç duyulmaktadır. Bu ihtiyaçları karşılayabilmek için çok-kanallı gerçek zamanlı Bİ edinme ve işleme sistemlerinde alanda programlanabilir kapı dizileri (FPGA) tabanlı çözümler literatürde [3,4] kullanılmıştır.

Literatürde, çok-kanallı veri toplanmasına olanak veren, gerçek zamanlı, düşük maliyetli ve düşük güç tüketimli FPGA tabanlı işaret işleme sistemlerine ihtiyaç duyulmaktadır. Elektroensefalografik işaretler (128 kanallı) [13], iğnecik treni işaretleri (94 kanallı) [14] ve akciğer işaretleri (14 kanallı) [11] çok-kanallı işaretlere örnek olarak verilebilir.

Yukarıda anlatılanlar bağlamında bu tezde, ÇAKDD algoritmasının FPGA platformunda gerçek zamanlı olarak gerçekleştirilmesi için taşınabilir uygulamaları göz önüne alarak geliştirilmiş üç farklı mimari önerilmiştir. Bu mimarilerde, düşük-güç tüketimi ve alan verimliliği ön planda tutulmuştur. Ek olarak ise; geliştiricilerin yapmış oldukları çalışmalarında ÇAKDD yönetimini sistemlerinde kolay ve hızlı bir şekilde kullanabilmesi için esnek bir yapı kurgulanmış ve kod üretici bir yazılım geliştirilerek gerekli parametrelerin girilmesiyle ihtiyaç duyulan mimarinin kolayca elde edilebilmesi sağlanmıştır.

1.1 Literatür Özeti

Biyomedikal alanında kullanılan cihazların geçmişi 19. yüzyıla kadar dayanmaktadır. Bu cihazlar, canlıların fizyolojik yapısının öğrenilmesi, işleyişi hakkında bilgi edinilmesi ve bu bilgiler ışığında normal olmayan durumların teşhis ve tanısının konulması aşamalarında büyük önem arz etmektedir. Günümüzde olmazsa olmaz konumunda olan biyomedikal cihazların temel prensibi bazen yalnızca bir elektriksel işaret, ses, sıcaklık vb. gibi ölçümler yapmak bazen de bir uyarıcı kullanarak verilen reaksiyonu ölçmek şeklinde olabilmektedir. Ölçüm işlemleri elektrotlar veya dönüştürücüler aracılığıyla yapılır. Organizmada asıl ölçülmek istenen bölgenin dışında birçok öğe de bulunduğu için elde edilen ham işaretin içerisinde istenmeyen veriler de bulunmaktadır. Bu veriler gürültü olarak kabul edilir ve öncelikle işaretin gürültü kabul edilen verilerinin süzülmesi gerekmektedir (ön işaret işleme). Süzülen işaretler daha

sonra amaca yönelik olarak tekrar işlenir. İşlenen işaret yorumlanarak veya gruplandırılarak bir çıktı oluşturulur ve elde edilen çıktıya göre tanı konulabilir.

İşaret işleme safhasında çeşitli yöntemler kullanılmaktadır. Bir işaretin frekansı Fourier Dönüşümü kullanılarak hesaplanır. Frekans bilgisinin gerekliliğine örnek olarak EKG işaretleri verilebilir. EKG, kalbin elektriksel aktivitesinin grafik kayıtlarını vermektedir. Fourier dönüşümü, hangi frekans bilgisinin hangi anda var olduğunu göstermediği için değişken frekanslı işaretler için uygun bir teknik değildir. Bu tür durumlarda KZFD, gibi dönüşümlere ihtiyaç duyulmaktadır.

Günümüzde biyomedikal sistemler çok hızlı bir şekilde gelişmektedir. Tek bir sensörden gelen veriyi değil aynı anda birden çok sensörden eşzamanlı gelen veriyi işleyen cihazlar kullanılmaktadır. Problemlerin daha da karmaşıklaşması ve işlenmesi gereken verinin gittikçe büyümesi yeni çözümlerin üretilmesini gerektirmektedir. Mikroişlemciler, Sayısal İşaret İşleme (Sİİ) işlemcileri gibi cihazlar, sınırlı kapasiteleri itibariyle bahsi geçen problemlere gerçek zamanlı bir çözüm olamamaktadır. Gerçek zamanlı sistemlerde, kapasitesi, hızı, tasarım esnekliği gibi üstün yanlarıyla FPGA platformu öne çıkmaktadır. Tez çalışması kapsamında FPGA platformunda ADD'nin geliştirilmiş bir versiyonu olan ÇAKDD yöntemi gerçeklenmiştir.

Literatürde ADD'nin bir takım gerçeklemeleri daha önce yapılmıştır. Örneğin, [15]'te ADD çerçeve bölümlenmiş mimari kullanılarak FPGA'de gerçeklenmiş ve simülasyon sonuçları verilmiştir. Bu çalışmada Ping-Pong mimarisi tabanlı, 2'li tampon bellek yapısı kurgulanmış ve alan verimliliği düşünülmüştür. [16]'da ADD, iç içe geçmiş zaman tabanlı olarak seri-bit mimarisi kullanılarak gerçeklenmiştir. [17]'de LUT tabanlı bir FPGA mimarisinden faydalanılarak ve dağıtılmış aritmetik algoritmasına uygun olarak dalgacık hesaplama yeniden formüle önerilmiş bir ADD sunulmuştur. [18]'de ise hız verimliliğinin ön planda olduğu, her seviyede alçak ve yüksek geçiren filtrelerin ayrı ayrı ve aynı anda çalıştığı bir mimari düşünülmüştür. [3]'de yüksek-yoğunluklu sinirsel algılama uygulamaları üzerine bir çalışma yapılmıştır. Bu çalışma içerisinde bu tezin konusuna yakın bir mimari kullanılmıştır. Ancak bu mimari geliştirilen sistemin içerisinde kullanılmış olan bir alt modüldür. Tez kapsamındaki gibi esnek ve farklı amaçlara uygun olarak tasarlanmış bir yapı değildir. [4]'de beyin işaretlerinin

sıkıştırılarak iletilmesi amacıyla FPGA platformunda ADD gerçeklenmiştir. Mimari 32 kanallı olup her kanal 4 seviyeden oluşmaktadır. Veriler 10 bit genişliğe sahiptir.

Görüldüğü gibi, genel olarak ayrık dalgacık dönüşümü ve ayrık dalgacık paket dönüşümünün FPGA'e dayalı çözümleri önceden önerilmiştir. Ancak bu çalışmalarda sabit bir mimari ve sabit veri genişlikleri kullanılmıştır. Bu tezde ise, FPGA platformu üzerinde gerçekleştirimi literatürde henüz olmayan ÇAKDD algoritmasının gerçekleştirilimi için farklı mimariler önerilmektedir. Önerilen bu mimarilerde bit genişliğinin ve kanal sayısının ayarlanabilmesi, ihtiyaç duyulan hıza göre asgari alan kullanımının desteklenmesi ve bunlara ek olarak her kanalın istendiğinde farklı katsayı setleri ile bağımsız çalışabilmesi gibi birçok üstün özellik bulunmaktadır. Herhangi bir araştırmacının çalışmasında ihtiyaç duyduğu giriş veri hızı (input data rate) aralığının bu çalışmalardaki ile uyumlu olması halinde ÇAKDD'nün donanımsal gerçeklemesini çok basit ve hızlı bir şekilde kendi çalışmasında kullanabilecektir.

1.2 Tezin Amacı

Biyomedikal alanında canlılardan bir takım veriler toplanmaktadır. Elde edilen veriler genel itibariyle organizmaya yerleştirilen almaçlar tarafından aynı anda birden fazla kaynaktan gelmektedir. Bu farklı işaretlerin aynı anda işlenmesi ve bu tür işaret işleme sistemlerinin taşınabilir sistemlere entegre edilebilmesi önem kazanmıştır. Bu bağlamda çoğu tıbbi cihazların taşınabilir, hızlı, düşük güç tüketimli olması beklenmektedir. Tezin temel amacı biyomedikal alanında geniş bir kullanım imkânı bulunan ÇAKDD algoritmasının, birden fazla kanaldan gelen verileri paralel olarak gerçek zamanda işleyebilecek, bunu en düşük maliyetle ortaya koyacak ve taşınabilir sistemlere entegre edilebilecek şekilde geliştirilmesidir.

Yukarıdakilere ek olarak genel kullanıma açık olan bir kod üretici yazılım geliştirilerek araştırmacıların ihtiyacına uygun tasarımın, girilen parametrelere göre üretilmesi hedeflenmektedir. Bu sayede araştırmacılar ve geliştiriciler için, çalışmalarında ÇAKDD modülünü herhangi bir donanım tasarım tecrübesi olmadan kolay ve hızlı şekilde uyarlayabilecekleri bir çözüm sunulmuş olacaktır.

1.3 Hipotez

Teknolojik gelişmelerin artması ile canlılardan birçok işaret toplanmaktadır. Bu işaretlerin çeşitli arayüzlere aktarılması veya farklı sistemlerle birlikte çalışması ihtiyacı gün geçtikçe artmaktadır. Bİ'ler ile çalışılırken taşınabilir, az yer kaplayan ve aynı zamanda hız konusunda verimli gerçek zamanlı sistemlere gereksinim duymaktadır. Tezin temel hipotezi, yukarıda sayılan ihtiyaçlar göz önüne alınarak özellikle çok kanallı Bİ'lerin gerçek zamanda işlenebilmesi ve sonuçların diğer arayüzlerle doğrudan entegre edilebilmesi için en uygun çözümün FPGA tabanlı sistemler olduğudur.

1.4 Sayısal İşaret İşleme

İşaret, kısaca bilgi taşıyan her şeydir denebilir. Konumuzda ilgilendiğimiz anlamıyla işaret, bir veya birden fazla değişkene sahip gerçek ya da karmaşık değerli bir fonksiyondur. İşaretin taşıdığı bilgiyi elde edebilmek veya değerlendirebilmek için bir takım hesaplamaların yapılması gerekmetedir. Yapılan bu tür işlemlere genel olarak işaret işleme denir. Pratikteki pek çok işaret zaman uzayında tanımlanır [19]. Bir başka deyişle işaretin grafiği göz önüne alındığında eksenlerden biri zaman (bağımsız değişken), diğeri ise genliktir (bağımlı değişken). İşaret işleme ile ilgili birçok uygulamada zaman-genlik gösterimi ile sonuca gitmek zordur. Çoğu durumda en önemli bilgi frekans bileşeninde yatmaktadır.

1.4.1 Fourier Dönüşümü

Bir işaretin frekansı Fourier Dönüşümü kullanılarak hesaplanır. Zaman alanındaki bir işaretin Fourier dönüşümü alındığı takdirde o işaretin frekans-genlik bilgileri elde edilir. Bir başka deyişle Fourier dönüşümü sonunda elde edilen bilgi eksenlerinden biri frekans, diğeri ise genliktir. Bu bilgiler, işarette her frekanstan ne kadar bulunduğunu göstermektedir.

Fourier dönüşümü iki yönlü bir dönüşümdür. Bir işarete Fourier dönüşümü uygulandığında işaret frekans uzayında ifade edilmiş olur. Frekans uzayındaki bu işarete ters Fourier dönüşümü uygulandığında ise, işaretin zaman uzayındaki ifadesine tekrar dönülmüş olur. Ancak belirli bir anda işaret ya zaman uzayında ya da frekans

uzayında ifade edilebilir. Bir başka deyişle zaman uzayı grafiğinde frekans bilgisi alınamazken, zaman bilgisine de Fourier dönüşümü uygulanmış işaretten ulaşmak mümkün değildir.

Sabit frekanslı işaretlerde Fourier dönüşümünün zaman hakkında bilgi verememesinin önemi yoktur. Frekans zamanla değişmediğinden dolayı frekans o işaret boyunca sürekli vardır.

Çoğu gerçek işaretlerin (biyomedikal işaretler gibi) frekans ve genlikleri zamanla değişiklik göstermektedir. Tek başına zaman uzayındaki ya da frekans uzayındaki gösterimleri, bu gibi işaretlerin özelliklerini ve niteliklerini anlamak için yeterli değildir. Bu yüzden, işaretlerin hem genlik hem de frekans bilgilerinin birlikte görülebilmesini sağlayan KZFD ve Wigner dağılımları gibi zaman-frekans çözümlemesi yöntemleri önerilmiştir. Bu yöntemlerden biri de dalgacık dönüşümüdür. Dalgacık dönüşümü zaman-frekans bilgisini verebilmektedir.

1.4.2 Dalgacık Dönüşümü

Dalgacık dönüşümü, KZFD'nin çözünürlük problemini kısmen çözen bir dönüşümdür. Düşük frekanstaki işaretler için yüksek frekans çözünürlüğü yüksek frekanstaki işaretler için yüksek zaman çözünürlüğü özelliğinden dolayı optimum zaman-frekans çözünürlüğü sağlar. Pratik olarak dalgacık dönüşümü, kısa zamanlı Fourier dönüşümü ile aynı biçimde uygulanır. İşaret, fonksiyon ile çarpılır ve sonuçlar toplanır [20].

Dalgacık dönüşümü, frekans bilgilerinin yanı sıra konumsal bilgileri de verebilmektedir. Bir işaret, dalgacık dönüşümü alınarak elde edilen katsayılar kullanılarak veride herhangi bir kayıp ya da bozulma olmadan geri çatılabilir. Dalgacık analizindeki temel fikir, bir işaretin fonksiyonuna bağlı olarak bileşenlerine ayırmaktır [21,22].

Dalgacık dönüşümü sürekli ve ayrık Dalgacık dönüşümü olmak üzere iki çeşittir. Sürekli Dalgacık dönüşümü Fourier dönüşümü gibi integraller ve analitik eşitlikler yardımıyla hesaplanmaktadır. Günümüzde bu hesaplamalar bilgisayarlarla yapıldığından dolayı, bu dönüşümlerin ayrık çeşitleri de türetilmiştir.

Ana Dalgacık olarak kullanılabilecek pek çok temel fonksiyon vardır. Dönüşümde kullanılacak olan bütün fonksiyonlar ana Dalgacık tarafından üretildiği için, Dalgacık

dönüşümünün karakteristiğini ana Dalgacık belirleyecektir. Bu yüzden uygulamalarda uygun Dalgacık türünün seçilmesi büyük önem taşımaktadır. Uygulamalarda Dalgacık türü seçilirken Dalgacık'ın şekline ve analiz edebilme yeteneğine bakılır [27].

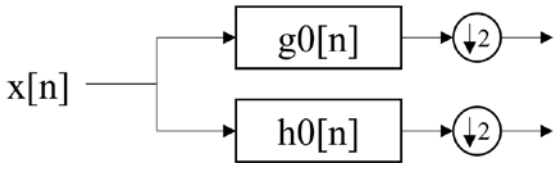
Ayrık Dalgacık Dönüşümü (ADD), dalgacık dönüşümünün ölçekleme parametresinin ikinin katları şeklinde gerçekleştirilmesi ile elde edilir. ADD'nin hızlı gerçeleştirilmesi için geliştirilmiş algoritmalar vardır [ref-fast wavelet transform]. Bu algoritmalarda işaret bir alçak bir de yüksek geçiren filtreden geçirilerek (eşitlik 1.2 ve 1.3) ve sonrasında örnek sayısı yarıya düşürülerek ADD gerçekleştirilir [22,23]. İşaretin uzunluğu *N* ise, bu işlem log_2N kere gerçekleştirilir. Bu işlem sonucunda N sayıda ADD katsayısı elde edilmiş olur [24].

$$y_{alçak}[n] = \sum_{k=-\infty}^{\infty} x[k].g[2n-k] \tag{1.1}$$

$$y_{yüksek}[n] = \sum_{k=-\infty}^{\infty} x[k].h[2n-k] \tag{1.2}$$

Bu iki işlemin ardından girdi verisi yüksek ve alçak frekans bileşenlerine ayrılmış olur. Örnek olarak, girdi verisi, 1024 örnek uzunluğunda ve 1000 Hz'lik bir işaretten oluşuyorsa, ADD'nin ilk seviyesinde 512 örnek uzunluğunda 0-500 Hz ve 512 uzunluğunda 500-1000 Hz frekans aralıklarından ibaret iki işaret elde edilir.

Filtreleme işlemi sonucunda yüksek geçiren filtreden elde edilen verilere ayrıntı (details) verileri, alçak geçiren filtreden elde edilen verilere ise yaklaşım (approximation) verileri denmektedir [25].



Şekil 1.1 ADD'nin bir seviye için blok diyagramı [26]

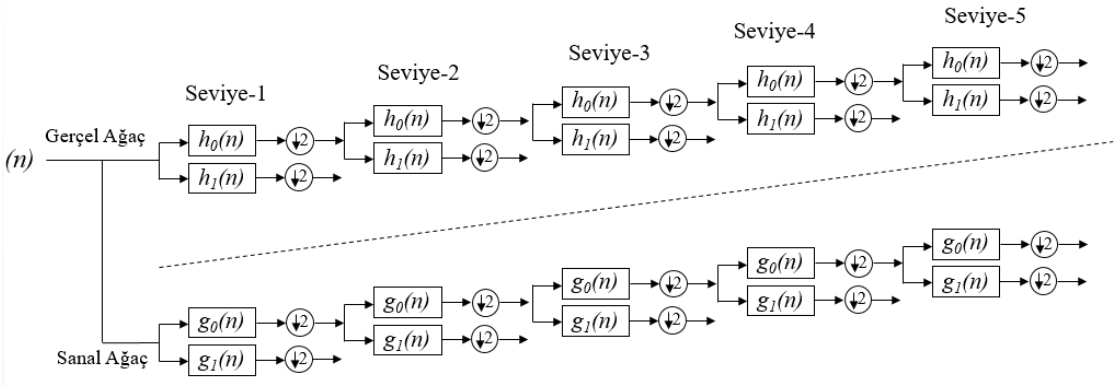
İşlem karmaşıklıkları incelendiğinde *N* adet veri için, hızlı Fourier dönüşümü $O(n\log n)$ işlem karmaşıklığına sahipken ADD $O(n)$ işlem karmaşıklığına sahiptir.

1.4.3 Dalgacık Dönüşümü ile Fourier Dönüşümü Arasındaki Farklar

Hem Dalgacık dönüşümü hem de KZFD işareti frekans bantlarına ayırarak işlem yapar. Dalgacık dönüşümünün Fourier'e karşı en önemli özelliği fonksiyonunun sonlu olmasıdır [28,29]. Dalgacık dönüşümünün diğer bir üstünlüğü ise Dalgacık pencerelerinin büyüklüğünün değişken olmasıdır [28,29].

ÇAKDD[30], ADD'nin yetersiz kaldığı işaretlerdeki faz kayması problemini çözmek amacıyla tasarlanmıştır. Temelde ÇAKDD içersinde iki tane gerçel ADD bulundurmaktadır. İlk ADD dönüşümün gerçel kısmını, ikinci ADD ise dönüşümün sanal kısmını oluşturmaktadır. Analiz ve sentez kısımlarında ADD'de olduğu gibi birbirini takip eden gerçel süzgeç bankları ard arda uygulanmaktadır. Her iki ADD'de birbirinden farklı süzgeç katsayıları kullanılmaktadır. Hem analiz hem de sentez kısımlarında kullanılan bu süzgeçler birbirleriyle ilişkilendirilmişlerdir ve dönüşüm bütün olarak ele alındığında yaklaşık olarak analitiktir.

$h_0(n)$ ve $h_1(n)$, dönüşümün gerçel-analiz kısmında kullanılan alçak-geçiren/yüksek-geçiren süzgeç çiftini; $g_0(n)$ ve $g_1(n)$ ise aynı şekilde dönüşümün sanal-analiz kısmında kullanılan alçakgeçiren/ yüksek-geçiren süzgeç çiftini temsil eder. Her iki süzgeç çifti de işarete uygulandıktan sonra aynı ADD'de olduğu gibi örnek sayısı yarıya indirilmektedir. Dönüşümün her bir aşamasında oluşan Dalgacık fonksiyonları her bir gerçel ADD için $\Psi_h(t)$ ve $\Psi_g(t)$ şeklinde gösterilebilir. Dönüşümün mükemmel geri çatılım özelliğini gösterebilmesi için süzgeç çiftleri, $\Psi(t)$ tüm dönüşümün karmaşık Dalgacık fonksiyonu olmak üzere, $\Psi(t):=\Psi_h(t)+j\Psi_g(t)$ koşulunu sağlamalıdır. Buna ek olarak Dalgacık fonksiyonlarından $\Psi_g(t)$, $\Psi_h(t)$ fonksiyonunun yaklaşık olarak Hilbert dönüşümü olmalıdır ve bu; $\Psi_g(t)\approx H[\Psi_h(t)]$ şeklinde gösterilir. Şekil 1.2'de beş seviye için ÇAKDD gösterilmektedir.



Şekil 1.2 Beş seviye için Çift-Ağaç Dalgacık Dönüşümünün Genel Görünümü

BÖLÜM 2

SONLU KELİME UZUNLUKLU ARİTMETİK

Tez kapsamında yapılan çalışmalarda aritmetik işlemlerin kullanılması gerekmektedir. Bu bölümde, ÇAKDD'nün gerçeklenmesi sırasında kullanılacak aritmetik işlemlerle ile ilgili çalışmalar anlatılmış ve hangi yöntemin ne tür durumlarda kullanılmasının verimli olacağı tesbit edilmiştir. Daha sonrasında ise bu yöntemler ile yapılan gerçeklemelere ait testlerin sonuçları karşılaştırılmıştır.

2.1 Sabit Noktalı Sayılar

Sabit noktalı sayılar günlük hayatımız b,c şeklinde gösterile

Günlük hayatta kullandığımız onluk sayı sistemi, sabit noktalı sayılara verilebilecek en basit örnektir. b,c şeklinde tanımlanır. "b" noktadan önceki basamak sayısını, "c" ise noktadan sonraki basamak sayısını ifade eder.

2.1.1 Sayıların Gösterimi

İki bitlik N adet rakam, 2^N olası duruma sahiptir. Bu durum temel sayma kuramında görülebilir. Şöyle ki; birinci bite ait iki olasılık vardır, 1 ve 0. İkinci duruma da ait iki olasılık vardır, N. bite de ait iki olasılık vardır. Sonuç olarak N ile

$$2 \times 2 \times 2 \cdots \times 2 = 2^N$$

gibi bir olasılık ya da durum elde edilir. Bu durumları kullanarak akla ve mantığa uygun herhangi bir şeyi ifade edebiliriz. İkili düzendeki bir kelimenin özünde hiçbir anlam

bulunmadığı halde pozitif tam sayı olarak düşünmek insanlara cazip gelmektedir. Hâlbuki N-bit bir kelimenin anlamı tamamen nasıl yorumlandığı ile alakalıdır.

2.1.2 İşaretsiz Tam Sayı Gösterimi

Doğal ikili gösterimi, her ikili kelimeyi pozitif bir sayı olarak yorumlar. Örnek olarak 8-bit bir sayıyı ele alırsak;

$b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0$

tamsayı olarak;

$$x = b_7 2^7 + b_6 2^6 + \cdots b_1 2 + b_0 = \sum_{i=0}^{7} (2^i b_i) \quad (2.1)$$

Bu yolla, *n*-bit'lik ikili bir kelime 0 ve 2^N-1 arası bir sayıya tekabül eder. Diğer bir deyişle 0 ile 2^N-1 arasındaki sayılar N-bit'lik ikili sayılar şeklinde gösterilebilirler. İkili sayıların bu şekildeki yorumuna işaretsiz tamsayı gösterimi denir.

Bütün sayılar pozitif tamsayı olduğu için yapılacak olan işlemlerin de sonuçları pozitif tamsayı olacaktır. Bunun sonucunda da basit bir şekilde ikili toplama ve çarpma işlemleri yapılabilmektedir.

Ancak yapılan işlemlerde iki tane N-bit sayının toplamı genelde N+1 bit'lik bir sonuç vermektedir. Böyle bir durumda taşma meydana geldi denir. Genel olarak iki tane N-bit sayının çarpımının sonucu 2N-bit bir kelimedir. İki sayıyı birbirleri ile çarptığımızda, işlemin sonucunda oluşan bit sayısı belirsiz bir şekilde artmaktadır. Bu da Sİİ algoritmalarının donanım üzerinde gerçeklenmesi sırasında istenmeyen bir durumdur.

Başka bir problem de işaretsiz tamsayı gösteriminde yalnızca pozitif tamsayıların gösterilmesidir. Negatif tamsayıların gösterimi için ilave bir yoruma ihtiyaç vardır ve 2'ye tümleyen sayı sistemi ve buna bağlı olarak aritmetiğine giriş yapmak gerekir.

2.1.3 2'ye Tümleyen Tamsayı Sistemi

Doğal ikili gösteriminde, N-bit bir kelime ile 0'dan 2N-1'e kadar olan pozitif tamsayılar gösterilebilmektedir. Ancak, negatif sayıları temsil edebilmek için 2'ye tümleyen sayı

sistemi kullanılır. 2'ye tümleyen sayı sisteminde N-bit bir kelime 'den kadar olan tam sayılar temsil edilebilmektedir. Örnek olarak 8-bit bir sayıyı ele alırsak;

$b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0$

tamsayı olarak;

$$x = -(b_7 2^7) + b_6 2^6 + \cdots + b_1 2 + b_0 = -b_7 2^7 + \sum_{i=0}^{6} (2^i b_i) \tag{2.2}$$

2'ye tümleyen tamsayı sisteminde x'in alabileceği değerler -128 ile 127 arasındadır. Birkaç örnek vermek gerekirse;

Çizelge 2.1 İkilik ve Onluk Taban Örnekleri

İkili	Onlu
00000000	0
00000001	1
01000000	64
01111111	127
10000000	-128
10000001	-127
11000000	-64
11111111	-1

x'in pozitif veya negatif değerleri için, -x'in değerini bulabilmek için şöyle bir örnek verebiliriz. x'in 64 (01000000_2) olduğunu kabul edelim. Öncelikle yapmamız gereken bütün bitlerin tersini (10111111) almaktır. Ardından elde edilen değere 1 (10111111_2+1) eklemektir. Yapılan işlemin sonunda -64'ün 2'ye tümleyen tamsayı sisteminde 11000000_2 şeklinde ifade edildiği görülür. Burada dikkat edilmesi gereken başka bir husus da en büyük basamağın (MSB) işaret biti olarak kullanıldığıdır. 0 pozitifliği, 1 ise negatifliği ifade eder.

2'ye tümleyen sayı sisteminde bulunan işaret biti sayesinde çıkarma işlemleri, $x - y = x + (-y)$ şeklinde toplama işlemi ile yapılmaktadır.

2.1.4 Kesirli Gösterim

İkiye tümleyen tamsayılarla toplama ve çarpma işlemlerinin ikisini de yapabilmemize rağmen tamsayılar Sİİ algoritmalarını başarabilmek için uygun değildir. Örneğin iki tane

8-bit sayıyı birbirleri ile çarptığımızı düşünelim elde edilecek sonucu tutabilmek için 16-bit'e ihtiyacımız vardır. Problemi tam olarak çözmek mümkündür ancak karmaşıktır. Problemi basit bir şekilde çözmek için tamsayılar yerine [-1,1] aralığındaki sayılar kullanılabilir. Çünkü [-1,1] aralığındaki sayıların çarpımı gene aynı aralıktadır.

İkiye tümleyen kesirli sayı gösteriminde, N-bit'lik bir kelime $\frac{(-2)^{-(N-1)}}{2^{N-1}} = 1$'den $\frac{2^{-(N-1)}}{2^{N-1}} = 1 - 2^{N-1}$ aralığını temsil edebilir. Örnek olarak *8-bit* bir kelimeyi ele alırsak;

$b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0$

Kesirli sayı olarak;

$$x = \frac{-(b_7 2^7) + b_6 2^6 + \cdots + b_1 2 + b_0}{2^7} = -(b_7) + \sum_{i=0}^{6} (2^{i-7} b_i) \in [-1, 1 - 2^{-7}] \quad (2.3)$$

Bu gösterime aynı zamanda ***Q-format*** da denmektedir. MSB'nin hemen sağındaki bitten itibaren ikili bir sayı oluşturduğumuzu düşünelim. *N*-bit'lik bir kelimemiz varsa ve MSB'yi de işaret biti olarak kullanırsak geriye kesirli kısmı gösterebilecek *N-1* bit kalmaktadır. Bu şekilde yazılmış bir sayı için *Q-(N-1)* formatında gösterilmiştir denir. Ancak bu gösterim, bit sayısının belli olduğu durumlarda kullanılabilir. Eğer belli değilse *b,c* şeklinde gösterilir. Örnek vermek gerekirse 11,001 sayısı *2,3* şeklinde bir düzene sahiptir denir. 16 bitlik bir yapı söz konusu ise 12,4 (Q-4), 1,14 (Q-14) vs. şeklinde gösterilebilir.

Sabit noktalı sayıların bellekte temsili, b,c noktalı sayısı için, tamsayı kısmı aynen alınır, kesir kısmı ise; K=kesir*2^c şeklinde hesaplanır. Örnek vermek gerekirse, 4.12 formatında 3,68 sayısı K=0,68*4096=2785=0xAE1, böylece sayı 0x3AE1 olacaktır.

2.1.5 İkiye Tümleyen Aritmetiği

İkiye tümleyen formatının kolaylığı, negatif sayıları gösterebilmesi ve aynı algoritma ile çıkarma işlemini toplama işleme yaparak hesaplamasından gelir. Genelde Sİİ işlemcilerin kullandığı format Q-15 olduğu için Sİİ algoritmalarının bu işlemciler üzerinde kolay uygulanabilmesi için Q-15 formatı kullanılır. Bu sebepten dolayı aşağıda kullanılan örnekler de Q-15 formatındadır.

2.1.5.1 Toplama ve Çıkarma

İki ikili sayının toplamının hesaplanması ondalı sayı düzeni ile aynı şekilde yapılır. 0+0=0, 0+1=1, 1+0=1 ve 1+1=10 ilişkisi kullanılarak ikili sayı düzenindeki sayılarla toplama işleme kolayca yapılır.

Ancak ikili sayıları toplarken dikkatli olmak gerekmektedir. Çünkü bütün sayıların Q-15 formatı aralığında olması gerekir. Eğer iki Q-15 formatındaki sayının toplamı bu aralıkta değilse sonucun Q-15 formatında gösterimi mümkün değildir. Böyle bir durumun gerçekleşmesi halinde taşma olmuştur denir. Dikkatli ele alınmadığı takdirde bu taşma yanlış sonuca sebebiyet verir. Sİİ algoritmalarının uygulanması sırasında taşmaların gerçekleşmesinin önlenmesi gerekir. Bunu engellemenin yollarından biri küçük sayılarla çalışmaktır. Bu sayede yapılan işlemlerin sonuçları [-1,1) aralığında olacaktır. Kullanılacak sayının daha küçük bir sayı olarak ölçeklenmesi ile işleme sokulması istenen durumu sağlayacaktır. Burada önemli olan hangi miktarda ölçeklenme yapılacağıdır. Ölçekleme, yalnızca taşmayı önleyecek kadar yapılmamalı aynı zamanda sonucunda hata miktarına dikkat edilmelidir. Ölçeklenme gereğinden fazla yapılırsa sonuçlar hatalı olmaya başlayacaktır ve alt problem çözülürken ana problem çözülemeyecektir.

Taşma (veya alttaşma) probleminin başka bir çözümü de doyurmadır (saturation). Elde edilen sonuç kullanılan veri büyüklüğü ile gösterilebilen aralıktaki en yakın sayıya doyurulur.

2.1.5.2 Çarpma

İkiye tümleyen sayı sisteminde çarpma işlemi biraz daha zordur. Bunun sebebi işaret bitidir. Onlu düzendekine benzer şekilde Q-N formatındaki iki sayının sonucu Q-2N'dir. Çarpılan sayıların özelliğinden dolayı elde edilen sonucun ikili sayının sol tarafında bir ya da iki işaret biti bulunur. Bunların birincisine işaret biti, ikincisi var ise genişletilmiş işaret biti denir. İkinci işaret bitinden kurtulmak için bir bitlik öteleme yapılır.

Şekil 2.1'de iki kesirli sayının çarpımı ile ilgili bir örnek bulunmaktadır.

```
      0.110   0.75     Q-3
   x  1.110   -0.25    Q-3
---------------------------------------
        0000
       0 110
      01 10
     101 0
+
---------------------------------------
     1110100    -0.1875 Q-6
```

Şekil 2.1 Q-Format Çarpma Örneği

Yukarıdaki tüm kısmi çarpımlar toplama için Q-6 formatında yazılmış ve hesaplanmıştır. 4. kısmi çarpıma dikkatli bakmak gerekmektedir. Çünkü 0,110*1,000 işleminde 1,000, -1'i göstermektedir. Buradan, -0,110=1,01000 (Q-6 formatında) bu da 0,11000 sayısının 2'ye tümleyenidir. Bu örnekte de görüldüğü üzere kısmi çarpımların toplama işlemine girmeden önce Q-6 formatında yazılması oldukça önemlidir.

2.1.5.3 Aritmetik Hesaplamaların Sonuçlarının Temsili

Bölüm 2.2.2'deki örnekte de görülebileceği üzere Q-N formatındaki iki sayının sonucu tam olarak Q-2N formatı ile temsil edilebilir. Ancak gerçek hayattaki uygulamalarda her seferinde Q-2N formatına çıkmak pek de mümkün olmamaktadır. Bunun için kullanılan bir takım yöntemler vardır.

2.1.5.3.1 Yuvarlama

Yapılan hesaplamanın sonucunun kullanılacak formata uygun bir sayıya yuvarlanarak gösterimidir. Örnek vermek gerekirse Q-6 formatında yazılmış 0,000110 sayısının Q-4 formatına dönüşümü sırasında sayı alta ya da üste yuvarlanarak yazılır. Bu örnekte sayı üste yakın olduğu için üste yuvarlanır ve Q-4 formatında 0,001 olarak yazılır.

2.1.5.3.2 Kesme

Yapılan hesaplamanın sonucunun kullanılacak olan formata uygun olarak sığdırılırken taşan kısmın atılması şeklinde uygulanan yöntemdir. Örnek vermek gerekirse, Q-6 formatında 0,011001 sayısı Q-4 formatına çevrilirken sondaki iki bit atılır ve sayı 0,0110 şeklinde yazılır.

2.2 Kayan-Noktalı Sayılar

Kayan-noktalı sayılar gerçel sayıların bilgisayar ortamındaki gösterim şekillerinden biridir [32]. Yukarıda da bahsedildiği üzere gerçek dünyada sayılar sonsuza kadar gidebilirken, bilgisayar ortamındaki kısıtlardan dolayı sonsuza kadar giden bir gösterimin olması mümkün değildir. Sonsuza giden değerler yaklaşık olarak gösterilir. En az kayıpla sınırlı bir alanda gösterim sağlayan en iyi yöntem Kayan-Noktalı sayı sistemidir. Kayan-noktalı sayılar sistemi, bir sayı ile 10'un herhangi bir kuvvetinin çarpımı şeklinde sıklıkla kullanılan bilimsel gösterime oldukça benzeyen bir notasyona sahiptir ve en sık kullanılan IEEE 754 standardına göre şekillendirilmiştir [32].

2.2.1 Kayan-Nokta Gösterimi

Bilimsel gösterimde herhangi bir sayı, virgülden önce bir basamak olacak şekilde 10'un üsleriyle çarpım halinde gösterilir. Örneğin 0,00741 sayısı $7{,}41 \times 10^{-3}$ şeklinde gösterilir. Bu formatta 10'un kuvvetinin arttırılması durumunda virgül bir basamak sola, azaltılması durumunda ise virgül bir basamak sağa kaydırılır. Bu işleme normalizasyon denir. Kayan-Nokta sisteminin çıkış noktası bu mantığa dayanır.

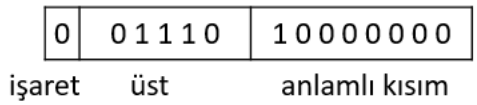
Kayan-Noktalı sayılar, ikilik düzendeki sayıların bilimsel gösterimle gösterilmesidir. Kayan-noktalı sayılar işaret, anlamlı kısım ve üst (2'nin üssü şeklinde) olmak üzere üç kısımdan oluşur.

$$M \times B^E$$

Burada M kayan-noktalı sayının mantis'i; B tabanı, E ise üssüdür. Kayan-noktalı sayının gösteriminde anlamlı kısmın bitleri fazla olursa sayının duyarlılığı, üst bitleri fazla olursa gösterilebilecek sayı aralığı artar.

Bu modelde $0{,}25_{10}$ gibi bir sayı gösterilmek istendiğinde $0.25_{10} = 1.0 \times 2^{-2}$ olacağından üst için ayrı bir işaret biti kullanmak gerekmektedir. Bunun yerine saptırılmış üst yöntemi ismi verilen bir yöntem kullanmak daha verimli olmaktadır. Bu yöntemde, gerçek üst değerine sabit sapma değeri eklenerek saptırılmış üst elde edilir. Örneğin bu modelde saptırma değeri olarak 16 seçilmiş olsun. Bu durumda üst değeri 16'dan büyük olan değerler pozitif üstleri, 16'dan küçük olan değerler için negatif

üstleri gösterir. 0.25_{10} sayısı Şekil 2.2'de de görülebileceği üzere işaret biti 0, anlamlı kısım 10000000_2, negatif üst için saptırılmış değer -2+16=14 yani 01110_2 olacaktır.



Şekil 2.2 Kayan-Nokta Bileşenleri Örneği

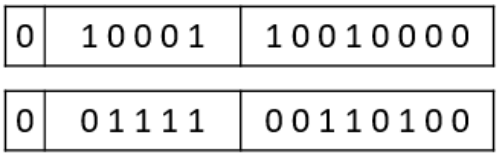
2.2.2 Kayan Nokta Aritmetiği

Kayan-nokta gösterimi ile çalışan sistemlerde yapılacak işlemler için özel yöntemlerin kullanılması gerekmektedir. Bu yöntemler aşağıda örnekleri ile birlikte anlatılmıştır.

2.2.2.1 Toplama

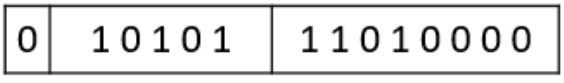
Kayan-noktalı sayılar toplanırken ilk olarak toplanacak olan sayılardan küçük olan sayının anlamlı kısmı, üstler eşitleninceye kadar sağa kaydırılır. Ardından anlamlı kısımlar toplanır ve gerekiyorsa sayı olağanlaştırılır. Son olarak ise anlamlı kısım, gösterimde belirlenmiş olan bit sayısına yuvarlanır. Sonuç, olağan hale gelinceye kadar sayının olağanlaştırılması ve gösterimde belirlenmiş olan bit sayısına yuvarlama işlemleri tekrarlanır.

Şekil 2.6'da işaret biti hariç 13 bitle (5 bit üst ve 8 bit anlamlı kısım) gösterilen ve üst saptırma değeri 15 olan sayılar toplanmak istenirse;



Şekil 2.3 Kayan-Nokta Toplama Örneği

Şekil 2.3'teki sayıların değerleri 11,0010000 ve 0,100110100'dir. İki sayını toplamı ise 11,0010000 + 0,100110100 = 11,10111010 olarak elde edilir. Anlamlı kısmın gösteriminde bit sayısı kısıtlaması olması sebebiyle sayıyı olağanlaştırıldığında sağdaki bitler kaybedilir ve toplam Şekil 2.4'te olduğu gibi gösterilir.



Şekil 2.4 Kayan-Nokta Toplama Örneğinin Sonucu

2.2.2.2 Çarpma

Kayan-noktalı sayılarda çarpma işlemi yapılırken ilk olaraki ki sayının üst değerleri toplanarak sonucun üst değeri elde edilir. (Ancak bu üst değerleri saptırılmış olduğundan, sonucun üst değeri iki kez saptırılmış olarak elde edilir. Bu nedenle sonucun üst değerinden saptırma değeri çıkarılarak gerçek üst değeri bulunur.) İkinci olarak sayının anlamlı kısımları çarpılarak sonucun anlamlı kısmı hesaplandıktan sonra sonuç olağanlaştırılır. Son olara ise bit sayısına göre yuvarlama yapılır.

Örnek olarak Şekil 2.5'teki 1 biti işaret, 8 biti üst ve 4 biti anlamlı kısmı gösteren ve üst için saptırma değeri 127 olan olağanlaştırlımış iki sayının çarpımı incelenirse,

```
  0  10000100  0100
x 1  00111100  1100
```

Şekil 2.5 Kayan-Nokta Çarpma Örneği

Sonuç işareti 1 olacaktır. Anlamlı kısımlar çarpıldığında (olağanlaştırlımış biçimde olduğundan dolayı gösterilmeyen bit de eklenir); Şekil 2.6'daki gibi bir sonuç elde edilir.

```
      1.0100
x     1.1100
     00000
    00000
   10100
  10100
+ 10100
 1000110000
10.0011000
```

Şekil 2.6 Kayan-Nokta Çarpma: 2. Adım

Üstler hesaplanırken ise Şekil 2.7'de görülen 3. adım işlem yapılır.

```
  10000100
+ 00111100
----------
  11000000
```

Şekil 2.7 Kayan-Nokta Çarpma İşlemi: 3. Adım

Elde edilen sonuç iki kez kaydırılmış üst değerini vermesi sebebiyle, kaydırma değeri bir kez çıkarılır ve 11000000 - 01111111 = 01000001 olarak sonuç bulunmuş olur.

Sonuç olağanlaştırıldıktan ve anlamlı bölüm dört bite yuvarlandığı takdirde çarpım 1 01000010 0001 şeklinde elde edilir.

2.3 Sabit-Nokta ve Kayan-Nokta Karşılaştırılması

Yukarıda anlatılanlar göz önüne alınarak iki yöntem ele alındığında iki yöntemin de bir takım avantajları ve dezavantajları bulunmaktadır. Bu noktada sorulması gereken soru hangi yöntemin nerede kullanılması gerektiğidir.

Her iki yöntemin de kullanıldığı hesaplamalarda en büyük problem sonlu kelime uzunluğudur. Sonlu kelime uzunluğu yapılan hesaplamanın sonucunun bir kısmının kaybına sebep olur ve hesaplamalarda artarak büyüyen bir gürültüye neden olur [33]. Hata ve dinamik bölge bit sayısı ile ilgilidir. Sabit-noktada dinamik bölge kelime boyutu ile doğrusal artarken, kayan-noktada üstel olarak artar. Sabit noktalı işlemlerde yapılan hatalar kayan-noktaya göre daha fazladır. Ancak aynı bit sayısı için mutlak hatalar, kayan-noktada daha fazladır.

Herhangi bir sistemin kayan-noktalı sistem ile geliştirilmesi sabit-noktaya göre daha hızlı olmaktadır. Kayan-noktalı sistemler geliştirilirken taşma, alt-taşma vs. gibi durumlarla uğraşılma zorluğu yoktur. Sabit-noktada ise bunların hiç bir zaman göz ardı edilmemesi gerekmektedir.

Ürün maliyeti ele alındığında ise donanımsal olarak kayan-noktalı sistemler çok daha maliyetli olmaktadır. Sabit-noktalı sistemlerde –özellikle Sİİ sistemlerde- 16-bitlik bir yapı kullanılırken kayan-noktada ise bütün veri yollarının kaydedicilerin 32-bit olması gerekmektedir. Buna ek olarak mimaride kayan-nokta daha karışıktır. Ekonomide, çağımızın en önemli kavramı olan rekabet göz önüne alındığında karlılığın arttırılmasında fiyat artışından daha çok maliyetlerin aşağıya çekilmesi daha büyük bir

öneme sahiptir. Ekonomik kaygılar göz önüne alındığında ise gerçek zamanlı uygulamalarda sabit-nokta bariz bir üstünlüğe sahiptir. Donanımsal büyüklükler karşılaştırıldığında ise sabit-noktalı sistemler mimari olarak daha basit olduğu için daha az yer kaplar.

BÖLÜM 3

ÇAKDD ALGORİTMASININ FARKLI PLATFORMLARDA GERÇEKLENMESİ

Çift-ağaç kompleks dalgacık dönüşümü (ÇAKDD) yöntemi, gerçek zamanlı bir sistemde uygulanmak amacı ile çeşitli ortamlarda gerçeklenmiştir. Alt başlıklarda kullanılan yöntemler ve ortamlar detayları ile anlatılmıştır.

3.1 Matlab

MATLAB (matrix laboratory) sayısal hesaplama ve dördüncü nesil programlama dilidir. MathWorks firması tarafından geliştirilmektedir. MATLAB, matris işlemleri, fonksiyon çizimi, algoritma uygulaması, kullanıcı arayüzü oluşturulması ve diğer dillerle yazılmış programlar ile etkileşim oluşturulması gibi görevleri yapabilen kullanıcı dostu bir programdır.

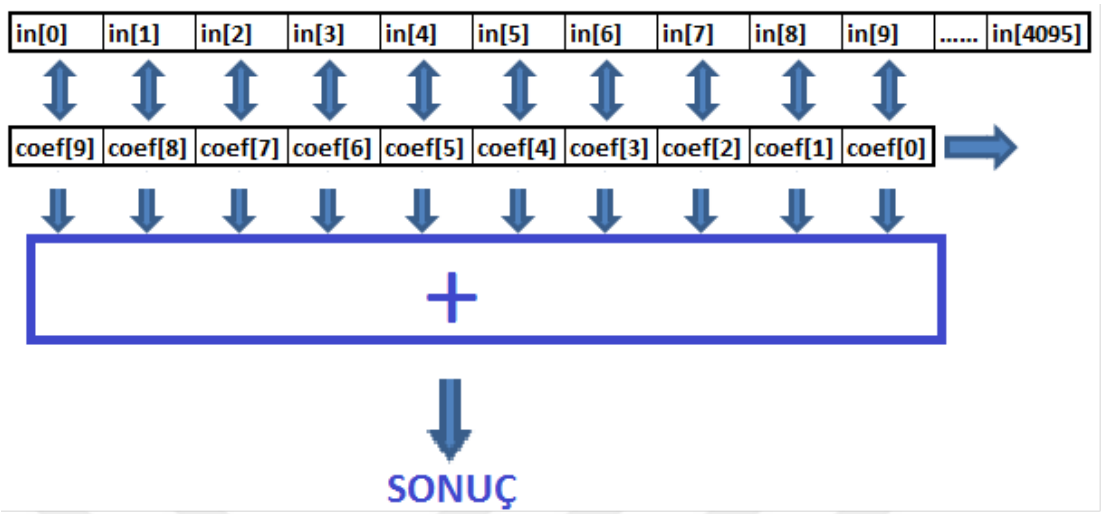
MATLAB, öncelikli olarak sayısal işlemlere yönelik olarak geliştirilmesi rağmen, isteğe bağlı olarak sembolik hesaplama yapabilen MuPAD sembolik motorunu da kullanır. Dinamik ve gömülü sistemler için Simulink, grafiksel çoklu alan simülasyonu ve model tabanlı tasarım gibi görevleri yerine getirmek için ek paketler de kullanılabilir.

MATLAB üzerinde yapılan çalışmalarda MATLAB'in altında hazır olarak bulunan bir takım filtre fonksiyonların da kullanımı ile yöntem gerçeklenmiştir. Yapılan testler sonucunda 100Khz frekansa kadar gerçek zamanlı olarak yöntemin çalıştırılabileceği görülmüştür.

3.2 C Dili ile Gerçeklenmesi

ÇAKDD yönteminin C programlama dili ile gerçeklemesi sabit-nokta formatı kullanılarak yapılmıştır. Yöntemin uygulanırken öncelikle veriler alınarak diziye kaydedildikten

sonra dizi üzerinde filtre katsayıları dizi Şekil 3.1'de görüldüğü gibi kaydırılarak FIR filtre yöntemi uygulanmıştır.



Şekil 3.1 ÇAKKD Yönteminin C Programlama Dili Gerçeklenmesinde Kullanılan Yöntem

Test sırasında karmaşık girdi verisi programa off-line olarak beslenerek sonuçlar doğru bir şekilde elde edilmiştir. Yapılan testin sonucunda 1,6GHz 4 çekirdekli işlemci, 4GB belleğe sahip bir kişisel bilgisayarda yaklaşık 6,4M örnek/sn. girdi hızına ulaşılabileceği görülmüştür. Ancak çevre birimlerle olan etkileşimden dolayı gerçek zamanlı hız yaklaşık 0,2M örnek/sn. hıza kadar düşmektedir.

3.3 PIC Platformunda Gerçeklenmesi

PIC, *Peripheral Interface Controller* kelimesinin başharflerinin kullanılması ile oluşturulmuş bir kısaltma olup Microchip firmasının ürettiği mikrodenetleyicilere verilen addır [34]. PIC, Harvard mimarisine ve RISC işlemcisine sahiptir. Bu özelliklerinden dolayı performans açısından oldukça verimlidir.

ÇAKDD'nün PIC üzerinde gerçeklenmesi, bilgisayar üzerinde çalıştırılan, C ile yazılmış, bir önceki bölümde bahsi geçen programa ait algoritma ile mümkün olmamaktadır. Bunun başlıca sebebi PIC, kısıtlı kaynağa –özellikle bellek açısından- sahip bir cihazdır. Bu durumun üstesinden gelebilmek için yalnızca, işlenmek üzere filtrelere gelen son on (Filtre, 10-tap olduğu için) veri hafızada tutulmuştur. Filtrelere her veri gelişinde de bir sonuç üretilmiştir. Bir önceki bölümde anlatılan yöntemde ise bütün veriler alındıktan sonra işlemler yapılmıştır. Kullanılan yöntem sayesinde yaklaşık *[(Kullanılan Filtre*

*Sayısı)*10+(Filtre Katsayıları)]*4B+(Programda kullanılan değişkenler vs.)* kadar bir hafıza kullanılmıştır.

Testler, PIC 18F452 model ile hem kayan-noktalı hem de sabit noktalı olarak yapılmış olup sabit-nokta kullanımı koyan noktaya göre yaklaşık %35 daha iyi bir sonuç vermiştir. Ancak sabit noktada dahi 10000 karmaşık sayı verisini 19.6 sn.'de işleyebilmiştir. Sonuçlar, gerçek zamanlı olarak bu PIC 18F452 modelinde 500hz'in üzerindeki frekanslar için gerçek zamanlı olarak çalışılamayacağını göstermiştir. Çözüm için daha gelişmiş bir modelin kullanılması gerekmektedir.

3.4 Raspberry

Raspeberry Pi, Rasberry Pi Vakfı tarafından 2009 dan beri geliştirilmesi devam eden bir platformdur. Boyutu kredi kartı büyüklüğünde olup, tek kart üzerinde küçük bir bilgisayar özelliği taşımaktadır. Üzerinde Cortex A7 4 çekirdekli işlemci, Broadcom VideoCore IV grafik işlemcisi, 512 MB RAM, HDMI ve Ethernet gibi arayüzler bulundurmaktadır. Cihaz PIC gibi taşınabilirliği yüksektir. Bununla birlikte üzerine işletim sistemi kurulabilmektedir. İşletim sistemi kurulabilme özelliği Rasberry Pi'in hızlı bir şekilde yazılım geliştirlebilmesine imkân sağlar. Çünkü işletim sisteminin tüm özellikleri ve diğer geliştiricilerin sağladığı kütüphaneler kullanılabilir hale gelmektedir. Bu özellik PIC'de bulunmamaktadır.

Rasberry Pi bir bilgisayar gibi olmasına rağmen, tasarımı nedeniyle çok az güç çekmektedir. Gömülü sistem uygulamalarında kullanıma çok uygundur. Üzerine işletitm sistemi olarak, Noobs, RASPBIAN, Ubuntu Mate, Windows 10 IOT gibi kurulumlar yapılabilmektedir.

Tez kapsamında cihazın üzerinde cihaz ile tam uyumlu, cihazdan maksimum performans alınabilen RASPBIAN işletim sistemi kurulmuştur. Daha önceden geliştirilmiş olan C dilindeki algoritma, Linux tabanlı olan RASPBIAN üzerinde gerekli değişiklikler yapılarak koşturulmuştur.

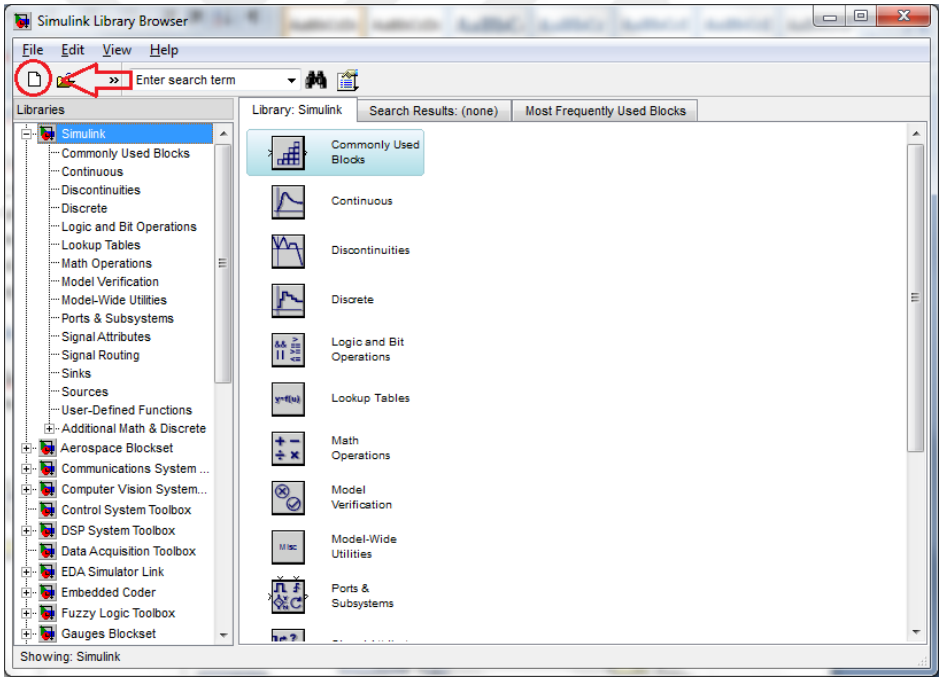
Kolay taşınabilir olması çok önemli bir unsurdur. Ancak hem kolay taşınabilirlik hemde yüksek hesaplama gücü gerektiren uygulamalarda RASPBIAN'ın yetersiz kaldığı görülmüştür.

3.5 System Generator

System Generator, Xilinx firması tarafından geliştirilmiş, Mathwork firmasına ait model-tabanlı tasarım programı Simulink'te FPGA tasarımına olanak sağlayan bir Sİİ aracıdır. System Generator'da tasarım yapabilmek için daha önceden her hangi bir FPGA veya RTL tasarım metodolojisi deneyimine sahip olmaya gerek yoktur. Yapılan tasarımlar Simulink modelleme ortamında tutulur. FPGA programlama dosyasının oluşturulması için gerekli olan tüm geliştirme aşamaları otomatik olarak gerçekleştirilir.

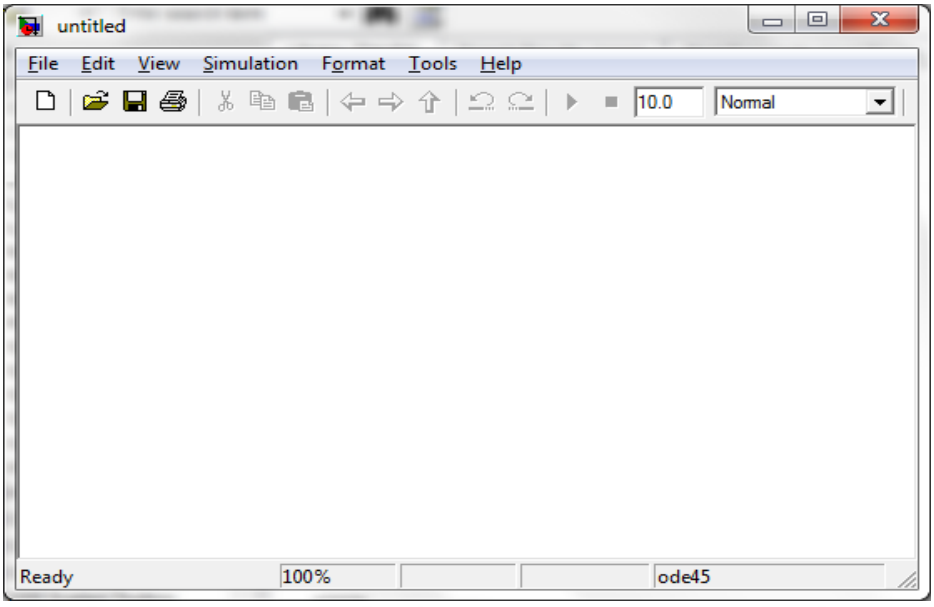
System Generator'ı kullanarak Sİİ modellemesi yapabilmek için Simulink'in altında Xilinx bloğunun kullanılması gerekmektedir. Bu bloğun altında bulunan modüller ile adım adım tasarım tamamlanır.

İlk yapılması gereken Şekil 3.2'de gösterilen "Simulink Kütüphanesi Tarayıcısı" penceresinden "new" butonuna tıklanarak yeni modelin açılmasıdır.



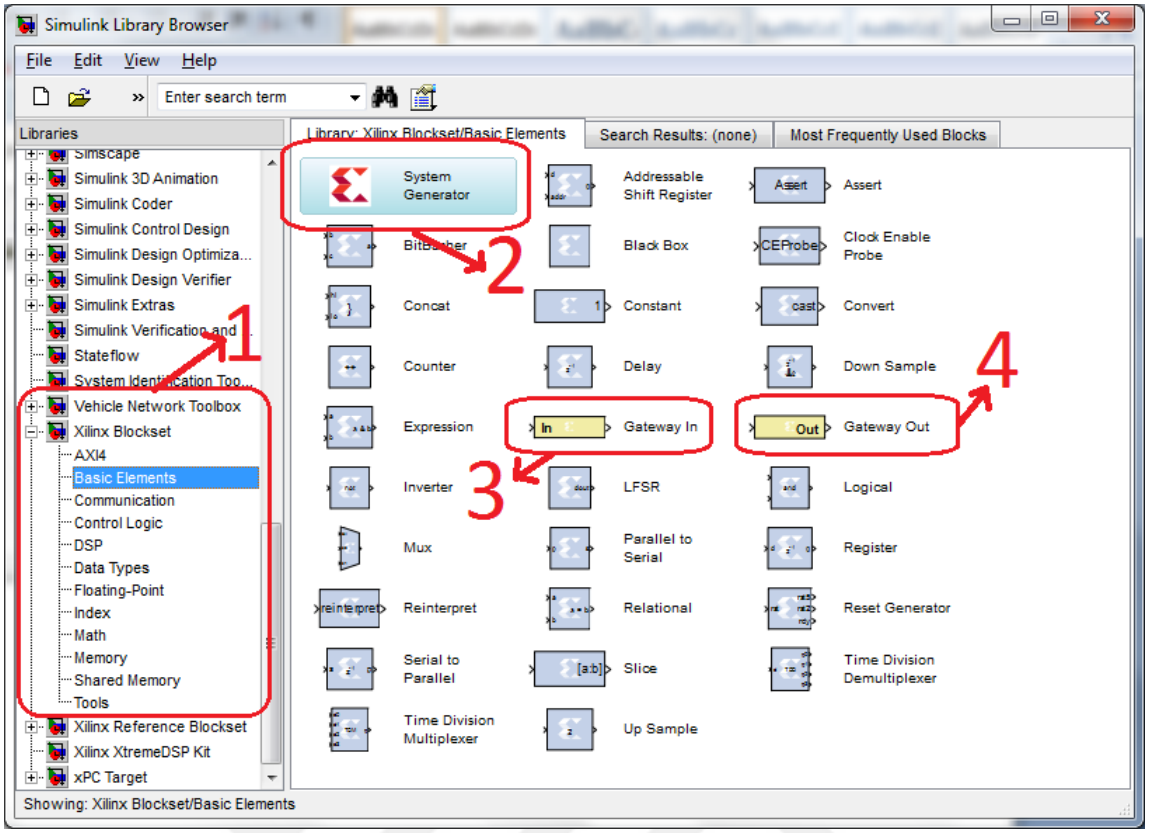
Şekil 3.2 Simulink Kütüphane Tarayıcısı

Yeni bir pencere açılır ve bu pencerede modelleme yapılır. (Şekil 3.3)



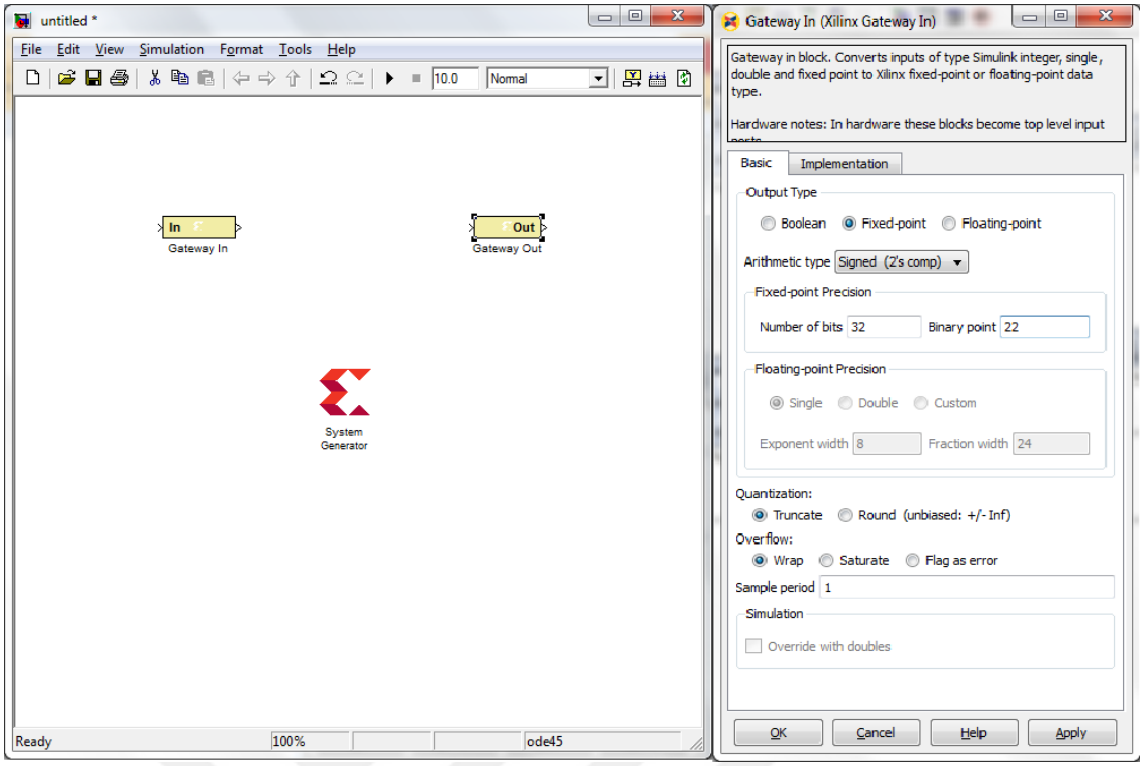
Şekil 3.3 Yeni Model Penceresi

Modele eklenecek modüller Simulink Kütüphanesinde Şekil 3.4'te "1" ile gösterilen bölümden Xilinx Blockset menüsünden seçilir. İlk eklenmesi gereken eleman her modelde bulunması gereken Şekil 3.4'te "2" ile gösterilen System Generator simgesidir. Bu simge modelle ilgili sistem ve simülasyon parametrelerinin kontrolünü sağlayan bir kontrol paneline, net listelemesi için gerekli olan kod üreticisine sahiptir. Simge bir kez eklendikten sonra kod üretiminin ve simülasyonunun nasıl yapılacağı ayarlanabilir. Zorunlu olmasa da hemen her modelde en az bir giriş bir de çıkış vardır. Bu giriş çıkışların Xilinx bloklarına belirli bir düzende bağlanması gerekmektedir. Bunun sağlanabilmesi için modele dışarıdan gelen girdilerin "Gateway In" (Şekil 0.14'te 3 ile gösterilen simge), modelden çıkışların "Gateway Out" (Şekil 3.4'te 4 ile gösterilen simge) simgesinden geçmesi gerekmektedir. "Gateway In" simgesi kullanılarak Simulink'e ait tamsayı, kayan noktalı ve sabit noktalı sayı formatlarını System Generator sabit nokta formatına dönüştürür. Aynı mantıkla çıktılar için de tersi işlem "Gateway Out" ile yapılır.



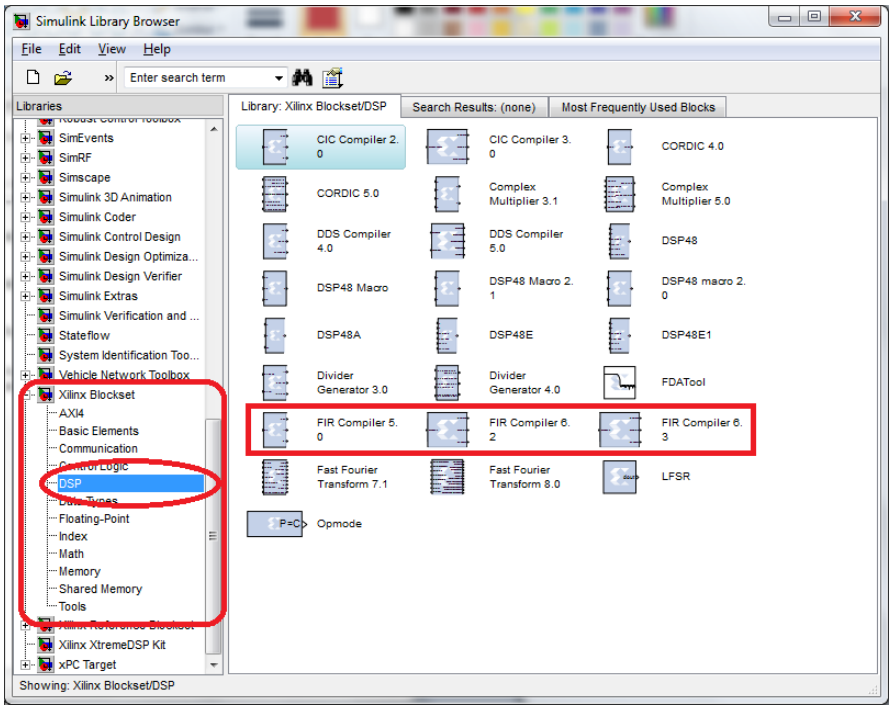
Şekil 3.4 Xilinx Blockset Altında Temel Elemanlar

Yukarıda bahsedilen üç temel simge modele eklendikten sonra giriş ve çıkışlar için bir takım ayarlar yapılmalıdır. Şekil 3.5'te simgeler eklendikten sonraki hali ile model görünmektedir. "Gateway In" simgesine çift tıklanarak açılan menüde (Şekil 3.5'te sağdaki pencere) istenilen tercihler seçilebilmektedir. Çıktı tipleri (Output Type) kısmında mantıksal, sabit nokta (fixed-point) ve kayan nokta (floating-point) tipleri seçilip ardından bit genişliği ve ikili noktanın (binary point) konumu seçilebilir. Sabit nokta için kes (truncate), yuvarlama (rounding) yöntemleri belirlenebilir. Geliştirilen model için girdi verileri 32 bit kelime genişliğine ve 10 bitlik bir tamsayı kısmına sahip olacak şekilde ayarlanmıştır.



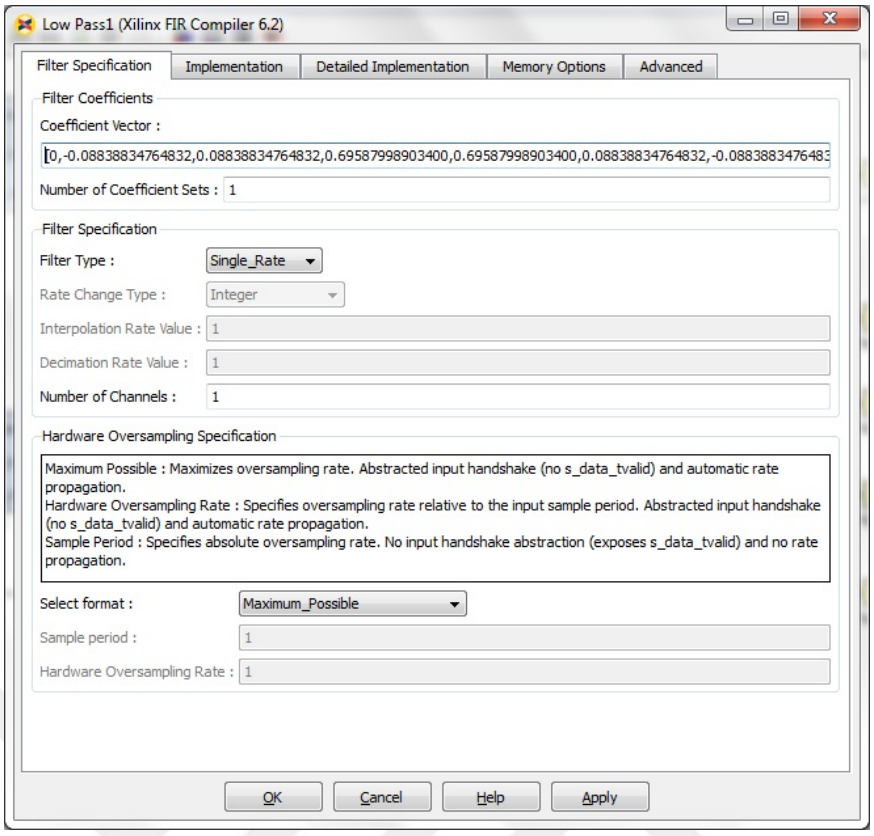
Şekil 3.5 Gateway In Menüsü ve Modelin Simgeler Eklendikten Sonraki Görünümü

Tasarımda kullanılacak en önemli modül ise FIR filtredir. FIR filtre, Xilinx Blockset'in altında Sİİ menüsü seçilerek kütüphaneden seçilebilir. Kütüphanede FIR filtre ile ilgili üç versiyon (Şekil 3.6) bulunmaktadır. Modelde FIR Compiler 6.2 versiyon kullanılmıştır.



Şekil 3.6 Xilinx Blockset'in Altında DSP Menüsü

FIR Compiler 6.2 modele eklendikten sonra modele eklenen simgeye çift tıklanarak açılan pencerede filtre ile ilgili tüm gerekli ayarlar yapılabilir. Açılan pencerede karşılaşılan ilk menüde filtrenin bir takım özellikleri tanımlanır. Bunlardan ilki katsayı vektörüdür. Katsayı vektörü isteğe bağlı olarak elle, dosyadan okutularak veya Matlab ortamından okutularak ilgili alana tanımlanabilir. İkinci olarak filtrenin tipi belirlenir. Burada seçilebilecek üç farklı filtre tipi vardır. Bunlar "Single_Rate", "Interpolation" ve "Decimation" şeklindedir. Single Rate'in seçilmesi durumunda filtreye giren verinin frekansı ile çıkan verinin frekansının aynı olması sağlanır. Interpolation'ın seçilmesi durumundaysa çıkan verinin frekansı giren veriye göre belirlenen oranda fazla olmaktadır. Örnek vermek gerekirse Interpolation seçildikten sonra aktif edilen Interpolation Rate Value parametresi 2 olarak belirlenirse birim zamanda gelen N adet veriye karşılık 2N adet veri filtreden çıkacaktır. Bu olaya "Up Sampling" denmektedir. Decimation'ın seçilmesi durumundaysa çıkan verinin frekansı giren veriye göre belirlenen oranda azalmaktadır olmaktadır. Interpolation seçildikten sonra aktif edilen Decimation Rate Value parametresi 2 olarak belirlenirse birim zamanda gelen N adet veriye karşılık N/2 adet veri filtreden çıkacaktır. Bu olaya ise "Down Sampling" denilmektedir. Up Sampling Down Sampling işlemleri filtre parametreleri içinde belirlenmeden de yapılabilir. Anlatımın devamında bunun nasıl yapıldığı detayları ile gösterilmektedir. Şekil 3.7'de de görülebileceği gibi filtre katsayıları elle girilmiş olup, filtre tipi Single_Rate olarak belirlenmiştir.

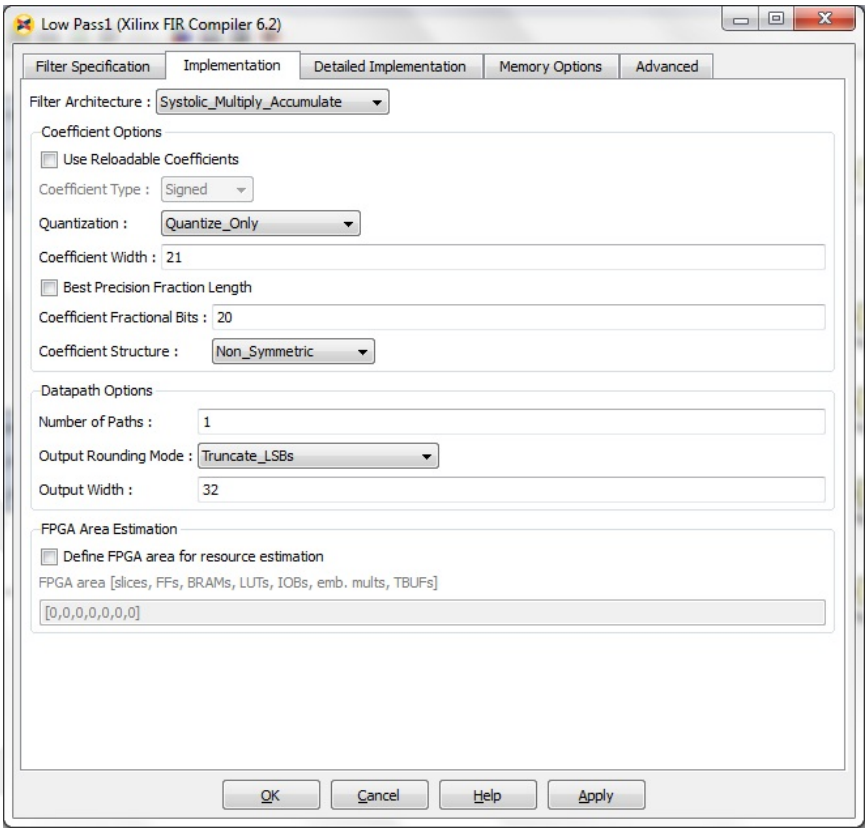


Şekil 3.7 FIR Compiler 6.2'ye ait Filtre Tanımlama Menüsü

Filtrenin geliştirilmesi ile ilgili yapılacak tercihler "Implementation" menüsü (Şekil 3.8) altında bulunmaktadır. Filtre mimarisi "Systolic_Multiply_Accumulate" ve "Transpose_Multiply_Accumulate" olmak üzere iki farklı şekilde yapılabilmektedir. Modelde tercih edilen mimari Systolic_Multiply_Accumulate olmuştur. "Quantization" seçeniğinin altında ise "Integer_Coefficients", "Quantize_Only" ve "Maximize_Dynamic_Range". Integer_Coefficients, filtre katsayılarının tamsayı olduğu durumlarda seçilir. Sabit nokta aritmetiği kullanıldığında Quantize_Only kayan nokta kullanıldığında ise Maximize_Dynamic_Range kullanılır. Tasarımı yapılan bu modelde ise sabit nokta kullanıldığı için Quantize_Only seçilmiştir.

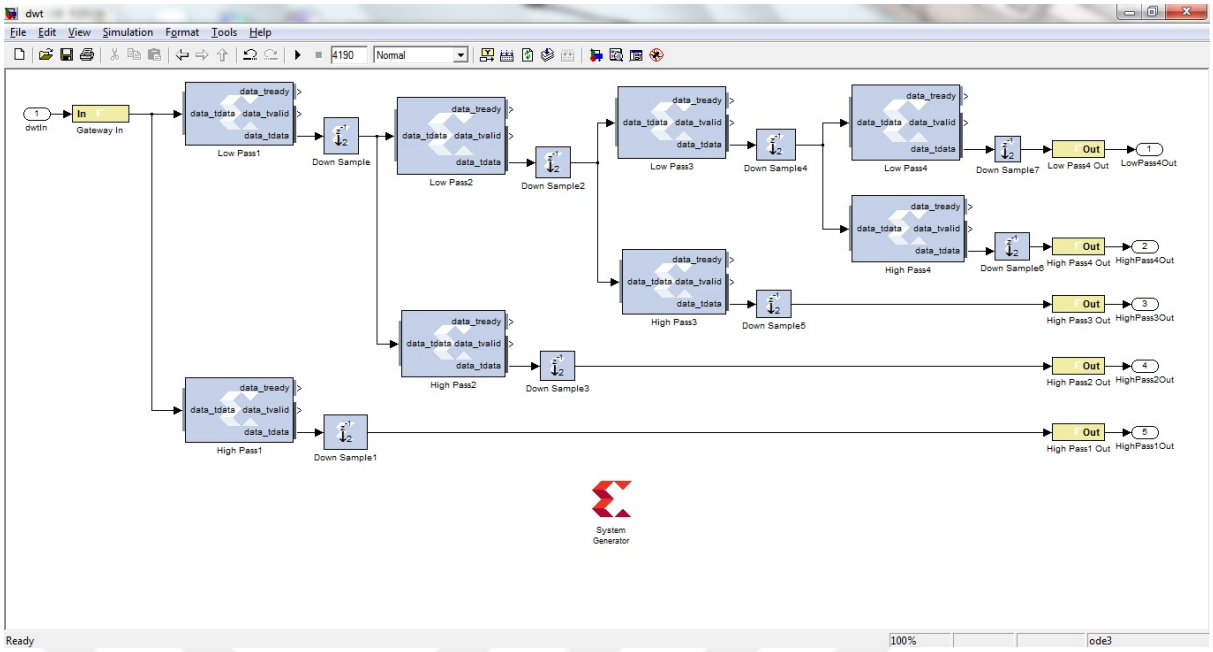
Menünün devamında ise şekil 3.8'de görülebileceği üzere katsayıların ve yapılan hesaplamaların sonuçları ile ilgili bit genişlikleri ve yapıları belirlenmektedir. Katsayıların bit genişliği (Coefficient Width) modelde 21 olarak belirlenmiş olup kesirli kısmı (Coefficient Fractional Bits) 20 bit olarak ayarlanmıştır. Katsayıların yapısı (Coefficient Structure) ise katsayıların simetrik olmayışından dolay "Non_Symmetric" şeklinde belirlenmiştir. Çıktı yuvarlama modu (Output Rounding Mode) ise

“Truncate_LSBs” (en önemsiz bitleri kes) seçeneği tercih edilmiştir. İsteğe bağlı olarak diğer yuvarlama yöntemleri de seçeneklerde bulunmaktadır. Son olarak ise çıktı genişliği (Output Width) 32 olarak belirlenmiştir.



Şekil 3.8 FIR Compiler 6.2’ye Ait Uygulama Menüsü

Yukarıda sıralanmış işlemler yapılarak her hangi bir FIR filtreyi gerçeklemek mümkündür. ÇAKDD tasarımı yapılırken kullanılan ağaçlardan her hangi biri tamamlandığında Şekil 3.9’a benzeyen bir görüntü ortaya çıkacaktır. Şekil 3.9’da daha önce anlatılmayan “Down Sample” simgesi Down Sampling işleminin yapılabilmesi için Xilinx Blockset kütüphanesinden eklenmiştir. Şekilde görüldüğü üzere her seviye için ayrı ayrı alçak geçiren ve yüksek geçiren filtre modülleri kullanılmıştır.



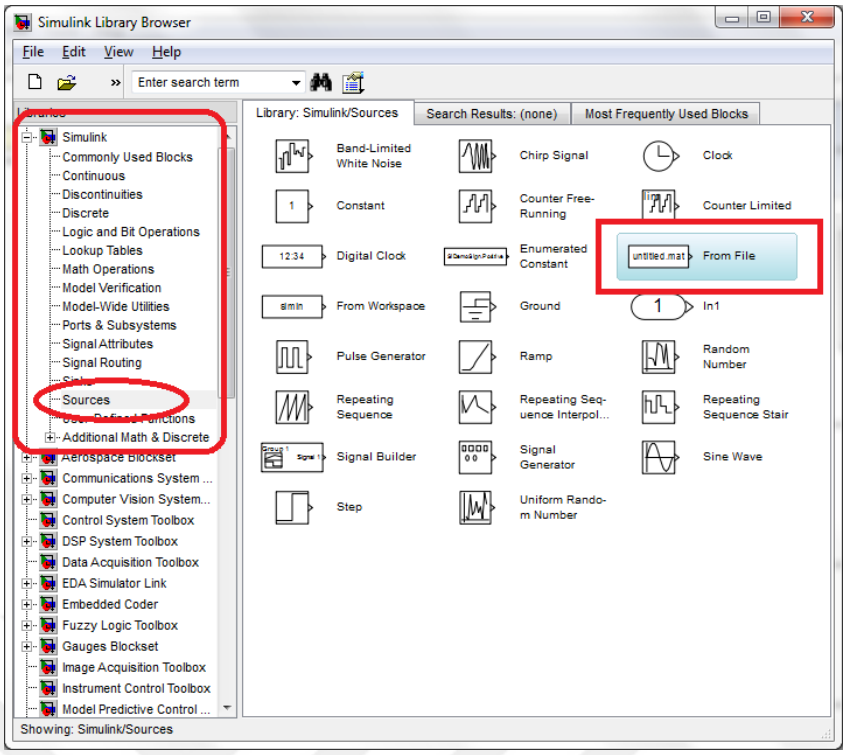
Şekil 3.9 ÇAKDD Algoritmasının System Generator Üzerindeki Tasarımı

Simülasyonun çalıştırılıp, testin gerçekleşebilmesi için girdi verileri ".mat" uzantılı bir dosyaya simülasyon programının belirtmiş olduğu formatta kaydedilmiştir. Dosayada bulunan verinin formatı ilk satırda zaman bilgisi, daha sonraki satırda ise verinin kendisi olacak şekildedir. Burada dikkat edilmesi gereken husus ilk satırda zaman verisi yazılı olduğu için parametreler baştan sona doğru artan bir düzene sahip olmak zorundadır (Şekil 0.20).

	1	2	3	4	5	6
1	1	2	3	4	5	6
2	-160.77	-63.374	-50.385	112.76	-94.650	-178.28

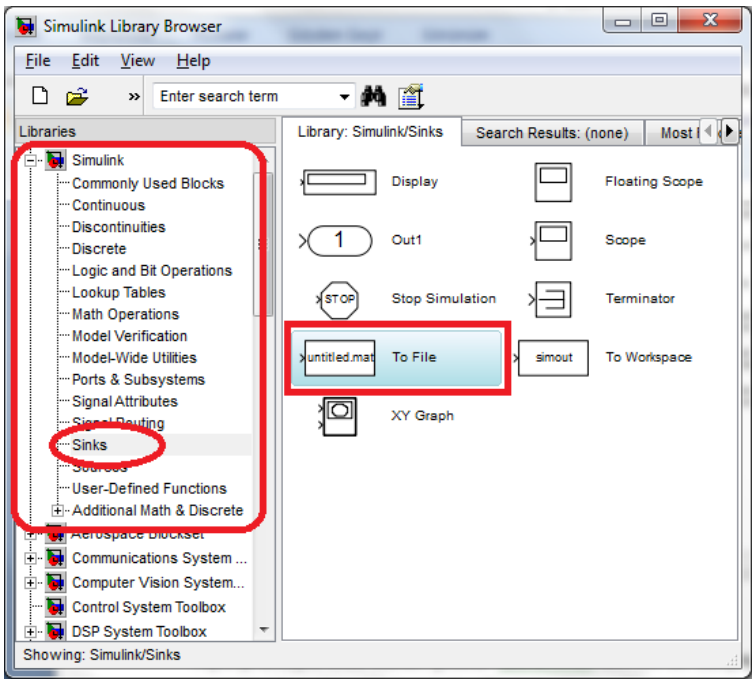
Şekil 3.10 Sonuçlar

Kaydedilen bu verilerin test esnasında dosyadan okunarak sisteme beslenmesi gerekliliği ortaya çıkmıştır. İşlemin yapılabilmesi için Simulink kütüphanesinden "*Sources"* seçeneği tıklanarak açılan simgelerden *"From File"* (Şekil 3.11) simgesi modele eklenir.



Şekil 3.11 Kaynaklar Menüsü

Aynı prensiple elde edilen çıktılar tekrar.mat uzantılı bir dosyaya kaydedilerek beklenen sonuçlarla uyumlu olup olmadığı kontrol edilmiştir. Elde edilen çıktıların dosyaya yazılabilmesi için ise Simulink kütüphanesinin altında *Sinks* seçeneğine tıklanarak açılan simgelerden *"To File"* (Şekil 3.12) modele eklenir.



Şekil 3.12 Çıktı Menüsü

Sistem sabit nokta formatıyla tasarlanmış olup hız verimliliği ön plandadır. Bilindiği üzere sabit nokta formatı kullanmak işaret işleme gibi toplama ve çıkarma işlemlerinin çokça kullanıldığı sistemlerde donanımsal ve zamansal açıdan çok büyük bir verimlilik sağlamaktadır. Ancak System Generator programının üretmiş olduğu tasarım, alan açısından istenilen verimi sağlamamıştır. Tasarım, deneylerimiz sırasında kullanmış olduğumuz Xilinx Spartan-6 board'una sığmamıştır. Yapılan denemeler sonucunda ancak bir alçak veya yüksek geçiren filtre modülü Spartan-6 board'una sığmıştır. Tasarımının tamamı ancak daha gelişmiş bir board kullanımı ile mümkün görünmektedir.

3.6 Sonuç

Şu ana kadar yapılan çalışmaların ışığında; ilk olarak, çalışmanın nihayetinde gerçek zamanlı bir donanımın ortaya konulacağı göz önüne alındığında hız ve kapladığı yer itibariyle sabit-nokta aritmetiği ile devam edilmesi gerektiği görülmüştür. İkinci olarak, bilgisayar ortamında ve mikrodenetleyici ortamında yapılan hız testlerinde gerçek zamanlı olarak yapılabilecek olan işlemlerin düşük frekansta olması gerektiği sonucu çıkmıştır. Yüksek frekanslarda ise tezin sonunda gerçeklenmesi hedeflenen FPGA ortamına ihtiyaç duyulduğu görülmüştür.

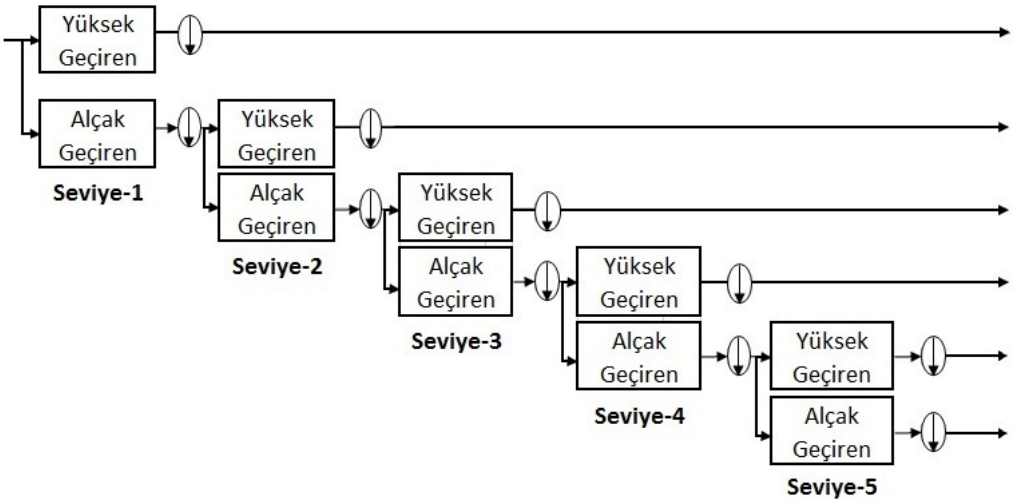
BÖLÜM 4

FPGA PLATFORMUNDA YAPILAN ÇALIŞMALAR

Önceki bölümde bahsedildiği üzere System Generator programı ile yapılan çalışmada hız verimliliğinin ön planda olduğu bir tasarım ortaya çıkmış ve bu durumun, alan açısından sıkıntıya sebep olduğu gösterilmiştir. Anlatılanlar ışığında, ÇAKDD'nin FPGA üzerinde donanımsal gerçeklemesi yapılırken tasarımın, alan verimliliğine ve düşük güç tüketimine sahip olması hedeflenmiştir. Bu amaçla ÇAKDD algoritmasının kullanım amacına göre farklı hız ve alan verimliliklerine sahip tasarımları ortaya konulmuştur. Ayrıca asgari miktarda hafıza elemanı kullanılarak alan verimliliği azami ölçüde sağlanmıştır. Bu algoritmaların en önemli işlem bileşenleri çarpıcı ve toplayıcılardır. Alan etkinliğine ve işlem hızına doğrudan etki eden çarpıcı ve toplayıcıların sayısına göre tasarlanan mimariler aşağıda anlatılmıştır.

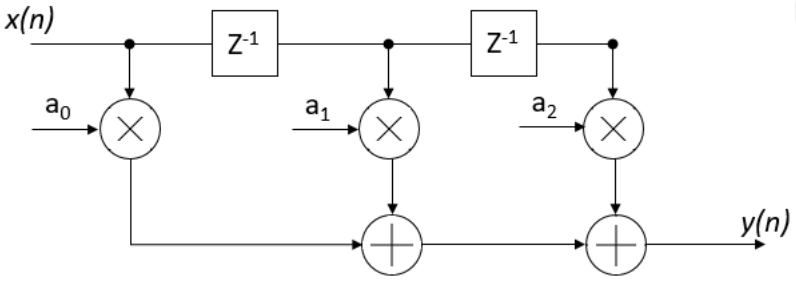
4.1 Her Ağaç İçin Bir Çarpıcı ve Bir Toplayıcı

Teorik olarak Şekil 4.1'de her seviyede yüksek geçiren ve alçak geçiren olmak üzere ikişer süzgecin uygulandığı klasik yöntem yazılımsal vb. gerçeklemelerde sıkça kullanılmaktadır. Ancak Şekil 4.1'de görüldüğü biçimi ile ÇAKDD'yi bire bir gerçeklemek verimli bir yöntem olmamaktadır. Yazılımsal veya donanımsal olarak bir takım yöntemlerle optimize edilmesi gerekmektedir. Donanımsal olarak bire bir uygulanması durumunda ise hız verimliliği açısından azami verim sağlarken alan açısından ise asgari verime neden olur. Yapılan çalışmada alan verimliliği açısından neler yapılabileceği adım adım incelenerek donanımsal gerçeklenme safhaları anlatılmıştır.



Şekil 4.1 Çift-Ağaç Karmaşık Dalgacık Dönüşümündeki Ağaçlardan Biri

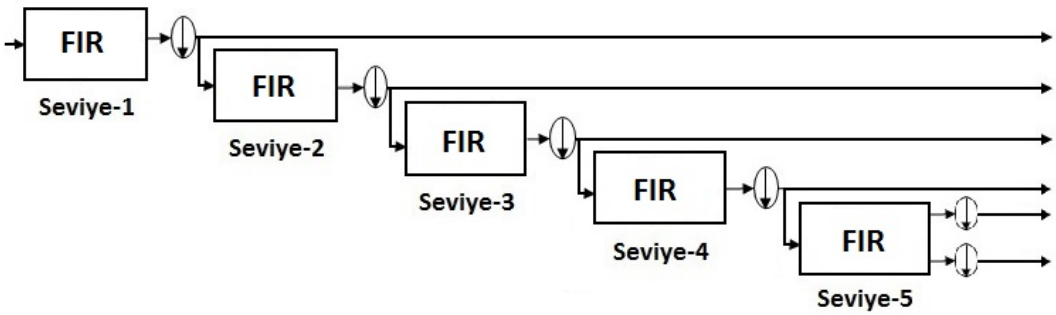
Şekil 4.2'de de görülebileceği üzere FIR süzgeçin içyapısı incelendiğinde, (N-Tap'li bir süzgeç için) gelen son N girdi verisi ara bellekte tutulmakta ve her biri sırasıyla ilgili katsayı ile çarpılarak elde edilen değerler toplanır ve bir sonuç elde edilir. Bu bilgiler ışığında Şekil 4.1 dikkatli incelendiğinde aynı seviyedeki yüksek ve alçak geçiren filtrelerin ara belleklerinde tuttuğu değerlerin aynı olduğu görülmektedir.



Şekil 4.2 FIR Filtre

Alan verimliliği ön planda olan bir tasarımda aynı verileri tutan iki ayrı belleğin bulunması kabul edilebilir bir durum değildir. İki ayrı bellek tutmak yerine tek bir bellekte bu işi yapmak mümkündür. Fakat bu durumda alçak geçiren ve yüksek geçiren filtreler birbirinden bağımsız tasarlanamamaktadır. İki süzgecin bir arada bulunduğu modelde ise yüksek ve alçak geçiren süzgeçlerin katsayılarının tutulduğu iki ayrı bellek bulunmaktadır. Sırasıyla önce alçak geçiren süzgecin katsayıları ile ara bellekteki değerler çarpılır daha sonra ise aynı işlem yüksek geçiren süzgecin katsayıları için tekrarlanır. Yapılan değişiklikle ortaya çıkan yeni mimari Şekil 4.3'te görünmektedir. Bu

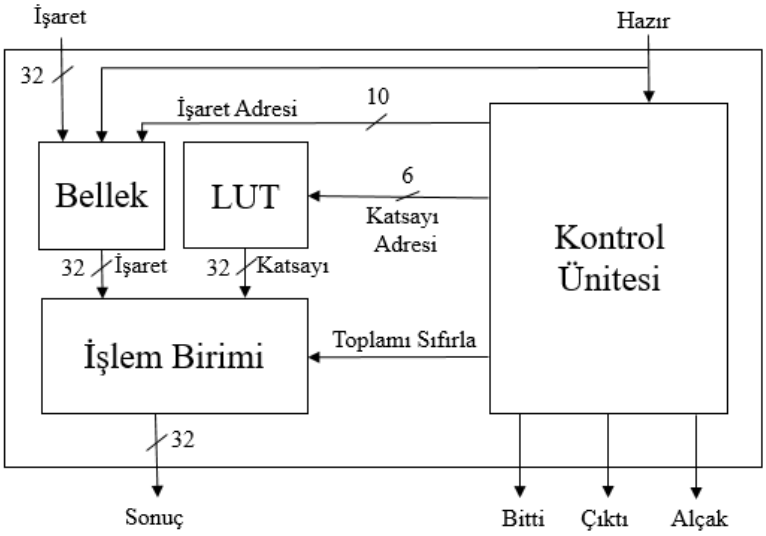
sayede hafıza birimlerinden *N×Seviye Sayısı×Bit-Genişliği* kadar bir tasarruf sağlanmıştır.



Şekil 4.3 Çift-ağaç Dalgacık Dönüşümünün Güncellenmiş Modeli

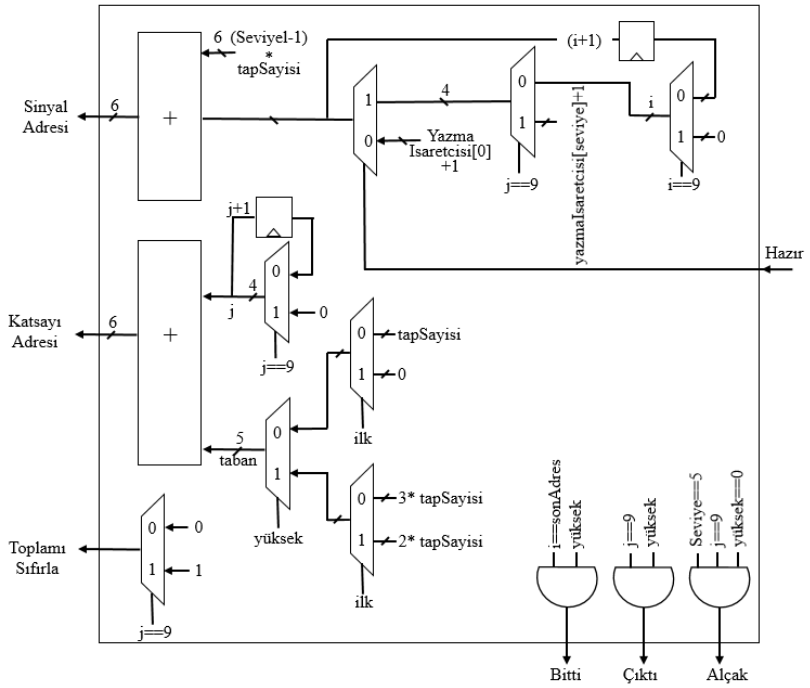
Hız verimliliği ön planda olan bir modelde *N×Seviye Sayısı* kadar toplayıcı ve aynı sayıda da çarpıcı kullanılması gerekmektedir. Bu şekilde bir yaklaşım, gelen veri hızının yüksek olduğu durumlarda tercih edilebilir ve maliyeti çok ciddi şekilde arttırmaktadır. Biyomedikal işaretler gibi nispeten düşük hızlı ancak çok kanallı paralel verilerin işlenmesi durumunda alan verimli modeller tasarlamak maliyet açısından yüksek kazançlar sağlamaktadır.

Mantıksal devrelerde çarpma, toplama vb. aritmetik işlemleri yapmak önemli miktarda maliyete sebep olmaktadır. Belirtilen durum göz önüne alındığında, alan açısından daha da verimli olması amacıyla modelin tamamı için yalnızca bir toplayıcı ve bir çarpıcı kullanmanın, en önemli tasarrufu sağlayacağı aşikârdır.



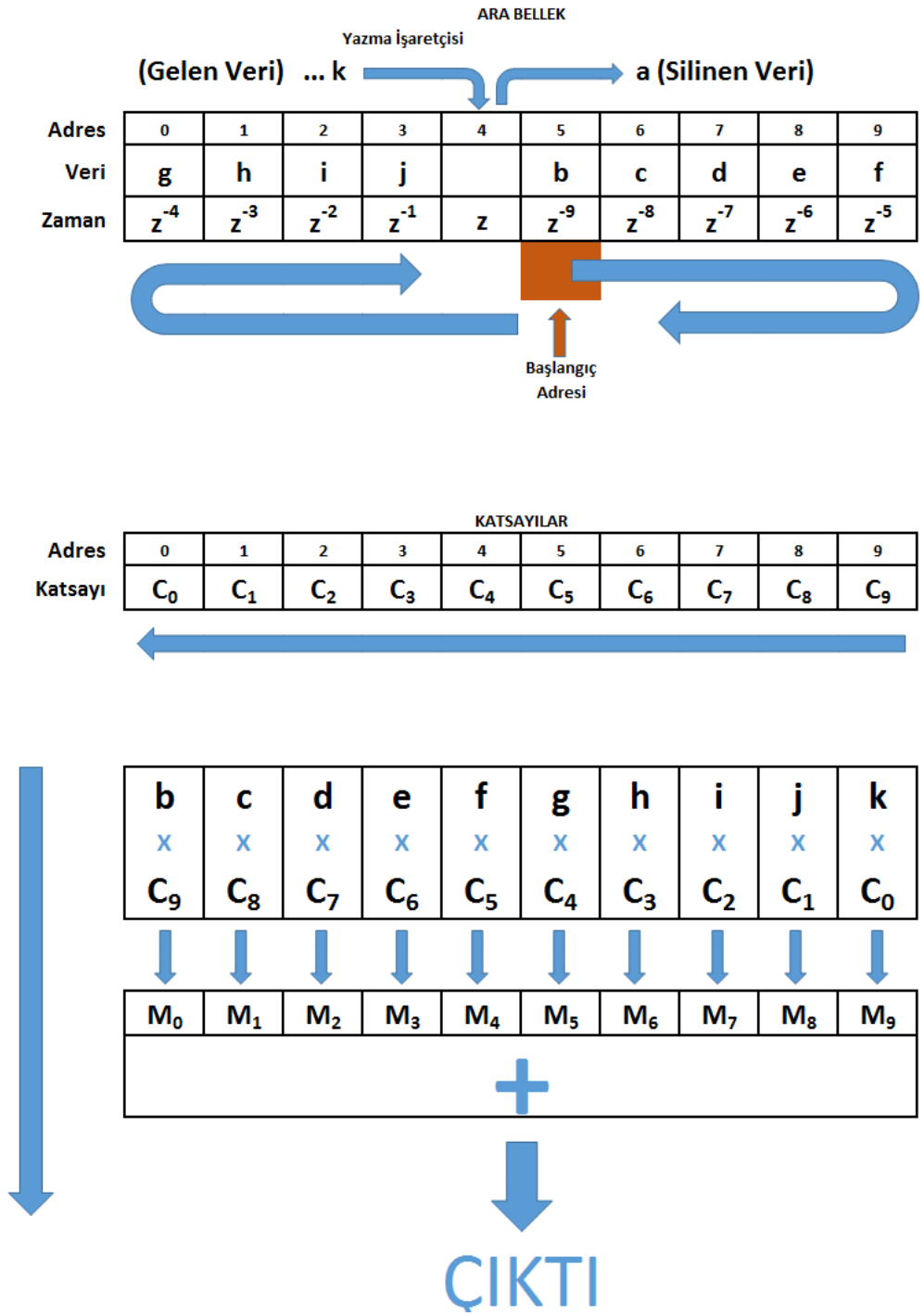
Şekil 4.4 32-Bit Mimarinin Genel Görünümü

Ayrıca tek toplayıcı ve çarpıcının olacağı bir tasarımda süzgeçlerin Şekil 4.3'te görüldüğü gibi bağımsız düşünülmesi söz konusu olamamaktadır. Fakat modeli bu haliyle tasarlamak daha da zorlaşmaktadır. Alan verimliliği sağlarken tasarım zorluğu nedeni ile gerçekleme süresinden feragat edilmektedir.



Şekil 4.5 Kontol Ünitesinin İç Yapısı

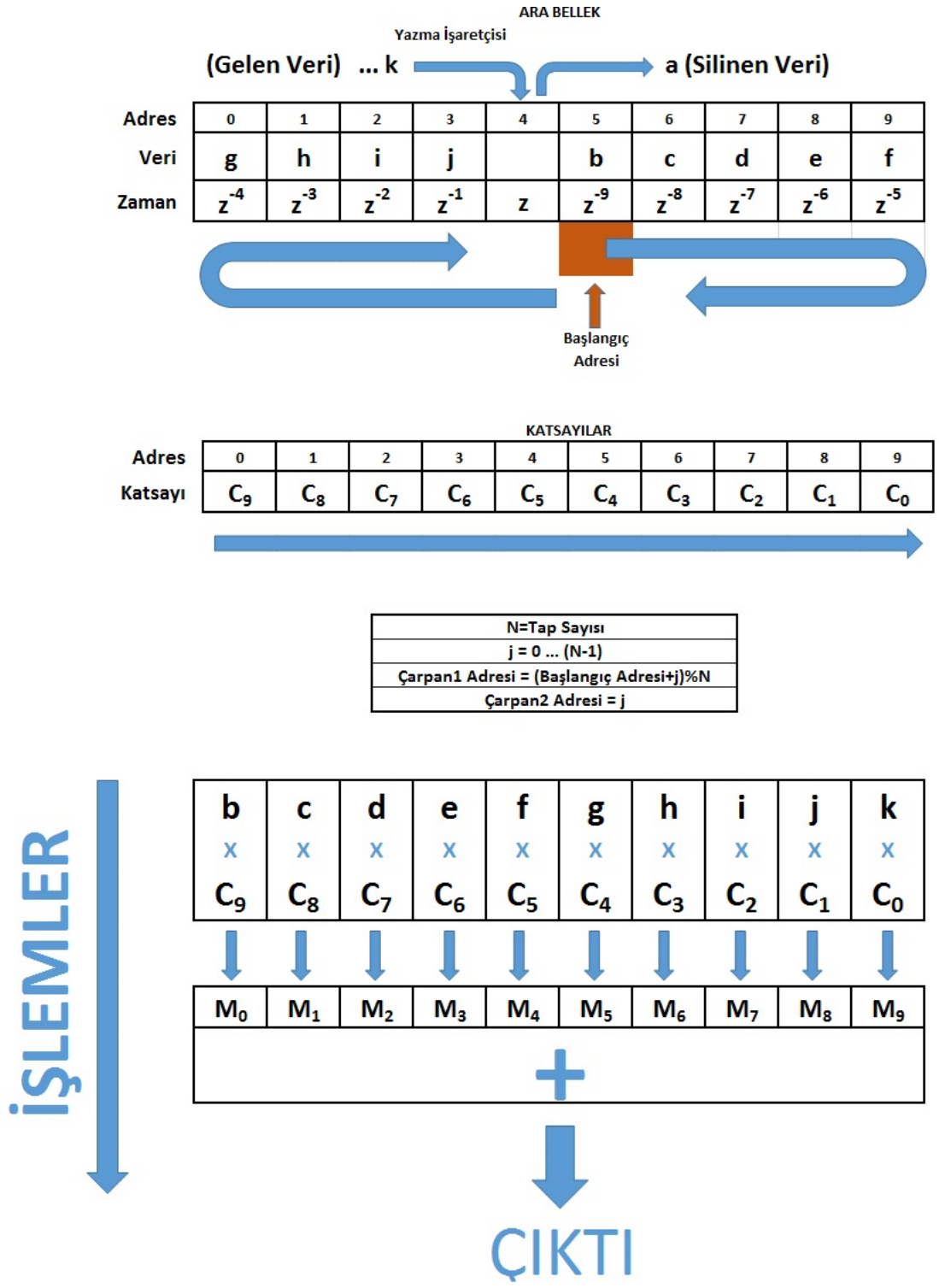
ÇAKKD'de, her ağaç için yalnızca bir toplayıcı ve bir çarpıcı kullanılması durumunda her saat tetiklemesinde yalnızca bir toplama ve bir çarpma işlemi yapılabileceği sonucu çıkar. FIR süzgece gelen her girdi verisine karşılık en az N saat tetiklemesine ihtiyaç duyulur ve bu işlemlerin döngü yöntemi ile yapılması gerekmektedir.



Şekil 4.6 Yeni Modele Ait Algoritma

Şekil 4.6'da, kullanılan yöntem gösterilmiştir. Yapılan işlemleri anlatmak gerekirse; geliştirilen tasarımda düşük güç tüketimi hedeflendiği için klasik yöntemde yapıldığı gibi ara bellekte bir kaydırma işlemi yapılmamaktadır. Bunun yerine "Yazma İşaretçisi" kullanılarak mantıksal kaydırma yapılmaktadır. Bu sayede her veri gelişinde klasik yöntemdeki gibi N adet yazma işlemi yapmak yerine yalnızca yazma işaretçisinin gösterdiği alana yazma işlemi yapılmaktadır.

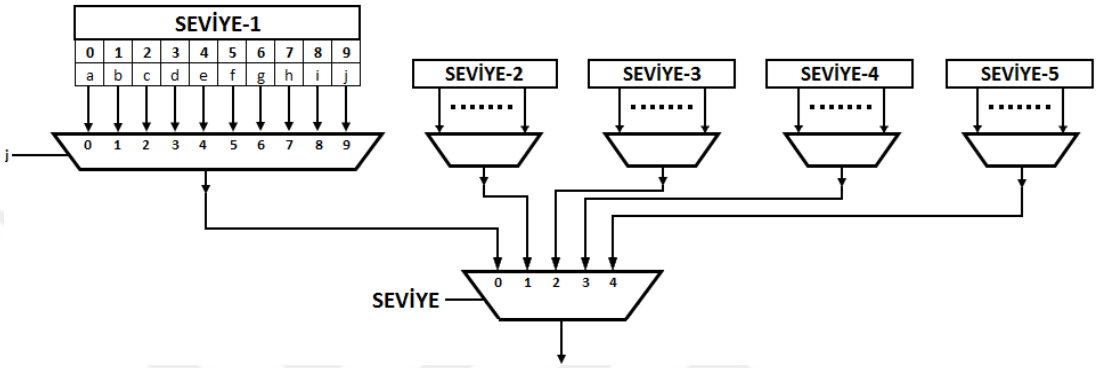
Gelen veri yazılırken aynı anda çarpma işlemi ile süreç işletilmeye başlanır. Ara bellekten alınacak ilk veri, yazma işaretçisinde tutulan değerin bir fazlasının mod N (Yazma işaretçisi+1 mod(N)) adresindeki veridir. Döngü bu değerle başlar, en son olarak yazma işaretçisinin gösterdiği adresteki değerle son bulur.



Şekil 4.7 Yeni Modele Ait Algoritmanın Son Hali

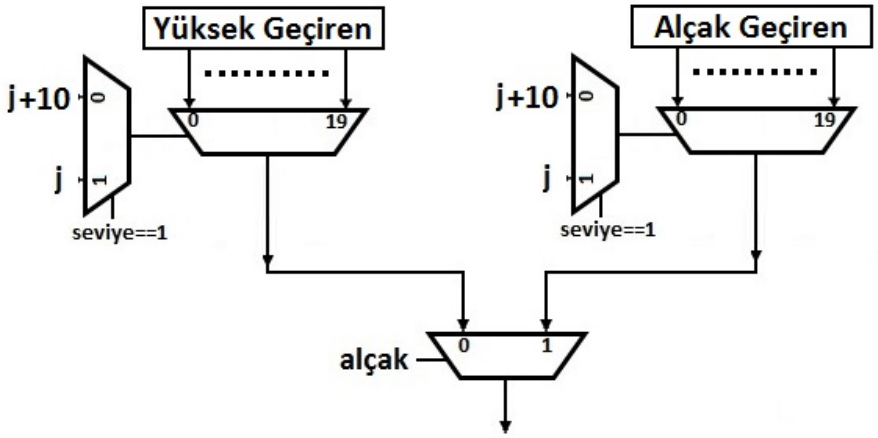
Katsayı seçimine dikkat edildiği takdirde N-1 adresindeki değerle başlayıp 0 adresindeki değerle biten bir döngü ile karşılaşılmaktadır. Döngüyü daha kolay yapabilmek amacıyla katsayıların sırası tersine çevrilirek döngü, klasik şekliyle 0'dan başlayıp N-1'e kadar çalışacaktır. Bu durumda model Şekil 4.7'deki gibi görünecektir.

Şu ana kadar anlatılanlar yalnızca bir FIR süzgeç için yapılan işlemlerdir. Bir bütün olarak ele alındığında ise ara bellekler Şekil 4.8'deki gibi görünmektedir.



Şekil 4.8 Ara Bellekler

Katsayılar ise "Alçak Geçiren İlk" (A.G. İlk), "Alçak Geçiren" (A.G.), "Yüksek Geçiren İlk" (Y.G. İlk), "Yüksek Geçiren" (Y.G.) olmak üzere Şekil 4.9'daki gibi tutulmaktadır.



Şekil 4.9 Katsayı Seçimi

Dışarıdan gelen her girdiye karşılık bütün süzgeçlerin çalışması gibi bir durum söz konusu değildir. Çünkü her filtre çıkışında aşağı örnekleme yapılmaktadır. Bu da ilk seviyeden son seviyeye doğru giderken çıktıların logaritmik olarak azaldığını göstermektedir. Hangi filtrenin ne zaman çalışacağına karar verebilmek için bir "durum" (S) değeri tutmak gerekir. Her veri gelişinde S değeri bir arttırılır. Yeni veri

geldiğinde S değerine bakılarak kaçıncı seviyeye kadar filtrelerin çalıştırılacağı kararı verilir. Örneğimizde 5 seviye kullanılmıştır ve 2^5 adet durumun kendini sürekli tekrar ettiği görülmektedir.

Çizelge 4.1 Durum Değerine Bağlı Koşul Çizelgesi

KOŞUL	Seviye	i
S%2==0	0	0
S%32==1	5	50
S%16==1	4	40
S%8==1	3	30
S%4==1	2	20
S%2==1	1	10

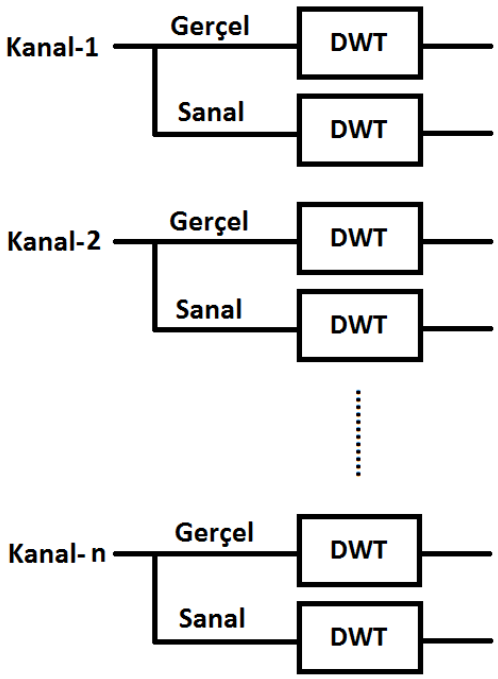
Çizelge 4.1'de S değerine bağlı olarak "Seviye" (kaçıncı seviyeye kadar çalıştırılacağı), "i" (yapılacak işlem sayısı veya döngünün sınır değeri) değerleri görülebilir. Burada dikkat edilmesi gereken nokta S'in çift sayı olduğu durumlarda Seviye değeri 0 olmaktadır. Seviye'nin 0 olması gelen verinin 1. seviyede Yazma İşaretçisi ile gösterilen adrese yazılıp, herhangi bir işlem yapılmamasıdır. İşlem yapılmamasının sebebi ise örnek indirgeme işleminin gerçekleştirilmesidir. Klasik yönteme ait modele bakıldığında çıktıların indirgendiği görülür. Düşük güç tüketiminin hedeflendiği bir tasarımda kullanılmayacak bir çıktının üretilmesi için işlem yapılmaması gerekir. Gerçeklenen donanımın tamamında örnek indirgeme işlemi yukarıda anlatıldığı gibi yapılmaktadır.

Çizelge 4.2'de, duruma (S) bağlı olarak hangi seviye için hangi işlemin nasıl yapılacağı gösterilmiştir. "-" işaretlenmiş alanlarda ilgili seviye için herhangi bir işlem yapılmayacağı, "↓" ile işaretlenmiş alanlarda ise ilgili seviyeye yazma işaretçisinin gösterdiği bellek alanına yazma işlemi yapılacağı, sarı renk ile işaretlenen yerlerde ise yazma işaretçisinin gösterdiği alana yazma işleminin yanı sıra filtrenin normal şekliyle çalışacağı ifade edilmiştir.

Çizelge 4.2 Duruma Bağlı Olarak İlgili Seviyede Yapılan İşlemler

	SEVİYE						
S	**1**	**2**	**3**	**4**	**5**	i	Koşul
0	↓	-	-	-	-	0	S%32==0
1						50	S%2==1
2	↓	-	-	-	-	0	S%2==0
3		↓	-	-	-	10	S%2==1
4	↓	-	-	-	-	0	S%4==0
5			↓	-	-	20	S%2==1
6	↓	-	-	-	-	0	S%2==0
7		↓	-	-	-	10	S%2==1
8	↓	-	-	-	-	0	S%8==0
9				↓	-	30	S%2==1
10	↓	-	-	-	-	0	S%2==0
11		↓	-	-	-	10	S%2==1
12	↓	-	-	-	-	0	S%4==0
13			↓	-	-	20	S%2==1
14	↓	-	-	-	-	0	S%2==0
15		↓	-	-	-	10	S%2==1
16	↓	-	-	-	-	0	S%16==0
17					↓	40	S%2==1
18	↓	-	-	-	-	0	S%2==0
19		↓	-	-	-	10	S%2==1
20	↓	-	-	-	-	0	S%4==0
21			↓	-	-	20	S%2==1
22	↓	-	-	-	-	0	S%2==0
23		↓	-	-	-	10	S%2==1
24	↓	-	-	-	-	0	S%8==0
25				↓	-	30	S%2==1
26	↓	-	-	-	-	0	S%2==0
27		↓	-	-	-	10	S%2==1
28	↓	-	-	-	-	0	S%4==0
29			↓	-	-	20	S%2==1
30	↓	-	-	-	-	0	S%2==0
31		↓	-	-	-	10	S%2==1

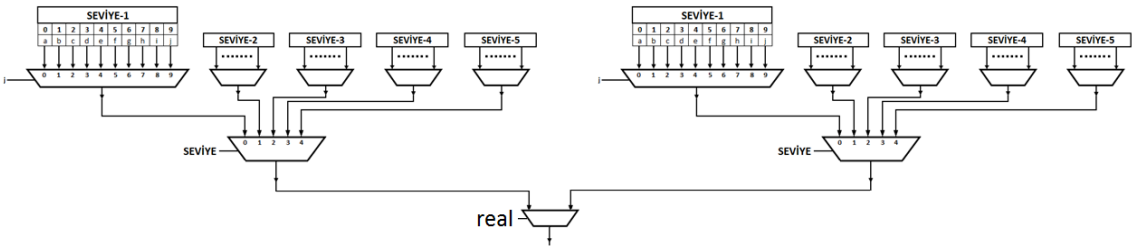
Şekil 4.9'da tek toplayıcı ve çarpıcı ile geliştirilen n-kanallı bir tasarıma ait mimari verilmiştir. Mimaride de görülebileceği üzere ağaçların birbirleri ile herhangi bir bağlantısı bulunmamakta ve paralel olarak çalışmaktadırlar.



Şekil 4.10 Her Ağaç İçin Tek Toplayıcı ve Çarpıcı Mimarisinin Çok Kanallı Yapısı

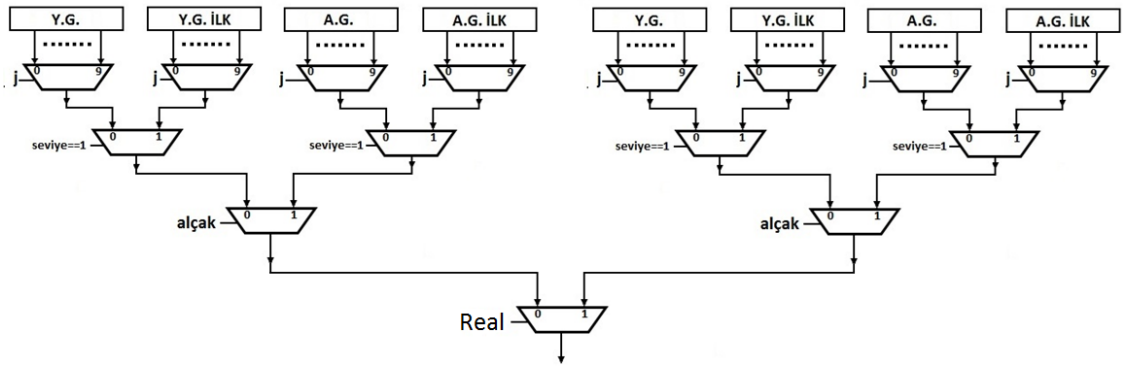
4.2 Her Bir Kanal İçin Tek Çarpıcı ve Tek Toplayıcı

Bir önceki bölümde, ÇAKDD'nde, her bir ağaç için bir toplayıcı ve bir çarpıcı kullanılmıştı. Alan verimliliğin arttırılması adına ÇAKDD, komple bir modül haline getirilerek tek bir toplayıcı ve tek bir çarpıcı ile gerçeklenmiştir. Bu durum alan verimliliği sağlamasına karşılık hızda yarı yarıya bir düşüşe neden olmuştur. Yüksek olmayan veri giriş hızları için bu durum herhangi bir dezavantaja sebep olmamaktadır.



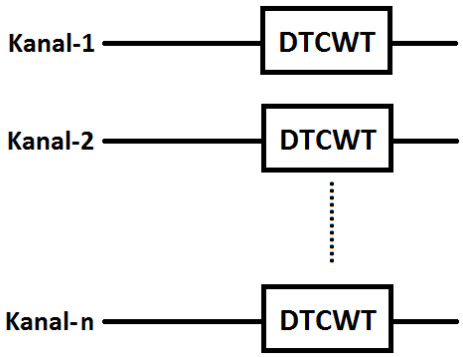
Şekil 4.11 Arabellek Organizasyonu

Şekil 4.11'de gelen işaretlerin kayıtlı olduğu arabellekten çarpıcıya gidecek olan verinin seçimi için gerekli olan tasarım gösterilmiştir. Herhangi bir işaret geldikten sonra sistem, öncelikle gerçek ağaç için çalışmaya başlar. Gerçek ağaçla ilgili işlemler bittikten sonra ise real işareti sıfıra çekilerek sanal ağaç için hesaplamalar başlatılır.



Şekil 4.12 Katsayı Seçimi

Şekil 4.12'de çarpıcı modüle aktarılması gereken katsayının seçimi için yapılan tasarım gösterilmiştir. İlk yapılan kontrol gerçek ağaçla mı çalışıldığı, ikinci olarak alçak geçiren filtre ile mi çalışıldığı. Son olarak ise birinci seviye filtre ile ilgili mi işlem yapıldığıdır.



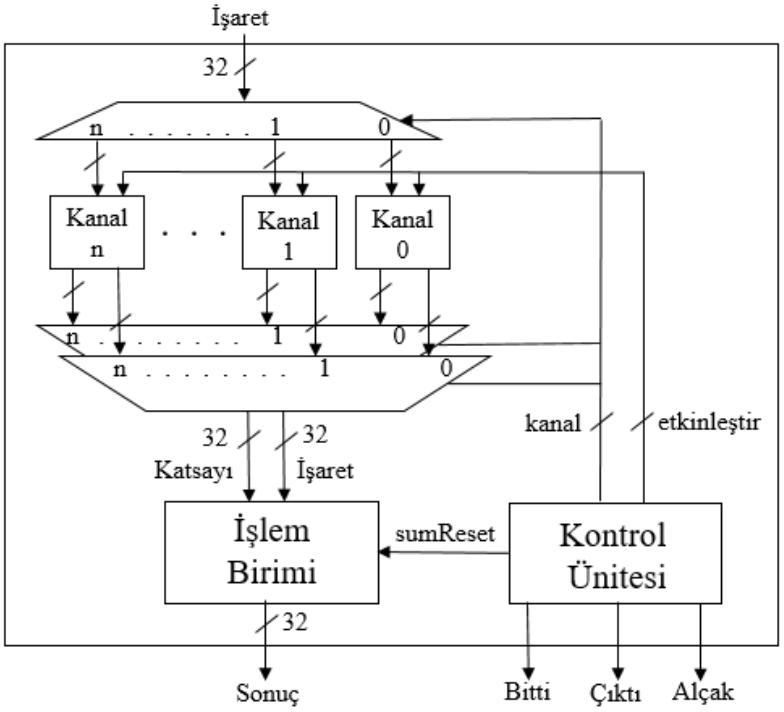
Şekil 4.13 Her Kanal İçin Tek Toplayıcı ve Çarpıcı Tasarımının N-Kanallı Genel Tasarımı

N-kanallı bir sistemin, her kanal için tek toplayıcı ve çarpıcı mimarisi kullanılarak gerçeklenmesi halinde Şekil 4.13'teki gibi bir tasarım ortaya çıkmaktadır. Fakat modeli, bu şekliyle tasarlamak daha da zorlaşmaktadır. Alan verimliliği sağlanırken tasarım zorluğu nedeni ile gerçekleme süresinden feragat edilmektedir.

4.3 Çok Kanallı Sistem İçin Tek Çarpıcı ve Tek Toplayıcı

Göreceli olarak daha önceki tasarımlara göre daha düşük hızlarda çalışabilen sistemlerde alandan daha da tasarruf edebilmek adına geliştirilmiş bir tasarımdır. Bu tasarımın ilk zorluğu sisteme gelen verilerin sırasıyla ilgili kanala ait hafızaya doğru bir şekilde yazılmasıdır. Veri yolundan gelen sayıların kanal sırasına uygun olarak geldikleri varsayılmıştır. Verinin üzerinde herhangi bir ekstra kanal bilgisi bulunmamaktadır.

Sisteme her gelen veri ile birlikte, sırasıyla birinci kanaldan n. kanala kadar bu veriler iletilir. İlk kanala yazılan veri ile birlikte birinci kanala ait hesaplamalar yapılır ve sonuçlar üretildikten sonra bir sonraki kanal aktif hale gelir. Bu şekilde sonuncu kanala ait işlemler de tamamlandıktan sonra tüm sonuçlar yine aynı sıralamayla gönderilir ve yeni veri istenir. Şekil 4.14'te genel mimari görülmektedir.



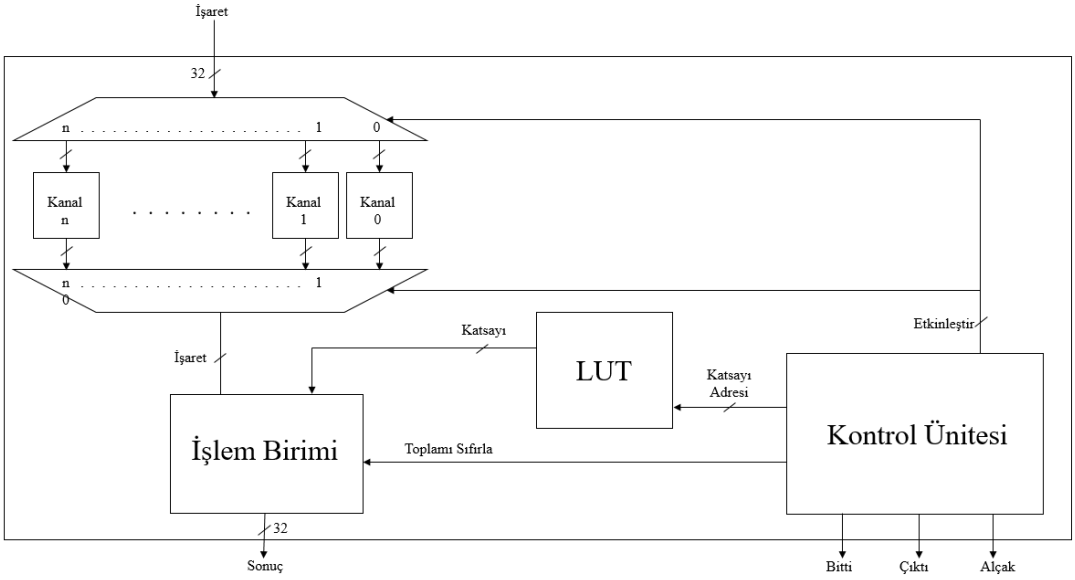
Şekil 4.14 Çok Kanallı Genel Mimari

Bu tasarımda toplayıcı ve çarpıcılardan sağlanan kazanç ile alan verimliliği artar ancak daha önceki tasarımlarla (her kanal için tek bir toplayıcı ve çarpıcı veya her ağaç için bir toplayıcı ve çarpıcı) kıyaslandığında kanal sayısının artması ile ters orantılı olarak hızda azalma olur. Önceki çalışmalarda her kanal için işlemler paralel olarak çalışırken tek çarpıcı ve tek toplayıcı kullanıldığında, her kanalın işlemleri seri şekilde yapılır.

4.4 Çok Kanallı Yapının Tek Bir LUT ile Tasarlanması

Çok kanallı sistemlerle çalışırken karşılaşılan senaryoların çoğunda kanalların hepsi için aynı katsayı seti kullanılmaktadır. Kanallardan gelen verilerin işlendiği modüllerin içinde katsayıların tutulduğu LUT'lar ciddi bir hafıza kullanımı doğurmaktadır.

Tüm kanalların aynı katsayıyı kullandığı durumlar için şekil 4.13'teki mimaride, alt modüllerden LUT'lar kaldırılarak şekil 4.14'teki gibi üst modülde tek bir LUT kullanımı ile alan tasarrufu sağlanmıştır.

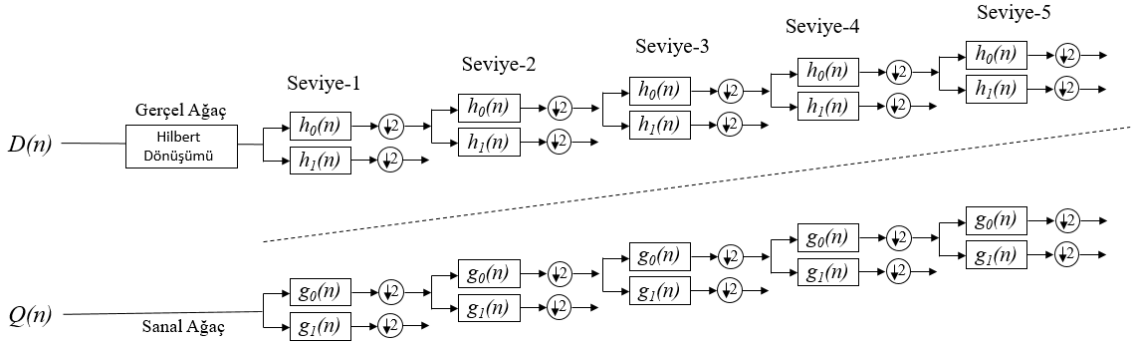


Şekil 4.15 Tek LUT ile Geliştirilen Çözüm

N-kanallı bir yapıda N adet LUT kullanılırken Şekil 4.15'te görülen mimari ile kanal sayısı ne olursa olsun yalnızca 1 adet LUT kullanılmaktadır.

4.5 Değiştirilmiş Çift-Ağaç Dalgacık Dönüşümü

[35]'te belirtildiği üzere ayrık dalgacık dönüşümünün dezavantajlarından biri analiz edilen işaretlerde faz kaymalarına neden olmasıdır. Bu dezavantajın üstesinden gelmek için, ayrık dalgacık dönüşümünün iyileştirilmiş bir gerçekleştirimi olan ÇAKDD kullanılabilir.



Şekil 4.16 Değiştirilmiş Çift-Ağaç Dalgacık Dönüşümü

Fakat Doppler işaretlerinde, ileri ve geri yönlü kan akış işaretlerinin bilgisini kodlanmış şekilde içeren *D(*n*)* ve *Q(*n*)* bileşenlerinin ayrı ayrı ÇAKDD ile işlenmesi gerekmektedir.

Bu durum yapılan işlem sayısını iki katına çıkartmaktadır. [31]'de ÇAKDD ile hemen hemen aynı performansı gösteren ve yarısı kadar işlemsel karmaşıklığa sahip modifiye edilmiş çift-ağaç karmaşık dalgacık dönüşümü (MEÇAKDD) önerilmiştir. Çalışmanın bu bölümünde gerçek zamanlı çalışan sistemlerdeki işlemsel karmaşıklığı azaltmak amacıyla DÇAKDD FPGA ortamında gerçeklenmiş ve elde edilen sonuçlar sunulmuştur.

Şekil 4.16'da DÇAKDD'nin genel mimarisi görülmektedir. DÇAKDD'de Dopler işaretini oluşturan D(n) ve Q(n) bileşenlerinden birinin öncelikle Hilbert dönüşümü alınmaktadır ve bu sayede iki bileşen arasındaki faz farkı 90^0'den 180^0 ya da 0^0'ye değiştirilmektedir. Daha sonra bileşenlerden biri ÇAKDD'nin gerçel ağacına diğer bileşen ise sanal ağacına ayrı ayrı sokulmaktadır ve bu ağaçlar kullanılarak analiz ve sentez yapılmaktadır. Dönüşümün sentez kısmında yapılan değişiklikler sayesinde hem Doppler işaretinden yön işaretleri elde edilmekte hem de analizin ara basamaklarında sadece dalgacık katsayıları kullanılarak işaretin incelenmesinin yolu açılmaktadır.

Çok kanallı sistemler için DÇAKDD gerçeklemesi ÇAKDD gerçeklemesinde olduğu gibi yapılamamaktadır. Hilbert dönüşümünden kaynaklı bir veri besleme problemi ortaya çıkmaktadır. Tez çalışmalarında öncelikli amacın alan verimliliği olduğu göz önüne alınarak en az sayıda Hilbert dönüşüm modülü kullanarak DÇAKDD'ın gerçeklenmesi hedeflenmiştir. Her ağaç için tek toplayıcı tek çarpıcı yönteminde Gerçel ve sanal ağaçlar birbirinden bağımsız olarak çalıştığı için Hilbert dönüşümünde kullanılan TAP sayısı kadar bir saat tetiklemesi sürelik gecikme ile çıktılar elde edilir. Diğer yöntemlerde ise gerçel ve sanal ağaçların, işlem birimini ortak kullanmalarından kaynaklı bir veri besleme problemi ortaya çıkmaktadır. Gerçel ağaca veri beslenmeden önce, gelen veriye Hilbert dönüşümü uygulanırken aynı anda sanal ağaç için filtreleme işlemleri başlatılır. Sanal ağaca ait işlemler bitinceye kadar gerçel ağaca beslenecek olan verilerin Hilbert dönüşümünden çıkması gerekmektedir.

Çizelge 4.3 - Zamanlama Çizelgesi

ÇAKDD	Sanal-1	Sanal-2	Sanal-3	Sanal-4	Sanal-5	Sanal-6	Sanal-7	Sanal-8	Gerçel-1	Gerçel-2	Gerçel-3	Gerçel-4	Gerçel-5	Gerçel-6	Gerçel-7	Gerçel-8
Hilbert	Gerçel-1	.	.	.	.	.	.	.	.	.	.	.	.	.	Gerçel-8	-

8-kanallı bir sistemin tek toplayıcı ve tek çarpıcı ile gerçeklendiği senaryoyu incelersek; Çizelge 4'te görüldüğü üzere ÇAKDD işlemleri Sanal ağaçlar için sırasıyla yapılır ve ardından Gerçel ağaçlara geçilir. Buradaki en önemli husus, son Gerçel ağaca ait işlemler yapılmaya başlanmadan evvel Hilbert dönüşümünden ilgili verinin hazırlanmış olmasıdır. Gerçel ağaca girecek verilerin gecikmesi sistemin çalışması açısından hatalı çalışma ve yanlış çıktı verme gibi problemlere yol açmamaktadır. Ancak iki modül arasındaki gecikmeler sistemin genel çalışma hızının düşmesine sebebiyet vermektedir. Hilbert dönüşümünden Gerçel ağaçlara beslenecek verilerde herhangi bir gecikme yaşanması halinde Hilbert modül sayısının arttırılması gereği ortaya çıkmaktadır. Aşağıda görülen eşitsizlik kullanılarak gereksinim tespit edilebilir.

$$40 x KanalSayısı - 20 \geq HilbertTAP x KanalSayısı \quad (4.1)$$

Eşitsizliğin kullanılması ile elde edilen sonuçların bir kısmı Çizelge 5'te görülebilir.

Çizelge 4.4-Kanal sayısı ve Hilbert modül sayısına bağlı maksimum TAP sayıları

	Kanal Sayısı					
Hilbert Modül Sayısı	**1**	**2**	**4**	**8**	**16**	**32**
1	20	30	35	37	38	39
2	40	60	70	75	77	78
3	40	90	105	112	116	118
4	40	120	140	150	155	157
5	40	120	175	187	193	196

Çizelge 4.4'te Hilbert modül sayısı ile kanal sayısına bağlı olarak olarak Hilbert modülünde kullanılabilecek maksimum TAP sayıları verilmiştir. Kanal sayısı arttıkça

Hilbert modülünde kullanılabilecek TAP sayılarının da arttırılabildiği görülmektedir. Hem ÇAKDD modülünde hem de Hilbert modüllerinde tek toplayıcı ve tek çarpıcı kullanılmıştır.

8-kanallı, 37 TAP Hilbert filtresine sahip bir sistemde 0,197W güç tüketimi ve 346K girdi/sn veri giriş hızına çıkılabilmektedir.

BÖLÜM 5

KOD ÜRETİCİ

Otomatik kod üretme konsepti, yüksek seviye bir dil veya bir program ile alt seviye bir dilde program üretme mantığına dayanmaktadır. Literatürde çeşitli alanlarda bu tarz uygulamaları görmek mümkündür. Kod üretici yazılımların (KÜY) bazıları yapay zeka seviyesinde programlar dahi yazabilmektedirler. Bu yaklaşımın temel amacı olabildiğince hızlı ve kolay bir şekilde optimum programlar yazmak üzerinedir. KÜY'ü geliştiren kişinin problemi çok iyi bilmesi ve en iyi çözümleri sunması beklenir. Bu açıdan bakıldığında KÜY'ler özellikle problemin çözümü ile ilgili herhangi bir fikri olmayan, programlama tecrübesi bulunmayan kullanıcılar için oldukça yararlıdır. KÜY'ü geliştiren kişinin bilgi birikimine sahip olmaya gerek kalmadan en uygun çözüme ulaşmak mümkün olabilmektedir.

KÜY'lerin kullanılacağı problemlerin buna uygun olması gerekmektedir. Çözümün öncelikle parametrik hale getirilmesi, daha sonrasında işlenecek veri ve kullanılacak elemanların belirli bir düzene oturtulması önem taşımaktadır. Aksi takdirde bu tarz bir yazılım ihtiyacı çözmeyeceği gibi neredeyse aynı geliştirme süresine bile gereksinim duyabilir.

Yukarıda bahsedilenler ışığında tez kapsamında çeşitli mimariler ile ÇAKDD metodunun gerçeklendiği ve yöntemin yalnızca biyomedikal alanında değil aynı zamanda birçok farklı konuda da kullanıldığı düşünüldüğünde KÜY ihtiyacının ortaya çıktığı görülmektedir. Yapılan çalışmalarda karşılaşılan problemlere uygun mimarinin seçimi ve adaptasyonu belirli bir zaman almaktadır. Dikkatli olunmadığı takdirde hatalar yapılabilmektedir. Ayrıca tekrar tekrar test edilmediği durumlarda bu hataları yakalamak bazen mümkün olmamaktadır. Bunlara ilave olarak mimariyi geliştiren kişi

haricindeki bir kullanıcının bunları yapması çoğunlukla imkânsız hale gelmektedir. Şu ana kadarki çalışmaların başka araştırmacıların çalışmalarında kullanabilmesi, adaptasyonların çok daha basitleştirilmesi amacıyla bir arayüz geliştirilerek farklı kombinasyonlara ait kodlar bu arayüz ile üretilir hale getirilmiştir.

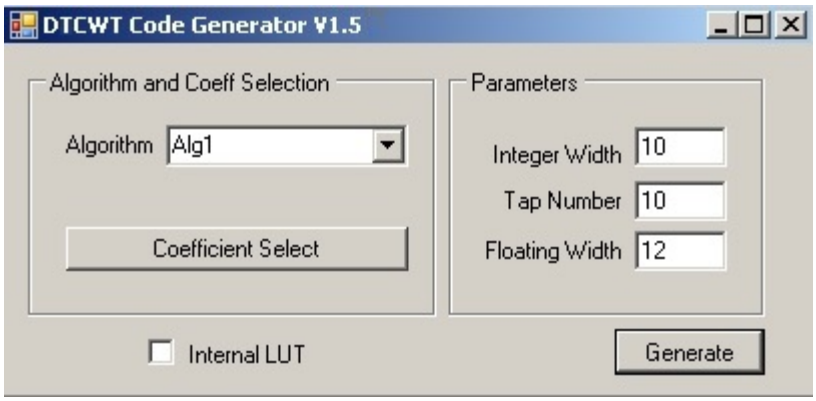
KÜY'de;

- TAP Sayısı
- Bit Genişlikleri
- Katsayıların Seçimi
- Kanal Sayısı

Mimari Türü:

- ***Her Ağaç*** için bir toplayıcı ve çarpıcı
- ***Her Kanal*** için bir toplayıcı ve çarpıcı
- ***Tüm Kanallar*** için bir toplayıcı ve çarpıcı
- ***Her Modül*** için LUT

Şeklinde seçimler yapıldıktan sonra tek tık ile kod üretilerek kullanıma sunulmaktadır.



Şekil 5.1 Kod Üretici Arayüzü

Parametre belirleme işlemleri Şekil 5.1'de görülebileceği üzere gayet kolay ve ilk bakışta anlaşılabilecek şekildedir. Ancak katsayı seçiminde belirli bir kural

işletilmektedir. Katsayı seçimi, *.txt* uzantılı bir dosyadan yapılmaktadır. Dosyanın içeriğindeki her bir satırda yalnızca bir sayı bulunmaktadır. Tap sayısı *n* olmak üzere;

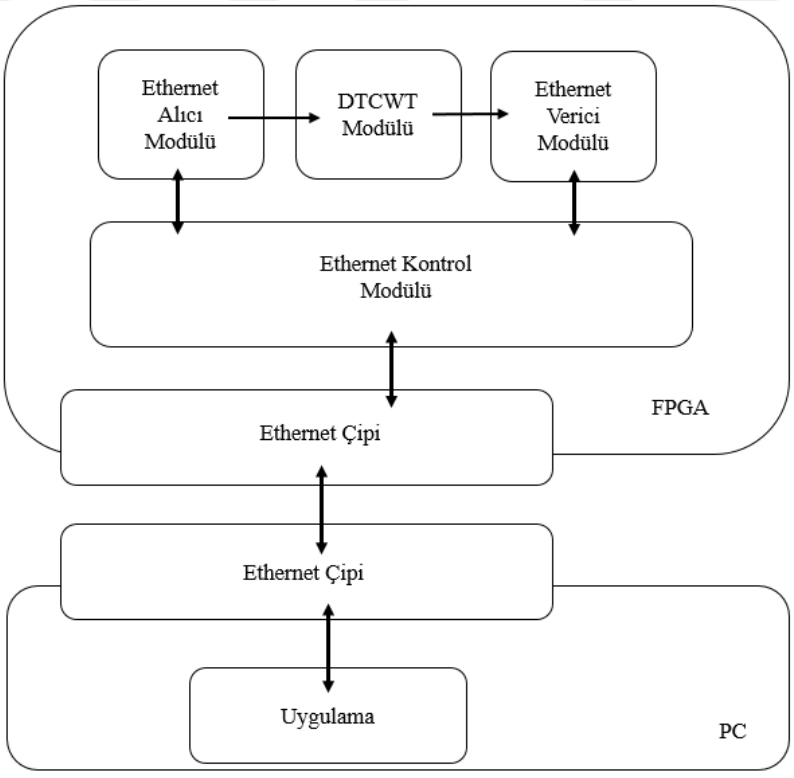
- İlk *n* satırda gerçek ağacın ilk seviyesine ait katsayı seti
- İkinci *n* satırda gerçek ağacın diğer seviyelerinde kullanılan katsayı seti
- Üçüncü *n* satırda sanal ağacın ilk seviyesine ait katsayı seti
- İkinci *n* satırda sanal ağacın diğer seviyelerinde kullanılan katsayı seti

bulunmaktadır. Her kanalda farklı bir katsayı seti kullanılacaksa; her kanala ait set, yukarıda sıralanan mantığa uygun şekilde ilk *4n* satırda ilk seviyeye ait katsayılar, son 4n satırda ise son kanala ait katsayı seti bulunacak şekilde dosya hazırlanır. Bu katsayılar dosyadan okunarak sabit noktalı sayı sistemine çevirilir ve kod içerisinde ilgili alanlara gömülür. Arayüz kullanılmadığı takdirde katsayıların sabit nokta formatına çevirilmesi bile epey vakit almaktadır.

BÖLÜM 6

TESTLER

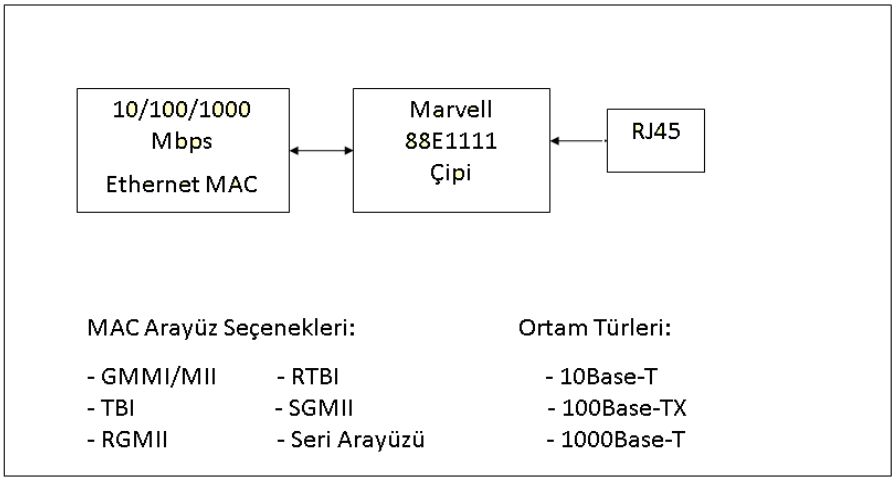
ÇAKDD modüllerine girdi verilerinin beslenebilmesi ve üretilmiş verilerin alınması gerekmektedir. Bu kapsamda kullanılan Spartan6 FPGA geliştirme platformunun üzerinde bulunan PHY çipi ile Ethernet haberleşme modülü [36]'daki çalışma temel alınarak geliştirilmiştir. Haberleşme için UDP, DHCP ve ARP protokollerinin gerçeklemesi yapılmıştır. Şekil 6.1'de sistemin genel işleyişi gösterilmiştir. İçerisinde bilgisayar tarafı için kütüphaneler, donanım tarafında ise çeşitli modüller barındırmaktadır. Haberleşme sisteminin detayları alt başlıklarda verilmiştir.



Şekil 6.1 Genel Mimari

6.1 PHY Çipi

Ethernet kullanılarak yapılan haberleşmelerde PHY çiplerinin kullanımı işlemleri kolaylaştırmaktadır. PHY çipi, fiziksel katmandaki gerekli modülasyonları gerçekleştirmektedir. Bu çipe karşı tarafa gönderilecek olan verinin verilerek, paketin aktarımı sağlanmaktadır. OSI modeli standartlarına göre haberleşme yapılması gerektiği için bu çipin kullanımı önemlidir.



Şekil 6.2 88E1111 Çipinin Uygulamalardaki Kullanımı[39]

Şekil 6.2'de, kullanılan Ethernet çipinin desteklediği 10/100/1000 Mbps hızları gösterilmektedir. Ayrıca farklı MAC arayüzleri de bulunmaktadır. Bu çalışmada gigabit Ethernet olan GMMI arayüzü kullanılmaktadır.

Çizelge 6.1 88E1111 Çipi Bağlantı Pinleri ve Açıklamaları

Pin Adı	Açıklama
MDI	Ortam Bağımlı Arayüz. Ethernet kablolarının crossover kabloya gerek olmadan haberleşmeyi sağlar.
GTX_Clk	GMII Aktarım Saati. Gigabit bağlantı olması sebebi ile 125mhz ile çalışmaktadır. Aktarım için kullanılan saattir. TX_En, TX_Err ve TXD[7:0]

	sinyalleri ile senkrondur.
TX_En	Aktarım Aktif Sinyali. Bu sinyal aktarımın olduğu zamanlarda aktif olmaktadır.
TX_Err	Aktarım Hatası. Kabloda aktarım hatası olduğu zaman aktif olmaktadır.
TXD	Aktarımı yapılan verileri taşır. Aynı anda tek yönlü olarak 8 bit taşıyabilmektedir.
RX_Clk	GMII Alım Saati. Gigabit bağlantı olması sebebi ile 125mhz ile çalışmaktadır. Alım için kullanılan saattir. RX_En, RX_Err ve RXD[7:0] sinyalleri ile senkrondur.
RX_En	Aktarım Aktif Sinyali. Bu sinyal aktarımın olduğu zamanlarda aktif olmaktadır.
RX_Err	Alım Hatası. Kabloda alım hatası olduğu zaman aktif olmaktadır.
RXD	Aktarımı yapılan verileri taşır. Aynı anda tek yönlü olarak 8 bit taşıyabilmektedir.
CRS	Taşıyıcı Algılama. Alma ortamının boş olmadığı durumlarda aktiftir.
COL	Çakışma. Hem aktarım hemde alım ortamı aktif olduğunda aktif olur.

Çizelge 6.1'de Ethernet çipinin sürülmesi gereken pinleri verilmektedir. Çip kullanılarak veri aktarımı yapılırken, standart olmayan özel bir protokol kullanılabilir. Ancak bu bilgisayar ile haberleşme yapılırken, bilgisayar tarafında koşan yazılım için daha fazla işlemin yapılmasını gerektirmektedir. Bu sebepten ötürü standartlar takip edildiğinde, hazır kütüphanelerin kullanımı ile geliştirme zamanı düşmektedir. Bunun ile ilgili bilgiler 6.2 bölümde aktarılacaktır.

6.2 Aktarım Protokolleri ve Tasarımları

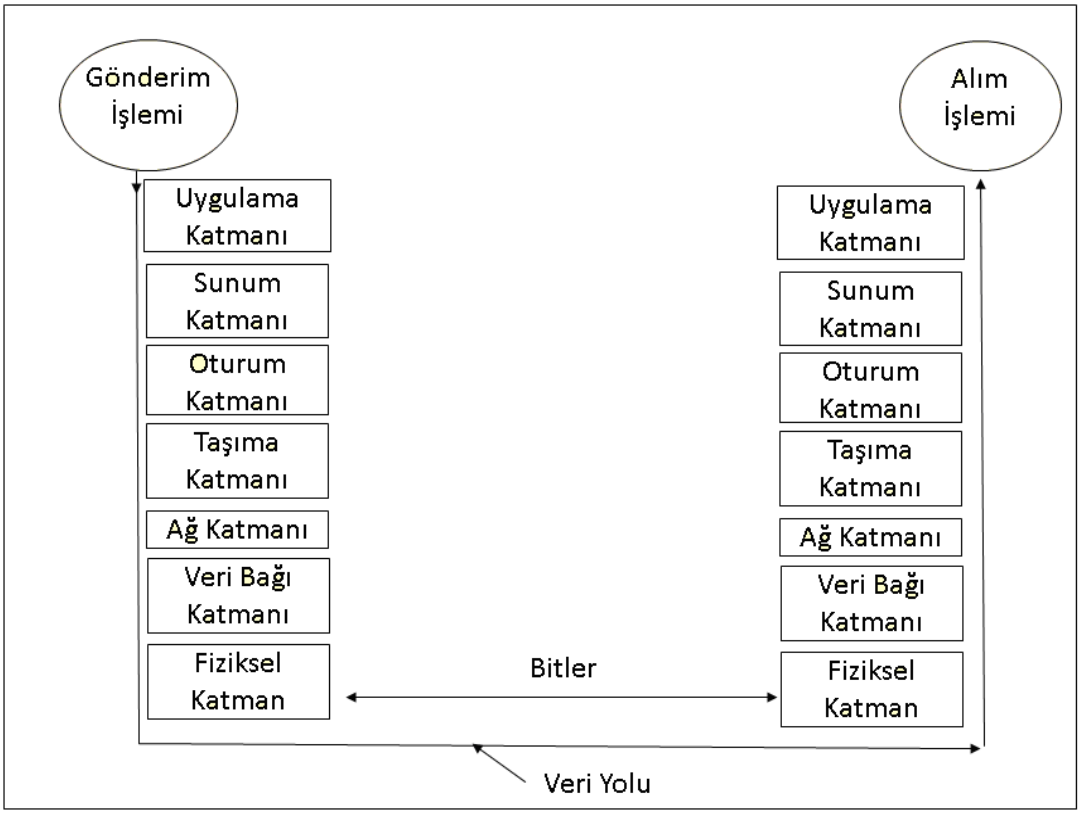
Geliştirilen sistem, OSI standartları takip edilerek tasarlanmıştır. Bu yüzden diğer platformlara kolay entegre edilebilir yapıdadır. Standart olmayan protokollerin kullanımı hedef cihazda daha fazla işlem yapılmasını gerektirir. Bu sebeplerden ötürü OSI standartlarında aktarım sağlanmıştır. Alt bölümlerde OSI modeli, kullanılan protokoller ve bu protokollerin çip üzerinde tasarımları hakkında bilgi verilecektir.

6.2.1 OSI Modeli

Uluslararası Standartlaştırma Örgütü (ISO) tarafından geliştirilmiş olan Open System Interconnection (OSI) modeli, cihazların kendi aralarında haberleşmelerinin standartlaştırılmasını sağlamaktadır. Kullanılan donanım ya da ağ özelliklerinin haberleşme için önemi olmamaktadır. Model, 7 katmandan oluşmaktadır. Bunlar sırasıyla fiziksel, veri bağı, ağ, taşıma, oturum, sunum ve uygulama katmanlarıdır. Her katman kendi ile ilgili olan görevi yaptıktan sonra verinin üst ya da alt katmana iletimini gerçekleştirirler. Katmanlara ayrılması her katmanın kendi içerisinde ayrı ayrı geliştirilebilmesi, düzenlenebilmesi, kısacası modülerlik kazandırmaktadır. [40,41] Katmanların kısaca açıklamaları aşağıda verilmiştir. Şekil 6.3'te OSI katmanları verilmiştir.

- Fiziksel Katman: Ağ üzerinde verinin fiziksel karakteristiğini tanımlamaktadır. Aktarımı yapılacak olan verinin, ortama uygun olarak dönüştürümünün nasıl yapılacağını belirtir. Bu ortam elektrik, ışık, radyo vb. sinyalleri içerebilir. Veriler 0 ve 1'ler şeklinde sinyaller olarak aktarılmaktadırlar. Gönderici, ortama sinyalleri aktarırken alıcı, cihazın fiziksel katmanında ortamdan sinyalleri alıp 0 ve 1'ler dizisine çevirmektedir.
- Veri Bağı Katmanı: Fiziksel ortama bağlanmak için gerekli olan işlemlerin yapıldığı katmandır. Ağdaki cihazların belirlenmesi, o esnada ortamı kullananın tespiti, kaynak ve hedef cihazlarının aynı frekansta çalışması ve fiziksel katmandan gelen verinin hata kontrolünü gerçekleştirmektedir. Fiziksel katman ve ağ katmanı arasında format dönüşümünü sağlamaktadır. Bu katmanda işlemler önemli bir ölçüde ağ kartının içerisinde gerçekleşmektedir.

- Ağ Katmanı: Ağa gönderilecek olan paketin, yönlendiriciler tarafından aktarılacağı adresin eklendiği katmandır. Bu katmanda veriler paketler halinde taşınmaktadır. Bir üst katman olan taşıma katmanının gönderdiği istekleri yanıtlar ve alt katman olan veri bağı katmanına iletmektedir. Paketin ağdaki gideceği en uygun yolu bulmaktadır. IP ve Arp protokolü bu katmanda çalışmaktadır.
- Taşıma Katmanı: Üst katmanlardan aktarılan verinin bölüm işlemini gerçekleştirir. Bölünmüş olan paketler ağ paketi boyutundadırlar. Taşıma katmanının üstündeki katmanlar, donanım ile ilgilenmeksizin veri ile ilgilenirler. Aynı durum alt katmanlarda da geçerlidir, veri ile ilgilenmeksizin donanım ile ilgilenirler. TCP, UDP vb… protokoller bu katmanda bulunmaktadırlar.
- Oturum Katmanı: Ağdaki cihazlar arasındaki iletişimin başlaması, kullanılması ve bitirilmesi işlemlerini gerçekleştirirler. Cihazın iki ve üzeri cihaz ile iletişim halinde olduğu durumlarda, veri aktarımının doğru cihaza sağlanmasını sağlar. Netbios, Sockets gibi protokoller bu katmandadır.
- Sunum Katmanı: Hedef cihaza gönderilecek olan paketin, hedef cihaz için anlaşılabilir olması için gerekli dönüşümü gerçekleştirir. Verinin formatı, filtrelenmesi, sıkıştırılması vb. işlemler bu katmanda tamamlandıktan sonra oturum katmanına gönderilir. Farklı uygulamalar, birbirlerinin verilerini bu şekilde kullanabilir hale gelmektedirler.
- Uygulama Katmanı: Katmanların en üstünde bulunmaktadır. Kullanıcıya en yakın olan katmandır. Cihazlardaki uygulamaların ağa erişimlerindeki arayüz görevini sağlar. HTTP, SMTP, POP, TTP vb… protokoller bu katmanda bulunmaktadırlar.



Şekil 6.3 OSI Katmanları

6.2.2 Haberleşme Protokolleri

Geliştirilen sistemde çok kanallı yapıların koşması nedeniyle yüksek bant genişliği gerekli olmaktadır. Aktarım için UDP veya TCP protokolü kullanılabilir. Ancak TCP protokolünde bağlantı kurulumu, aktarım kontrolü ve kapanış işlemleri için geçecek olan süre, ek maliyet getirecektir. Dolayısıyla UDP protokolünün getirmiş olduğu hız avantajı sebebi ile aktarım işlemlerinde UDP protokolü kullanılmıştır. Tasarımda ARP sorgusuna ihtiyaç duyulmaktadır. Paket gönderilecek olan hedef cihaz, paket gönderen cihazın ARP tablosunda yok ise, ARP sorgusu yapmalıdır. UDP ile veri transferi öncesinde ARP protokolü çalışmaktadır.

Geliştirilen sistem otomatik olarak arp yanıtı vermektedir. Bir diğer protokol ise DHCP protokolüdür. DHCP, sistemin ağa bağlandığında otomatik olarak ip alması, kendini tanıtması için gerekli bir protokoldür. Dolayısı ile UDP, ARP ve DHCP protokolleri tasarım için gerekli olmaktadır. Aşağıdaki alt başlıklarda bu protokoller hakkında bilgi verilecektir.

6.2.2.1 UDP Protokolü

UDP, hızlı aktarım yapabilmesi sebebi ile özellikle gerçek zamanlı görüntü/ses aktarımlarında tercih edilen bir protokoldür. Hızlı aktarmasının nedeni bağlantı kurma, kontrol ve kapatma gibi özellikleri barındırmamasıdır. Ancak bu özelliği paketin hedefe varmasını garanti etmemektedir. Bu eksikliği aşmak için ayrıca bir tasarıma ihtiyaç vardır. Bu yaklaşım gecikmeyi arttırmaktır. Paket yapısı Çizelge 6.2'de verildiği gibidir[38].

Çizelge 6.2 UDP Paket Yapısı

<table>
<tr><td colspan="5">Hedef Mac Adresi 0-3 Byte</td></tr>
<tr><td colspan="3">Hedef Mac Adresi 4-5 Byte</td><td colspan="2">Kaynak Mac Adresi 0-1 Byte</td></tr>
<tr><td colspan="3">Kaynak Mac Adresi 2-5 Byte</td><td colspan="2">Veri Uzunluğu</td></tr>
<tr><td>Versiyon</td><td>Başlık Uzunluğu</td><td>Servis Türü</td><td colspan="2">Toplam Uzunluk</td></tr>
<tr><td colspan="3">Kimlik</td><td>Bayraklar</td><td>Parça Numarası</td></tr>
<tr><td colspan="2">Yaşam Süresi</td><td>Protokol</td><td colspan="2">Başlık Sağlaması</td></tr>
<tr><td colspan="5">Kaynak IP Adresi</td></tr>
<tr><td colspan="5">Hedef IP Adresi</td></tr>
<tr><td colspan="4">IP Ayarları(Varsa)</td><td>Doldurma</td></tr>
<tr><td colspan="3">UDP Kaynak Port</td><td colspan="2">UDP Hedef Port</td></tr>
<tr><td colspan="3">UDP Mesaj Uzunluğu</td><td colspan="2">UDP Sağlaması</td></tr>
<tr><td colspan="5">Veri</td></tr>
<tr><td colspan="5">…</td></tr>
<tr><td colspan="5">Sağlama</td></tr>
</table>

Bir UDP paketinde birçok anlam taşıyan başlık vardır. Verinin iletimi, başlıklardaki bilgilere göre yönlendiriciler tarafından sağlanmaktadır. Tam bir paketin içermesi gereken başlıkların anlamları aşağıda açıklanmıştır.

- Hedef Mac Adresi: Hedef cihazın ağ arayüzüne atanmış olan 48 bitlik benzersiz tanımlayıcı anahtardır.
- Kaynak Mac Adresi: Gönderen cihazın ağ arayüzüne atanmış olan 48 bitlik benzersiz tanımlayıcı anahtardır.
- Versiyon: Gönderilen paketin IPv4 ya da Ipv6 olduğunu belirtir.
- Başlık Uzunluğu: Paket başlığının uzunluğunu belirtmektedir. En az 20 byte olabilir.
- Servis Türü: Bağlı olunan ağın isteyebileceği hizmetleri belirtmektedir. Öncelik, gecikme, veri akışı, güvenlik, verim gibi öncelikleri belirtmek mümkünüdür.
- Toplam Uzunluk: Paketin byte türünden büyüklüğünü bildirmektedir.
- Kimlik: Kaynak tarafından verilmiş numaradır. Paketlerin birleştirilmesi esnasında bu numaralar kullanılmaktadır.
- Bayraklar: Paketin parçalanmaya izninin olup olmadığı bilgisi, parçalanmış olup olmadığı bilgilerini içermektedir.
- Parça Numarası: Eğer Veri parçalanmış ise, parçalar içerisindeki yerini belirtmektedir.
- Yaşam Süresi: Bir paketin en fazla kaç adet ağ cihazından geçebileceğini belirten bilgidir. Bir paket en fazla, yaşam süresi sayısı kadar cihazdan geçebilir. Sonrasında hedefe ulaşmamış ise paket düşürülür. Bir paket en fazla 255 ağ cihazını geçecek şekilde ayarlanabilir.
- Protokol: Ulaşım katmanından hangi protokolün kullanılacağı bilgisini içermektedir.
- Başlık Sağlaması: Başlıkta hata varsa bu sağlama sayesinde tespit edilir. Hatalı paketler yok edilir.
- Kaynak IP Adresi: 32 bitlik kaynak IP adresini içermektedir.
- Hedef IP Adresi: 32 bitlik hedef IP adresini içermektedir.

- Kaynak Port Numarası: Hedeften bir cevap alınması gereken durumlarda, hedefin cevap verirken kullanması gereken port numarasını içermektedir. Alanın sıfır olması durumunda kaynağın port numarası bulunmadığı anlamına gelmektedir.
- Hedef Port Numarası: İletilecek olan paketin hedef cihazdaki varacağı port'un numarasını içermektedir. Hedef port numarasının bilinmediği durumlarda sıfır ile doldurulmaktadır.
- UDP Mesaj Uzunluğu: UDP paketinin içeresindeki kullanıcının oluşturduğu verinin uzunluğu bulunmaktadır. Bir UDP paketi teorik olarak en fazla 65.507 byte'dan oluşabilir.
- Sağlama: Veri ve başlıklarda hata kontrolü için kullanılmaktadır. Eğer gönderici tarafından hesaplanmamışsa tüm bitlere sıfır değeri atanır. Ipv6'da doldurulması zorunludur.

6.2.2.2 Arp Protokolü

Ağlarda veri alışverişi için donanımsal adresin bilinmesi gerekmektedir. Birçok cihaz, daha önce veri göndermemiş olduğu bir adrese veri gönderirken, önce karşı adreste birisinin olup olmadığını, varsa fiziksel adresini isteyen bir arp sorgusu gönderir. Bu işlem hedef cihazın ip'sinin bilindiği ancak fiziksel adresinin bilinmediği durumlarda yapılmaktadır. Cihazlar gönderdikleri istek doğrultusunda yanıt aldıklarında kendi Arp tablolarına, cihazın Mac adresi ve IP adresini yerleştirirler. Daha sonra göndereceksleri paketlerde bu tabloya bakıp, mevcut kayıt varsa tekrar sorgu göndermezler. Cihazlar bu protokolün yanıtını alamadıkları takdirde veri akışına başlamazlar. Dolayısıyla geliştirilen sistem, kendisine bir Arp sorgusu geldiğinde yanıtlamak durumundadır. Bu yüzden geliştirilen sisteme Arp sorgusunu yanıtlayacak tasarım sisteme dâhil edilmiştir. Aşağıdaki çizelgede arp paketinin içerdiği başlıklar ve açıklamaları yer almaktadır.[37]

Çizelge 6.3 ARP Paket Yapısı

<table>
<tr><td colspan="2">Donanım Türü</td><td>Protokol Türü</td></tr>
<tr><td>Donanım Adres Uzunluğu</td><td>Protokol Adres Uzunluğu</td><td>Operasyon Kodu</td></tr>
<tr><td colspan="3">Kaynak Donanım Adresi 0-3 Byte</td></tr>
<tr><td colspan="2">Kaynak Donanım Adresi 4-5 Byte</td><td>Kaynak Protokol Adresi 0-1 Byte</td></tr>
<tr><td colspan="2">Kaynak Protokol Adresi 2-3 Byte</td><td>Hedef Donanım Adresi 0-1 Byte</td></tr>
<tr><td colspan="3">Hedef Donanım Adresi 2-5 Byte</td></tr>
<tr><td colspan="3">Hedef Protokol Adresi 0-3 Byte</td></tr>
</table>

- Donanım Türü: Veri hattı katmanının ağ protokol tipini belirttiği alandır.
- Protokol Türü: Protokollerin bu alanda kullanabilecekleri numaralardır. Örneğin IPv4 için 0x0800 değerini almaktadır.
- Donanım Adres Uzunluğu: Donanım adresinin uzunluğunun byte türünden içermektedir. Ethernet adres uzunluğu 6 byte'dır.
- Protokol Adres Uzunluğu: Üst katman protokolündeki adresin byte türünden uzunluğunu içerir. IPv4 adres 4 byte, IPv6 adresin ise 16 byte'dır.
- Operasyon Kodu: Giden Arp paketinin istek ya da yanıt olduğunun belirtildiği operasyon kodunu içerir.
- Kaynak Donanım Adresi: Kaynak donanımın adresini içermektedir.
- Kaynak Protokol Adresi: Kaynak protokol adresini göstermektedir.
- Hedef Donanım Adresi: Hedef donanım adresini içermektedir. İstek içeren bir arp paketinde bu alan FF:FF:FF:FF:FF:FF ile doludur.
- Hedef Protokol Adresi: Hedef protokol adresini içermektedir.

6.2.2.3 DHCP Protokolü

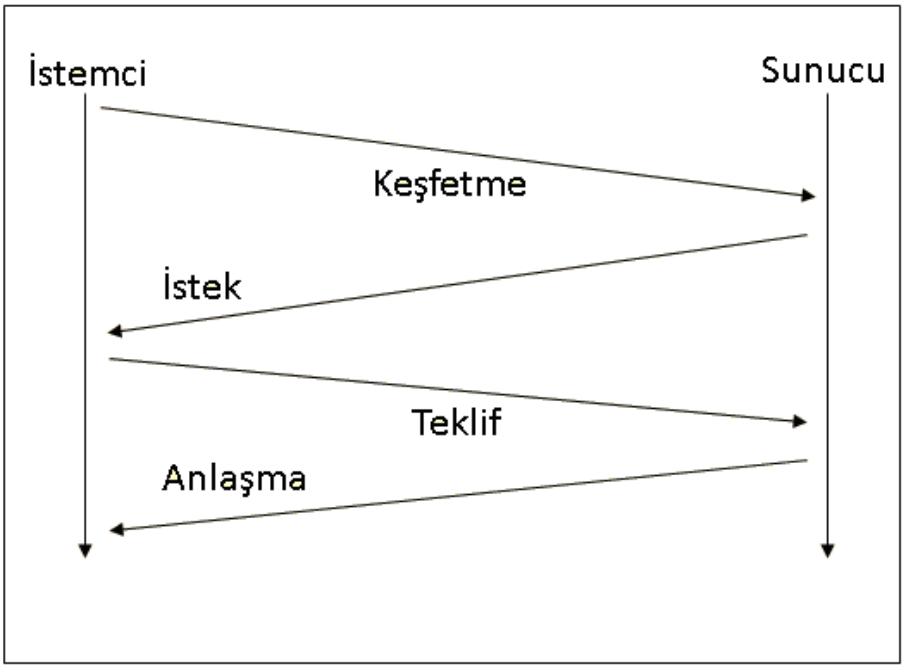
DHCP protokolü, ağ yönlendirici cihazların, ağa bağlanan cihazlar ile ip ataması yaparken anlaşmasını sağlamaktadır. Ağa bağlı olan her makineye otomatik olarak ip dağıtmak amacı ile geliştirilmiştir. Otomatik ip alabilmek için bu isteğin yanıtlanması zorunludur. 4 temel aşamadan oluşmaktadır. Bunlar; sunucu arama, teklif, istek ve anlaşmadır. Geliştirilen sistem yönlendirici cihazlar ile kullanılabilir olması için DHCP protokolü sisteme dâhil edilmiştir. Böylelikle sistemin yönlendirici üzerinden uzaktan erişimine imkân tanınmıştır. DHCP protokolünün birçok başlığı bulunmaktadır. Bunlar Çizelge 6.4'te verilmiştir.[37]

Çizelge 6.4 DHCP Paket Yapısı

Operasyon Kodu	Donanım Türü	Donanım Adres Uzunluğu	Atlama
Aktarım Kimliği			
Saniyeler		Bayraklar	
İstemci İp Adresi			
Kendi İp Adresi			
Sunucu İp Adresi			
Geçit İp Adresi			
İstemci Donanım Adresi			
Sunucu Adı			
Önyükleme Dosya Adı			
Ayarlar			

- Operasyon Kodu: DHCP mesajının türünü belirtmektedir. 1 ise istek, 2 ise yanıt anlamına gelmektedir.
- Donanım Türü: Ağda kullanılan donanım türünü belirtmektedir. Ethernet için 1 kodu bulunmaktadır.

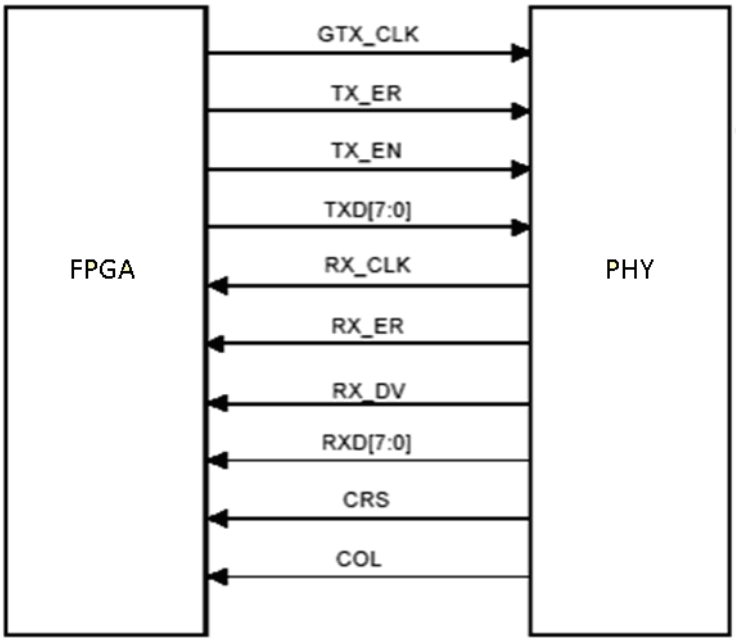
- Donanım Adres Uzunluğu: Donanım adresinin uzunluğunun belirtildiği alandır. Ethernet için 6 kullanılmaktadır.
- Atlama: Mesajın aktarılırken aktarılmış olan aktarıcı cihaz sayısı.
- Aktarım Kimliği: Sunuculardan gelen isteklerin istemciler tarafından karşılaştırılması için kullanılmaktadır.
- Saniyeler: İstemcinin, DHCP protokolünü işlemeye başladıktan sonra geçen süre.
- Bayraklar: Bir istemcinin ip adresinin bilinmediği durumlarda broadcast yapılarak istemcinin paketi alması sağlanır. Bu bit broadcast olup olmadığını belirtmektedir.
- İstemci İp Adresi: İstemcinin ip adresini içermektedir. Bu adres istemcinin ip adresini doğruladığı zaman geçerli olmaktadır.
- Kendi İp Adresi: Sunucunun doğrulamış olduğu istemci ip adresini içermektedir.
- Sunucu İp Adresi: Ayarlama sürecinde istemcinin kullanacağı ip adresini taşımaktadır.
- Geçit İp Adresi: Ağdaki yönlendirme görevini yapan cihazın ip adresini içermektedir.
- İstemci Donanım Adresi: İstemcinin donanım adresini içermektedir. Tanımlama ve iletişimde kullanılmaktadır.
- Sunucu Adı: Sunucunun ismini içermektedir.
- Önyükleme Dosya Adı: İsteğe bağlı olarak kullanılmaktadır. DHCPDISCOVER mesajı içerisinde geçmektedir.
- Ayarlar: DHCP işlemleri için bazı basit ayarları içermektedir.
- Şekil 6.4’te DHCP protokolüne göre istemci ve sunucu arasındaki veri trafiği görülmektedir.



Şekil 6.4 DHCP Protokol Akışı

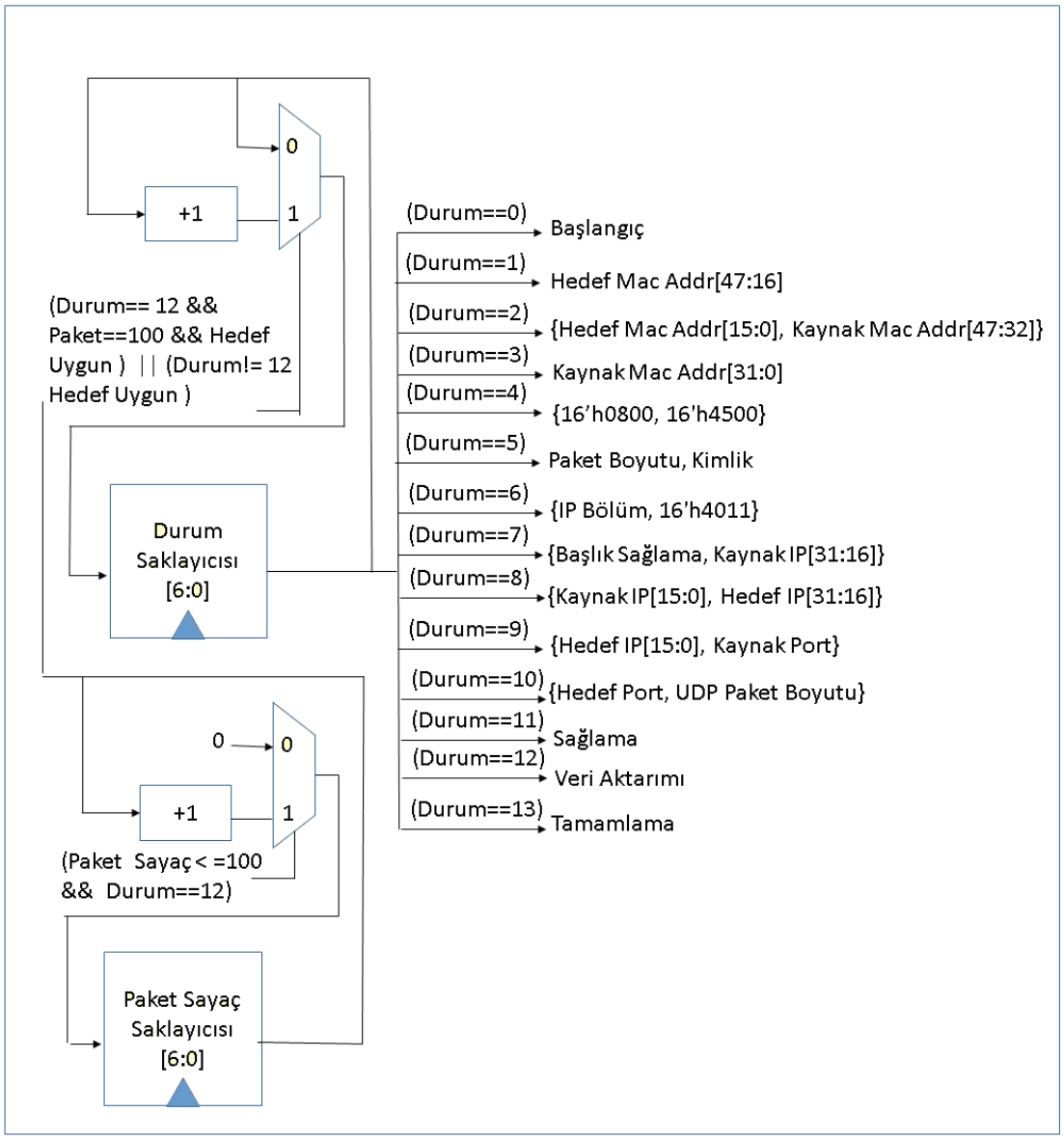
6.2.3 Haberleşme Tasarımı

Gigabit Ethernet, saniyede 1000Mbit aktarım hızına sahiptir. Bu aktarım saniyede 125MHz'lik 8 adet sinyalin aktarımı ile sağlanmaktadır. Şekil 6.5'te kullanılan Ethernet çipi ile FPGA arasındaki sinyaller verilmektedir.



Şekil 6.5 PHY FPGA Bağlantı Şeması[42]

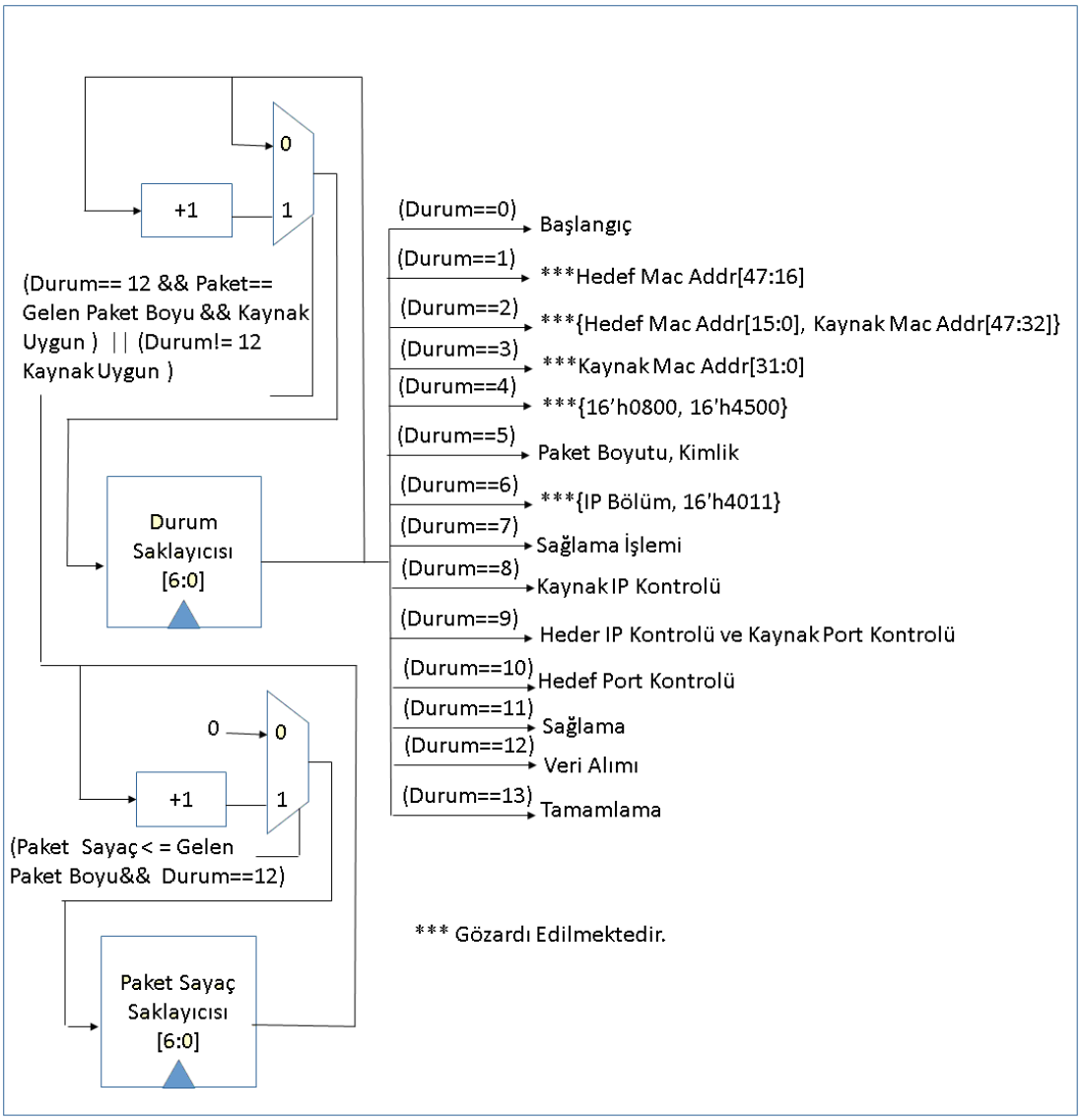
FPGA üzerinde işlenmiş verinin, hedef cihaz olan bilgisayara aktarımı için FPGA tarafında paket gönderme modülü, bilgisayar tarafında ise veri alma işlemi için paket alma modülleri tasarlanmıştır. Paket gönderme modülünün altında çalışan ARP ve DHCP modülleri, bu protokollere ihtiyaç olduğunda çalışmaktadırlar. Modüllerin tasarımları sonlu durum makinası ile tasarlanmıştır. Paket aktarımı sırasında gerçekleşmesi gereken işlemler, durum makinasının içerisinde bölünerek çalıştırılmaktadır.



Şekil 6.6 Paket Gönderim Modülü

Paket gönderme modülü 16 farklı durum ile tasarlanmıştır. Yapılan çalışmalarda UDP paketlerinin 512 byte ve üzerinde olması durumunda, paketin kaybolduğu veya eksik

geldiği gözlenmiştir. Dolayısıyla gönderimi yapılacak olan paketlerin 512 byte'ı aşmaması geremektedir. UDP paketinde 8, Ipv4 başlığı 20, toplam başlık byte toplamı 28 byte olmaktadır. Geriye 484 byte'lık veri gönderebilecek alan kalmaktadır. Şekil 6.6'da Paket Gönderim modülünün durum makinası diyagramı verilmiştir.



Şekil 6.7 Paket Alım Modülü

32 bitlik genişliğe sahip olan işlenmiş verilerin, alınması ve geri gönderilmesi, 100 adet verinin biriktirilmesi ile yapılmaktadır. 32*100 bit, yani 400 byte'lık veri paketleri oluşturulmaktadır. Bu verinin aktarımı UDP protokolü ile olduğu için paketlerin başında başlıklar bulunmaktadır. UDP için 28 byte genişliğinde başlık kullanıldığı için 400+28, 428 byte'lık paketler halinde aktarım yapılmaktadır. ARP ve DHCP protokollerini içeren

modüller, paket alım modülünden gelen verilere göre başlatılmaktadır. Önceki bölümlerde anlatılan protokollerin başlıklarına göre aşağıdaki şekle benzer bir aktarım yapılmaktadır. Şekil 6.7'de Paket Alım modülünün durum yapısı diyagramı verilmiştir.

Paket alım işlemi 16 farklı durumdan oluşmaktadır. Ağa bağlı olan FPGA'e, veri olmayan birçok paket gelebilmektedir. Bunun nedeni kullanılan işletim sisteminin ağ ararken gönderdiği mesajlar ya da ağa bağlı makinaların mesajları olabilir. Bunların filtrelenmesi gerekmektedir. İlki yapılan kontrol, gelen paketin UDP, ARP ya da DHCP paketi olup olmadığıdır. Bu paketlerin haricindekiler yok sayılmaktadır. Eğer paket bir ARP ya da DHCP paketi ise, gönderici modüle gerekli olan cevap gönderilir ve bu paketler yanıtlanır. Paket UDP paketi ise göndericinin ip adresi kontrol edilir. Paket istenen adresten geldiğinde işleme konulmaktadır.

6.3 Yazılımı Tasarımı

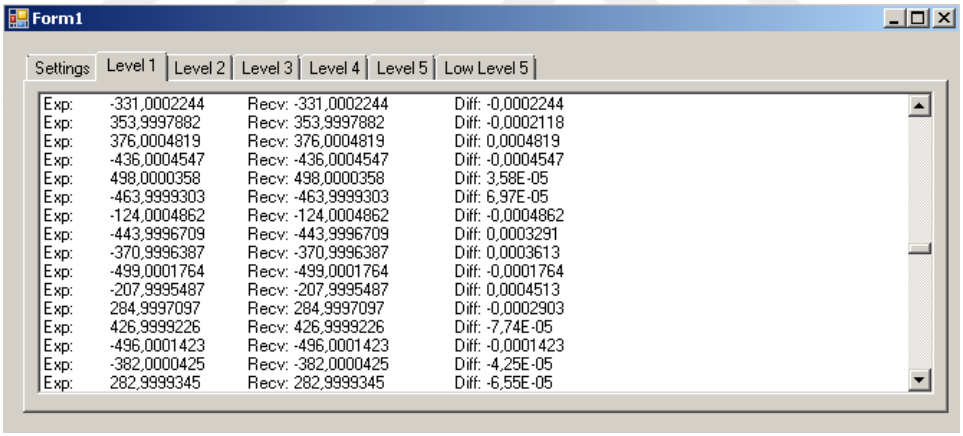
Tasarlanan sistem, hedef cihaz olan bilgisayar'a, işlenmiş olan verileri UDP paketleri olarak göndermektedir. FPGA üzerinden gelen verilerin bilgisayar tarafından alınabilmesi için yazılıma ihtiyaç duyulmaktadır. Bu yüzden .Net ortamında soket programlama kütüphaneleri kullanılarak bir uygulama geliştirilmiştir. .Net uygulaması text dosyasında bulunan verileri FPGA'e aktarmaktadır. Aynı anda da Ethernet üzerinden gelen verileri sürekli kontrol ederek, bu verileri toplamaktadır. Alınan veriler her kanal için ayrıştırılıp, farklı bölümlerde saklanmaktadır.

Şekil 6.8'de uygulamanın başlangıç arayüzü görülmektedir. Ana ekranda görülen combobox'ta hangi arayüz ile FPGA'e bağlanılacağı seçilmektedir. Bu yapı bilgisayara bağlı olan iletişim arayüzlerini listelemektedir. FPGA'e bağlı olan arayüz türü buradan seçimi yapılmaktadır. Port ve IP numaları yine bu arayüzden görülebilmektedir.

Başlat tuşuna basılmasıyla FPGA'e veri gönderimi ve FPGA'den veri alım işlemleri başlamaktadır. Alınan veriler arayüzde görülen seviyeler'e aktarılırlar. Her seviyenin verisi, kendine ait olan bölümde görülebilir. Ayrıca sonuçlar bilgisayar üzerinde hesaplanan değerler ile karşılaştırılmaktadır.

Şekil 6.8 Uygulama Giriş Arayüzü

Uygulamanın giriş arayüzü Şekil 6.8'de verilmektedir. Her bir seviye için farklı bölümler bulunmaktadır. Ayarlar bölümünde iletişimin hangi arayüzden ayarlanacağı, bilgisayar ve FPGA ileişim bilgileri bulunmaktadı. Başlat tuşu ile veri gönderimi ve aynı anda FPGA üzerinde hesaplanmış olan verilerin alınma işlemi gerçekleşmektedir. İstendiği takdirde FPGA'i resetlemek için reset butonu bulunmaktadır.



Şekil 6.9 Sonuçların Karşılaştırıldığı Ekran

Şekil 6.9'da ise seviye 1'deki alınan ve beklenen değerler gösterilmektedir. Aralarındaki farklar en sağda verilmektedir. Tüm seviyeler için beklenen, alınan ve fark formatında gösterimler yapılmaktadır.

BÖLÜM 7

SONUÇ VE ÖNERİLER

Tez çalışmaları kapsamında çeşitli platformlarda ÇAKDD yöntemi gerçeklenmiş olup elde edilen bulgular hız, alan, hata oranı ve güç tüketimi gibi veriler bu bölümde paylaşılmıştır.

7.1 Hız Karşılaştırması:

Çizelge 7.1'de görülebileceği üzere farklı platformlarda yapılan çalışmalarda elde edilen sonuçlar beklentiler ile uyumlu çıkmıştır. Paralel çalışmaya uygun olduğu için FPGA platformunda çok kanallı sistemler son derece verimli bir şekilde ortaya konabilmiştir. PIC ortamında elde edilen hızların gerçek zamanlı çalışmaya uygun olmadığı görülmüştür. Bilgisayar ortamında ise az kanallı sistemlerde gerçek zamanlı çalışılabilirken, çok kanallı sistemlerde ise seri çalışma prensibi dolayısıyla hızda düşüş meydana gelmektedir. Bunlara ek olarak çevre birimler ile olan iletişim ve işletim sisteminin farklı servislerinin de aynı anda çalışması uygulamayı yavaşlatarak performansı olumsuz şekilde etkilemektedir.

Çizelge 7.1 N-Kanal İçin Maksimum Frekans Değerleri

Ortam		**Azami Veri Giriş Hızı (K Örnek/Sn)**
PIC		0.5/N
MATLAB		100/N
C		6400/N
Raspberry		30/N
FPGA	Her Ağaç İçin Tek Toplayıcı&Çarpıcı	1840
	Her Kanal İçin Tek Toplayıcı&Çarpıcı	920
	Çok Kanal - Tek Toplayıcı&Çarpıcı	920/N

Çizelge 7.2 Bit genişliğine bağlı olarak kullanılabilecek azami saat frekansları (MHz)

32-Bit	18-Bit	12-Bit
73	111	115

7.2 Hata Oranları

Tez kapsamında, FPGA üzerindeki gerçeklemelerde elde edilen sonuçlar ve MATLAB'ten daha önce elde edilmiş olan sonuçlar kıyaslanmıştır. İki sonucun benzerliklerini değerlendirmek için işaret-fark oranı [43] olarak bilinen ölçütten yararlanılmıştır. İşaret-fark oranı;

$$\textit{İşaret-fark oranı} = \frac{\sum |\, X_{referans} - X_{gerçeklenen} \,|}{\sum |\, X_{referans} \,|} \times 100 \qquad (7.1)$$

şeklinde hesaplanır. Formülde $X_{referans}$, MATLAB'de elde edilmiş olan referans işaret, $X_{gerçeklenen}$ ise FPGA platformundan elde edilen işareti temsil etmektedir. Karşılaştırma sonuçları Çizelge 7.3'te raporlanmıştır.

Çizelge 7.3 İşaret-fark oranları

		Sabit-Nokta - FPGA	
Seviye	**Kayan-Nokta – Sabit-Nokta**	**32 Bit**	**18 Bit**
1	0,00008	0	1,25282
2	0,00008	0	1,45617
3	0,00009	0	1,56607
4	0,00012	0	1,31039
5	0,00014	0	1,62267

Çizelge 7.3'te, MATLAB simülasyon sonuçları ve sabit-nokta C gerçekleştiriminden elde edilen sonuçlar, FPGA platformunda 32-bit'lik sabit-nokta ve 18-bit'lik sabit-nokta aritmetiği ile aralarındaki fark oranları görünmektedir. Çizelgeye göre MATLAB simülasyonu ile yapılan çalışmada elde edilen sonuçlar arasında çok küçük bir fark vardır. Bu da FPGA platformunda kullanmış olduğumuz sabit-nokta ve MATLAB kullanılmış olan kayan-nokta aritmetiğinden kaynaklanmaktadır. C dilinde yapılan 32-

bit'lik gerçekleme ile FPGA'de yapılan 32-bit'lik gerçekleme arasındaki fark sıfırdır. Bu sonuç çalışmanın başarılı bir şekilde gerçekleştirildiğini göstermektedir. 32-bit'lik veri genişliği ve 18-bit veri kullanımı ile yapılan çalışmalar kıyaslandığında ise nicelemeden (quantization) kaynaklı küçük bir fark olduğu görülmüştür.

7.3 Güç Tüketimleri

Güç tüketiminden elde edilen sonuçlar, sentez aracının vermis olduğu sonuçlardır. Aşağıdaki figürlerde görülebileceği üzere çok kanallı mimariye ait güç tüketimleri daha yüksek görünmektedir. Ancak bunun sebebi içerisinde daha fazla kanal barındırmasıdır.

Çizelge 7.4 Güç Tüketim Değerleri

Power (W)	Yöntem-1	Yöntem-2	Yöntem-3
Mantıksal Eleman	0,172	0,174	0,169
Sinyal	0,037	0,021	0,019
Toplam	0,209	0,195	0,188

7.4 Alan Karşılaştırması

FPGA ortamında, farklı mimariler için yapılan geliştirmelerin alan verimliliği kıyaslaması Çizelge 7.5'te sunulmuştur. Beklentilere uygun olarak birinci yöntem olan her ağaç için bir çarpıcı ve bir toplayıcı yöntemi, alan açısından en verimsiz, tüm kanalların bir toplayıcı, bir çarpıcı ve bir LUT ile gerçeklendiği üçüncü mimari ise alan açısından en verimli mimari olmuştur.

Alan verimliliğinin daha da arttırılabilmesi, bu aşamadan sonra mimari olarak mümkün değildir. Ancak kullanılan verilerin bit genişliklerinin düşürülmesi ile alanı azaltmak mümkündür. Bu noktada dikkat edilmesi gereken husus, bit genişliğinin düşürülmesinin, elde edilen sonuçları etkilemesidir. Sonuçların hata toleransının düşük olması, bit genişliğinin düşürülmesine olanak sağlamaz. Fakat çalışılan problemin özelliğine bağlı olarak hata toleransı yüksek ise veri genişliğini düşürmek hem alanı azaltmaktadır hem de hızı arttırmaktadır. Hızın artmasındaki temel etken seçim

işlemlerinin (multiplexing) neden olduğu kritik yolun (critical path) gecikme miktarının azalmasıdır. Bu sayede 32bit'lik veriler ile çalışılırken kullanılan 73MHz'lik saat frekansı 18bit'lik verilerin kullanılmasıyla 111MHz'e çıkmıştır. Alan tarafında ise yaklaşık olarak %40'lık bir düşüş sağlanmıştır.

Çizelge 7.5 Alan Karşılaştırmaları

	12 Bit		**18 Bit**		**32 Bit**	
	Birleşik Fonksiyonlar	**Saklayıcı**	**Birleşik Fonksiyonlar**	**Saklayıcı**	**Birleşik Fonksiyonlar**	**Saklayıcı**
Yöntem-1	17366	18150	26453	26100	42228	43600
Yöntem-2	6022	15928	8056	22464	12522	38136
Yöntem-3	4275	13740	4704	20520	7563	33680
Yöntem-3 (Tek LUT)	1992	12063	2384	17776	4438	28802

Tez kapsamında yapılan çalışmalar ile ÇAKDD'nün alan verimliliği önplanda tutularak FPGA platformunda gerçeklemeleri yapılmıştır. İhtiyaca uygun mimari seçildikten sonra gerekli parametrelerin girilmesiyle kod üreten bir yazılım sayesinde araştırmacı ve geliştiricilerin çalışmalarında ÇAKDD'nü çok kolay ve hızlı bir şekilde kullanabilmesi sağlanmıştır.

Tez çalışmalarında elde edilen altyapı kullanılarak daha sonraki çalışmalarda öznitelik çıkarma, gürültü süzme ve temiz işaretlerin tekrar oluşturulması gibi işlemlerin çok basit ve hızlı şekilde yapılmasını sağlayacak bir platform oluşturulması hedeflenmektedir.

KAYNAKLAR

[1] Schalk G., McFarland D.J., Hinterberger T., Birbaumer N. ve Wolpaw J. R., (2004), "BCI2000: A General-Purpose Brain-Computer Interface (BCI) System", IEEE Transactıons On Bıomedıcal Engıneerıng, 51, 6, June 2004.

[2] Yazicioglu R.F., Kim S., Torfs T., Merken P. ve Hoof C.V., (2010), "A 30µW Analog Signal Processor ASIC for Biomedical Signal Monitoring", Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 124-125, Feb. 2010

[3] Huang P.T., Wu S.L., Huang Y.C., Chou L.-C., Huang T.C., Wang T.H., Lin Y.R., Cheng C.A., Shen W.W., Chuang C.T., Chen K.N., Chiou J.C., Hwang W. ve Tong H.M., (2014), "2.5D Heterogeneously Integrated Microsystem for High-Density Neural Sensing Applications" IEEE Transactions on Biomedical Circuits And Systems, 8:6.

[4] Kamboh A.M., Raetz M., Oweiss K.G. ve Mason A., (2007), "Area-Power Efficient VLSI Implementation of Multichannel DWT for Data Compression in Implantable Neuroprosthetics", IEEE Transactions on Biomedical Circuits and Systems, 1, 2, June 2007.

[5] Serbes G., Sakar C.O., Kahya Y.P. ve Aydin N., (2011), "Feature extraction using time-frequency/scale analysis and ensemble of feature sets for crackle detection." Conf Proc IEEE Eng Med Biol Soc 2011:3314–3317.

[6] Akay M., (1997), "Wavelet Applications in Medicine", IEEE Spectrum, 34, 5: 50-36.

[7] Mallat S., (1989), "A Theory for Multiresolution Signal Decomposition: The Wavelet Representation," IEEE Trans. Patt. Anal. Mach. Intell, 11:674–693.

[8] Aydin N., Marvasti F ve Markus H.S., (2004), "Embolic Doppler Ultrasound Signal Detection Using Discrete Wavelet Transform ". IEEE Trans Inf. Tech Biomed. 8, 2:182-190.

[9] Serbes G. ve Aydin N., (2010), "Denoising embolic Doppler ultrasound signals using Dual Tree Complex Discrete Wavelet Transform", Engineering in Medicine and Biology Society (EMBC), 2010 Annual International Conference of the IEEE, 1840 – 1843.

[10] Selesnick W., Baraniuk R.G. ve Kingsbury N.G., (2005), "The dual-tree complex wavelet transform", IEEE Signal Process. Mag. 22, 6:123-151.

[11] Serbes G., Sakar C.O., Kahya Y.P. ve Aydin N., (2013), "Pulmonary Crackle Detection using Time-Frequency and Time-Scale Analysis", Digital Signal Processing: A Review Journal, 23:3, 1012-1021.

[12] Serbes G., Sakar B.E., Aydin N. ve Gulcur H.O., (2014), "An Emboli Detection System Based on Dual Tree Complex Wavelet Transform", IFMBE Proceedings, 41:819-822.

[13] Benjamin B., Dornhege G., Krauledat M., Müller K., Kunzmann V., Losch F. ve Curio G., (2006), "The Berlin Brain–Computer Interface: EEG-Based Communication Without Subject Training, IEEE Transactions On Neural Systems And Rehabilitation Engineering", 14, 2:147-152.

[14] Oram M. W., Hatsopoulos N. G., Richmond B. J., ve Donoghue J. P., (2001), "Excess synchrony in motor cortical neurons provides redundant direction information with that from coarse temporal measures.", J. Neurophysiol, 86: 1700–1716.

[15] Mei-hua X., Zhang-jin C., Feng R. ve Yu-lan C., (2006), "Architecture Research and VLSI Implementation for Discrete Wavelet Packet Transform" High Density Microsystem Design and Packaging and Component Failure Analysis, 1-4.

[16] Nibouche M., Bouridane A., Nibouche O., Crookes D. ve Boussekta S., (2000), "Design and Fpga Implementation of Orthonormal Discrete Wavelet Transforms", 1: 312-315.

[17] Al-Haj A. M., (2003), "International Journal of Applied Science and Engineering" 1, 2: 160-171.

[18] Farahanı M. A. ve Eshghı M., (2007), "Implementing a New Architecture of Wavelet Packet Transform on FPGA", Proceedings of the 8th WSEAS International Conference on Acoustics & Music: Theory & Applications, Vancouver, Canada, June 19-21, 2007.

[19] AVCI A., (2006), "Wavelet Dönüşümü İle Doku Öznitelikleri Çıkarılan Görüntülerin Rezonans Algoritması Kullanılarak Bölütlenmesi" Yüksek Lisans Tezi.

[20] Stollnitz E.J., DeRose T.D. ve Salesin D.H., (1995), "Wavelets for Computer Graphics: A Primer", IEEE Computer Graphics and Applications, 15,3:75-85.

[21] Uytterhoeven G., (1999), "Wavelets: Software and Applications", Doktora Tezi, Haverlee.

[22] Lu C.S., Chung P.C. ve Chen C.F., (1997), "Unsupervised Texture Segmentation via Wavelet Transform", Pattern Recognition, 30, 5: 729-742.

[23] Ruiz L.A., Fdez-Sarria A. ve Recio J.A., (2004), "Texture Feature Extraction for Classification of Remote Sensing Data using Wavelet Decomposition: A Comparative Study", XXth ISPRS Congress.

[24] Busch C., (1997), "Wavelet Based Texture Segmentation of Multi-Modal Tomographic Images", Comput. & Graphics, 21, 3:347-358.

[25] Castleman R.K., (1996), “Digital Image Processing”, New Jersey, Upper Saddle River, Prentice-Hall.

[26] Wikipedia, Wavelet, http://en.wikipedia.org/wiki/Wavelet, 21 Mart 2006.

[27] Deepika S., (2003), “Efficient Implementations of Discrete Wavelet Transforms Using FPGAs”, Yüksek Lisans Tezi, Florida.

[28] Graps A., (1995), “An Introduction to Wavelets, IEEE Computational Science & Engineering”, 2, 2:50-61.

[29] Daubechies I., (1990), “The Wavelet Transform-Frequency Localization and Signal Analysis”, IEEE Transactions on Information Theory, 36, 5:961-1005.

[30] Kingsbury N., (1998), “The Dual-Tree Complex Wavelet Transform: A New Technique For Shift Invariance And Directional Filters” , IEEE Digital Signal Processing Workshop, DSP 98, Bryce Canyon, Agust 1998.

[31] Serbes G. ve Aydın N., (2009), “Doppler İşaretlerinin İşlenmesinde Yeni Bir Dalgacık Dönüşümü Yöntemi”, Biomedical Engineering Meeting (BIYOMUT), İzmir.

[32] Wikipedia, Kayan Nokta, https://tr.wikipedia.org/wiki/Kayan_nokta

[33] Küçük Ü., “Sayısal İşaret İşleyiciler ve Uygulamaları Ders Notları”.

[34] "1977 Data Catalog", Micro Electronics from General Instrument Corporation.

[35] Kingsbury N., (1998), “The Dual-Tree Complex Wavelet Transform: A New Technique For Shift Invariance And Directional Filters” , IEEE Digital Signal Processing Workshop, DSP 98, Bryce Canyon, Agust 1998.

[36] Levent V.E., (2015), “Gerçek Zamanlı Görüntü İşleme Uygulamaları İçin Fpga Tabanlı Bir Sistemin Gerçeklenmesi” Yüksek Lisans Tezi.

[37] Sanduja V., R. Patial , (2012), “Sobel Edge Detection using Parallel Architecture based on FPGA”, International Journal of Applied Information Systems, 3:20-24.

[38] Zhai X., Bensaali F. ve Ramalingam S., (2011). “Real-time license plate localisation on FPGA”, Computer Vision and Pattern Recognition Workshops (CVPRW), 20-25 June 2011, Kolorado, 14-19.

[39] Marvell, Gigabit Ethgernet Receiver, http://www.marvell.com/transceivers/assets/Marvell-Alaska-Ultra-88E1111-GbE.pdf, 20 Ocak 2015.

[40] Briscoe N., (2000), “Understanding the OSI 7-layer model”, PC Network Advisor, 120:13-14.

[41] İTÜ Bilgi İşlem Daire Başkanlığı, Ağ Destek, bidb.itu.edu.tr/seyirdefteri, 4 Temmuz 2014

[42] Opencores, Ethernet, http://opencores.org/project,ethernet_tri_mode, 14 Temmuz 2014.

[43] Serbes G., Gulcur H.O. ve Aydin N., (2016), “Directional Dual-tree Complex Wavelet Packet Transforms for Processing Quadrature Signals”, 54,2:295–313.

ÖZGEÇMİŞ

KİŞİSEL BİLGİLER

Adı Soyadı	:Ferhat CANBAY
Doğum Tarihi ve Yeri	:04.12.1982, İstanbul
Yabancı Dili	:İngilizce
E-posta	:ferhatcanbay@gmail.com

ÖĞRENİM DURUMU

Derece	**Alan**	**Okul/Üniversite**	**Mezuniyet Yılı**
Y. Lisans	Bilgisayar Müh.	Bahçeşehir Üniversitesi	2009
Lisans	Bilgisayar Müh.	Bahçeşehir Üniversitesi	2006
Lise	Sayısal	Adnan Menderes Anadolu Lisesi	2001

İŞ TECRÜBESİ

Yıl	**Firma/Kurum**	**Görevi**
2013-…	İstanbul Üniversitesi	Koordinatör
2010-2013	Arel Üniversitesi	Öğretim Görevlisi
2006-2010	Bahçeşehir Üniversitesi	Araştırma Görevlisi

YAYINLARI

Makale

1. Ugurdag H.F., Gören S. veCanbay F., (2010), "Gravitational Pose Estimation," Computers and Electrical Engineering, Elsevier, 36:1165-1180.

Bildiri

1. Canbay F., Levent V.E., Serbes G., Ugurdag, H.F., Gören S. ve Aydın N., (2016), "A Multi-Channel Real Time Implementation of Dual Tree Complex Wavelet Transform In Field Programmable Gate Arrays,", 15th IEEE International Conference on Bioinformatics and Bioengineering (MEDICON'2016), Paphos, CYPRUS.

2. Canbay F., Levent V.E., Serbes G., Ugurdag, H.F., Gören S. ve Aydın N., (2015), "An area efficient real time implementation of dual tree complex wavelet transform in field programmable gate arrays,", 15th IEEE International Conference on Bioinformatics and Bioengineering (BIBE'2015), Belgrade, Serbia.

3. Canbay F., Levent V.E., Serbes G., Aydın N. ve Gören S., (2015), "Field Programmable Gate Arrays Implementation of Dual Tree Complex Wavelet Transform," 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC'2015), Milan, Italy.

4. Canbay F., Serbes G., Gören S. ve Aydın N., (2013), "Çift-ağaç Karmaşık Dalgacık Dönüşümü'nün Gerçek Zamanlı Gerçekleştirimi", Otomatik Kontrol Ulusal Toplantısı (TOK'2013), Malatya.

5. Uğurdağ H.F., Gören S. ve Canbay F., (2008), "Correspondenceless Pose Estimation from a Single 2D Image using Classical Mechanics," IEEE International Symposium on Computer and Information Sciences, October 2008, Turkey.

Proje

1. Yıldız Teknik Üniversitesi Bilimsel Araştırma Projeleri Koordinatörlüğü, 2014-04-01-DOP03 numaralı proje