VHDL KULLANARAK OFDM GERÇEKLENMESİ

Ferdi KARA

Bülent Ecevit Üniversitesi
Fen Bilimleri Enstitüsü
Elektrik Elektronik Mühendisliği Anabilim Dalında
Yüksek Lisans Tezi
Olarak Hazırlanmıştır

ZONGULDAK
Haziran 2015

KABUL:

Ferdi KARA tarafından hazırlanan "VHDL KULLANARAK OFDM GERÇEKLENMESİ" başlıklı bu çalışma jürimiz tarafından değerlendirilerek, Bülent Ecevit Üniversitesi, Fen Bilimleri Enstitüsü, Elektrik Elektronik Mühendisliği Anabilim Dalında Yüksek Lisans Tezi olarak oybirliğiyle kabul edilmiştir. 12/06/2015

Başkan: Prof. Dr. Ertan ÖZTÜRK ……
Bülent Ecevit Üniversitesi

Üye : Yrd. Doç. Dr. Hakan KAYA ……
Bülent Ecevit Üniversitesi

Üye : Yrd. Doç. Dr. Fuat KARAKAYA ……
Niğde Üniversitesi

ONAY:

Yukarıdaki imzaların adı geçen öğretim üyelerine ait olduğunu onaylarım. .../.../2015

Prof. Dr. Kemal BÜYÜKGÜZEL
Fen Bilimleri Enstitüsü Müdürü

"Bu tezdeki tüm bilgilerin akademik kurallara ve etik ilkelere uygun olarak elde edildiğini ve sunulduğunu; ayrıca bu kuralların ve ilkelerin gerektirdiği şekilde, bu çalışmadan kaynaklanmayan bütün atıfları yaptığımı beyan ederim."

Ferdi KARA

ÖZET

Yüksek Lisans Tezi

VHDL KULLANARAK OFDM GERÇEKLENMESİ

Ferdi KARA

Bülent Ecevit Üniversitesi
Fen Bilimleri Enstitüsü
Elektrik Elektronik Mühendisliği Anabilim Dalı

Tez Danışmanı: Prof. Dr. Ertan ÖZTÜRK
Haziran 2015, 93 Sayfa

Bu çalışmada Dik Frekans Bölmeli Çoğullama (Orthogonal Frequency Division Multiplexing-OFDM) tekniğindeki temel blokların Alanda Programlanabilir Kapı Dizileri (Field Programmable Gate Array-FPGA) donanımı üzerine yüklenmek üzere VHDL donanım tanımlama dili kodları oluşturulmuştur.

OFDM mimarisinin alıcı ve verici kısımlarında bulunan modülasyon, Ters Ayrık Fourier Dönüşümü (TAFD), Ayrık Fourier Dönüşümü (AFD) ve demodülasyon bloklarının ayrı ayrı VHDL kodları yazılmıştır. Modülasyon ve demodülasyon blokları için "Quartus II 13.1" ile yapılan sentezleme sonucu oluşturulan makine kodları Altera firmasına ait DE2- 115 kartındaki 4CE11529C7 numaralı FPGA'ya yüklenmiştir.

Modülasyon için sayısal modülasyon tekniklerinden 16'lı Dördün Genlik Modülasyonu (16-DÖGM 16-ary Quadrature Amplitude Modulation, 16-QAM) seçilmiştir. TAFD/AFD blokları, Hızlı Fourier Dönüşümü (HFD, Fast Fourier Transform -FFT) algoritmasını kullanan Altera firmasının "MegaWizard Plug-IN" aracının sunduğu kütüphane ile gerçekleştirilmiştir.

ÖZET (devam ediyor)

OFDM mimarisinin giriş verisi olarak 640 kelime uzunluğundaki MATLAB ortamında oluşturulan rastgele veri dizisi kullanılmıştır. VHDL kodu yazılan her bir bloğun fonksiyonel testleri "Altera Modelsim 10.1" kullanılarak yapılmıştır. Her aşamada elde edilen veriler MATLAB da oluşturulan modellerle karşılaştırılmıştır.

OFDM mimarisinin verici ve alıcı olarak iki ana blok halinde yapılan fonksiyonel testinde verici kısmında giriş verisi olarak kullanılan 640 kelime uzunluğundaki veri dizisinin, alıcı kısmının çıkış verisi ile bire-bir örtüştüğü gözlenmiştir.

Anahtar Kelimeler: OFDM, FPGA, VHDL, DÖGM, AFD

Bilim Kodu:609.02.07

ABSTRACT

M.Sc. Thesis

IMPLEMENTATION OF OFDM BY USING VHDL

Ferdi KARA

Bülent Ecevit University
Graduate School of Natural and Applied Sciences
Department of Electrical and Electronics Engineering

Thesis Advisor: Prof. Ertan ÖZTÜRK
June 2015, 93 pages

In this work, the VHDL codes for the main blocks of Orthogonal Frequency Division Multiplexing (OFDM) technique are formed in order to implement on a Field Programmable Gate Array (FPGA).

The VHDL codes of each block on the transceiver of OFDM technique such as modulation, Inverse Discrete Fourier Transform (IDFT), Discrete Fourier Transform (DFT) and demodulation are composed separately. The machine codes (bit stream files) of the modulation and demodulation blocks obtained after synthesizing by using "Quartus II 13.1" are loaded into the FPGA with the number of 4CE11529C7 located on DE2- 115 Kit of Altera Corporation.

16-ary Quadrature Amplitude Modulation (16-QAM) is chosen as a baseband digital modulation technique. IDFT/DFT blocks are implemented by using "MegaWizard Plug-In" tool of Altera Corporation using Fast Fourier Transform algorithm for DFT.

ABSTRACT (continued)

A random array with the 640 word length for the input data of OFDM technique is created by using MATLAB. The functional tests of VHDL codes for each block are conducted over "Altera Modelsim 10.1". On each stage, the outputs of functional tests are compared with the results of MATLAB models.

It is observed that the input data for the transmitter of OFDM technique is the same with the output data of receiver.

Key Words: OFDM, FPGA, VHDL, QAM, DFT

Science Code: 609.02.07.

TEŞEKKÜR

Tez çalışmamın Danışmanı olan Prof. Dr. Ertan ÖZTÜRK'e çalışmamın her anında bana verdiği katkılardan ve desteklerden ötürü teşekkür ederim. Tez çalışması süresince katkılarını esirgemeyen değerli jüri üyeleri hocalarım Yrd. Doç. Dr. Fuat KARAKAYA ve Yrd. Doç. Dr. Hakan KAYA'ya da teşekkürü bir borç bilirim.

Tezin tüm aşamalarında bana olan desteklerini her zaman hissettiğim dostlarıma ve tüm mesai arkadaşlarıma teşekkür ederim.

Ayrıca, bu tez çalışmasını BİDEB-2210 Yurt İçi Yüksek Lisans Burs Programı kapsamında maddi olarak destekleyen TÜBİTAK'a teşekkürlerimi sunarım.

Son olarak, önce iyi insan olmanın önemli olduğunu bana aşılayarak beni yetiştiren anne, baba ve abime ve üzerimde emeği olan herkese sonsuz teşekkür ve şükranlarımı sunarım.

İÇİNDEKİLER

Sayfa

İÇİNDEKİLER (devam ediyor)

Sayfa

ŞEKİLLER DİZİNİ

ŞEKİLLER DİZİNİ (devam ediyor)

ÇİZELGELER DİZİNİ

EK AÇIKLAMALAR DİZİNİ

SİMGELER VE KISALTMALAR DİZİNİ

SİMGELER

B_c : Kanalın Evre Uyumlu Bant Genişliği

d : İletimdeki Gecikme

f_i : Alt Taşıyıcı Frekansı

f_c : Taşıyıcı Frekansı

g(n) : Haberleşme Kanalını Belirten Vektör

g(t) : Temel Bant Darbe Şekillendirici

h[n] : Doğrusal Zamanla Değişmez Ayrık Zamanlı Haberleşme Kanalı

h(t) : Haberleşme Kanalının Karmaşık Sönümleme Katsayısı

k : Bit Sayısı

L : Kanal Uzunluğu

M : Modülasyon Seviyesi

n(t) : TBGG'nin Örnek Bir Fonksiyonu

N : Alt Taşıyıcı Uzunluğu / TAFD Uzunluğu

r(t) : Alınan Sürekli Zaman Bilgi İşareti

s[n] : Ayrık Zamanlı OFDM Sembolü

s(t) : İletilen Sürekli Zaman Bilgi İşareti

$\tilde{s}[n]$: Çevrimsel Ön Ek Eklenmiş Ayrık Zamanlı OFDM Sembolü

S[k] : Temel Bant Modülasyon Sonucu Elde Edilen Karmaşık OFDM Sembolü

t : Zaman

T : Periyot

T_m : Çoklu Yol Yayılım Maksimum Gecikme Süresi

T_S : İletilen İşaretin Sembol Süresi

x[n] : Haberleşme Sistemine Giriş Vektörü

y[n] : Kanaldan Alınan Sayısal İşaret

$\alpha(t)$: Haberleşme Kanalının Karmaşık Sönümleme Katsayısının Zarfı

Δf : İki Tane Komşu Alt Taşıyıcı Arasındaki Frekans Mesafesi

$\theta(t)$: Haberleşme Kanalının Karmaşık Sönümleme Katsayısının Fazı

φ_k : Bitişik Alt Taşıyıcılar

$\phi(t)$: Temel Bant Dalga Formunun Fazı

KISALTMALAR

ADC : Analog to Digital Converter (Analogdan Sayısala Dönüştürücü)

AFD : Ayrık Fourier Dönüşümü

ASIC : Application Sprecific Integrated Circuit (Uygulamaya Özgü Entegre Devre)

CP : Cyclic Prefix (Dairesel Ön-ek)

CPLD : Complex Programmable Logic Device

DAB : Digital Audio Broadcasting (Sayısal Ses Yayıncılığı)

DAC : Digital to Analog Converter (Sayısaldan Analoğa Dönüştürücü)

DFT : Discrete Fourier Transform (Ayrık Fourier Dönüşümü)

DVB : Digital Video Broadcasting (Sayısal Video Yayıncılığı)

DRAM : Dynamic Random Access Memory (Dinamik Rasgele Okunabilir Bellek)

DSL : Digital Subscriber Line (Sayısal Abone Hattı)

DSP : Digital Signal Processors (Sayısal İşaret İşleyiciler)

FDM : Frequency Division Multiplexing (Frekans Bölmeli Çoğullama)

FFT : Fast Fourier Transform (Hızlı Fourier Dönüşümü)

FIR : Finite Impulse Response (Sınırlı Dürtü Cevabı)

FPGA : Field Programmable Gate Array (Alanda Programlanabilir Kapı Dizileri)

HDTV : High Definition Television (Yüksek Çözünürlüklü Televizyon)

IEEE : The Institue of Electrical and Electronic Engineers

IFFT : Inverse Fast Fourier Transform (Ters Hızlı Fourier Dönüşümü)

JTAG : Joint Test Action Group

LAN : Local Area Network (Yerel Alan Ağı)

LE : Logic Element (Mantık Elemanı/Hücresi)

LTE : Long Term Evolution (Uzun Dönem Evrim)

LUT : Look-Up Table (Doğruluk Tablosu)

MQAM : M'li Dördün Genlik Modülasyonu

MPSK : M'li Faz Kaydırmalı Anahtarlama

SİMGELER VE KISALTMALAR DİZİNİ (devam ediyor)

OFDM : Dik Frekans Bölmeli Çoğullama
QAM : Quadrature Amplitude Modulation (Dördün Genlik Modülasyonu)
PAM : Pulse Amplitude Modulation (Darbe Genlik Modülasyonu)
RAM : Random Access Memory (Rasgele Okunur Bellek)
ROM : Read Only Memory (Sadece Okunabilir Bellek)
QPSK : Quadrature Phase Shift Keying (Dördün Faz Kaymalı Anahtarlama)
SAK : Semboller Arası Karışım
SDRAM : Synchronous Dynamic Random Access Memory (Senkronize DRAM)
SRAM : Static Random Access Memory (Statik Rasgele Okunur Bellek)
TAFD : Ters Ayrık Fourier Dönüşümü
TBGG : Toplanır Beyaz Gauss Gürültülü Kanal
USB : Universal Serial Bus (Evrensel Seri Veri Hattı)
WIMAX : Worldwide Interoperability for Microwave Access

BÖLÜM 1

GİRİŞ

Kablosuz haberleşme alanında her geçen gün büyük gelişmeler yaşanmaktadır. Kullanıcıların talepleri bu gelişmelerin yaşanmasında büyük rol oynamaktadır. Kablosuz haberleşme sistemlerinin en temel amacı farklı verileri (ses, video, veri paketleri vb.) kablosuz ortamda mümkün olan en yüksek hızda ve en iyi kalitede iletmektir. İstenilen hızların ve kalitenin yakalanabilmesi adına yeni teknolojiler yeni standartlar sürekli olarak oluşturulmaktadır. Ülkemizde 6-7 yıldır kullanılan "3.nesil hücresel sistemler 6-12 Mbps hızlarına ulaşabilmektedir. Yakın zamanda kullanıma geçmesi beklenen 4.nesil sistemler ile ise bu hız 100-200 Mbps seviyelerine ulaşacaktır. Bu teknolojinin de uzun vadede yeterli olmayacağı öngörülmekte ve bu nedenle 2020 yılı için standartlaşması beklenen 5.nesil sistemler üzerine çalışmalar başlanmıştır. İletilmek istenilen verinin boyutu her geçen gün arttığı düşünülürse sektörün sürekli olarak kendini yenilemesi ve yeni teknolojiler üretme ihtiyacı olağan karşılanmalıdır. Kullanılmaya başlanan her yeni teknikle beraber kullanılan donanımların da değiştirilmesi gerekecektir. Kullanılan alt yapının (baz istasyonları vb.) değiştirilmesi son derece maliyetlidir. Bu maliyetin önüne geçmek için kullanılmaya başlanan teknolojinin altyapı değiştirilmeden sisteme entegre edilmesi gerekmektedir. Bu durumda en ideal çözüm farklı sistemlerin aynı altyapıyı kullanan yazılım tabanlı radyolar (software defined radio – SDR) üzerinde gerçeklenmesi olarak gözükmektedir (Goldsmith 2005).

Haberleşme gibi veri girişinin çok fazla olduğu ve bu verilerin hızlı bir şekilde işlenmesinin gerektiği sistemlerde Uygulamaya Özgü Entegre Devreler (Application Specific Integrated Circuits – ASIC) kullanılmaktadır. Ancak, ASIC üzerinde mevcut sisteme yeni bir modül eklemek veya yeni baştan tasarım yapmak zaman ve maliyet açısından oldukça verimsizdir (Şahin 2010). Bu nedenle paralel işlem yapma kabiliyetleri sayesinde ASIC'lerle aynı hızlara ulaşabilen Sahada Programlanabilir Kapı Dizileri (Field Programmable Gate Array –FPGA) kuvvetli bir seçenek olarak önümüze çıkmaktadırlar. FPGA'ların sahada programlanabilme

özellikleri sayesinde mevcut donanımın değiştirilebilmesi ya da mevcut donanıma yeni bir tasarım eklenebilmesi son derece avantajlı bir durumdur.

Haberleşme sistemlerinde iletim hızlarının arttırılması kullanılan bant genişliği ile de doğru orantılıdır. Dolayısıyla meşgul edilen bandın verimli kullanılması son derece önem arz etmektedir. Tarihsel gelişimi uzun sürse de popülerliği özellikle son yıllarda fazlaca artan Dik Frekans Bölmeli Çoğullama (Orthogonal Frequency Division Multplexing- OFDM) bant verimliliği yüksek olan bir yöntemdir. Bant verimliliğin yanı sıra özellikle frekans sönümlemeli kanallara karşı dirençli olması da OFDM'i diğer yöntemlere göre avantajlı kılmaktadır. Bir diğer önemli avantajı ise diğer çoğullama tekniklerine oranla alıcı karmaşıklığının daha az olmasıdır. Bu avantajlarından dolayı Uzun Dönem Evrim (Long Term Evolution –LTE), Mikrodalga Erişimi İçin Evrensel Uyumluluk (Worldwide Interoperability for Microwave Access WIMAX), Kablosuz Yerel Alan Ağları (Wireless Local Area Networks - WLAN), Sayısal Video Yayını (Digital Video Broadcasting – DVB) ve Asenkron Sayısal Abone Hattı (Asynchronous Digital Subscriber Line – ADSL) gibi birçok haberleşme sistemi için standartları oluşturulmuştur (Aktürk 2014).

1.1 ÇALIŞMANIN AMACI

Son yıllarda oldukça artan veri iletim ihtiyacının karşılanması adına 4G üzerine yapılan çalışmalar oldukça artmıştır. Akademide yapılan bu çalışmalar çoğunlukla teorik ve simülasyon tabanlı olmaktadırlar (Tunçkaya 2009). Teorisi ve simülasyonu gerçekleştirilen sistemlerin endüstri uygulamaları üniversitelerde çok fazla ilgi görmemektedir. Bu tezde hem donanım gerçeklenmesi esnasında ortaya çıkabilecek problemlerin ve sistem performansının incelenmesi hem de donanım becerisinin geliştirilmesi amaçlanmıştır.

Bu tez kapsamında belirtilen amaçlar doğrultusunda 4.nesil hücresel sistemlerin temelini oluşturan OFDM tekniğindeki temel alıcı verici bloklarının FPGA üzerinde gerçeklenmesini mümkün kılacak olan VHDL kodlarının oluşturulması hedeflenmiştir. FPGA'lar günümüzde özellikle işlem yükünün çok fazla olduğu görüntü işleme uygulamalarında kullanılmaktadırlar. Bunun yanı sıra haberleşme sistemlerinde de modülasyon ve demodülasyon bloklarının gerçeklenmesinde kullanılmaktadırlar.

FPGA tercih edilmesinin sebebi sistemin esnek yapıda tasarlanabilmesinin ve sisteme daha sonradan tasarımların eklenebilmesinin mümkün olmasıdır. Bu sayede sisteme getirilecek yenilikler sisteme entegre edilebilecek ve sistem hibrid yapıda tasarlanmış olacaktır. Ayrıca modülasyon çeşidi veya FFT uzunluğu gibi özelliklerin değiştirilmesi de mümkün olacaktır.

OFDM tekniğinde modülasyon çeşidi olarak 16-DÖGM (Dördün Genlik Modülasyonu – Quadrature Amplitude Modulation- QAM) seçilmiş ve alt taşıyıcı uzunluğu olarak 64-IFFT/FFT belirlenmiştir. DÖGM diğer sayısal modülasyon tekniklerine göre yüksek modülasyon seviyelerinde sembol hata performansı daha iyi olduğu için yeni nesil iletişim sistemlerinde tercih edilmektedir.

Modülasyon ve demodülasyon blokları Altera firmasının Cyclone IV ailesine ait 4CE11529C7 numaralı FPGA üzerinde denenmiştir. Sentezleme aracı olarak "Quartus II 13.1" kullanılmıştır. Bütün bloklar için fonksiyonel testler ise "Modelsim 10.1" kullanılarak gerçekleştirilmiştir. VHDL kullanılarak gerçekleştirilen tasarımlar MATLAB sonuçları ile karşılaştırılmıştır. Her bir bloğun ve tüm sistemin benzetim sonuçlarının MATLAB sonuçları ile birebir örtüştüğü görülmüştür. Sayı formatı olarak IEE 754 32-bit kayan noktalı sayı formatının tercih edilmesi sonuçların hassas biçimde alınmasında önemli bir rol oynamıştır.

1.2 LİTERATÜR ARAŞTIRMASI

OFDM tekniğinin kullanıldığı çalışmalar genel olarak MATLAB gibi araçlarla benzetim tabanlı olsa da, donanım gerçeklenmelerine de çokça rastlanılmaktadır. Donanım gerçeklenmeleri için en fazla kullanılan platformlar Sayısal İşaret İşleyiciler (Digital Signal Processors) ve FPGA'lardır.

FPGA tabanlı gerçekleştirilen OFDM sistemlerinde ortak amaç hız, kaynak kullanımı ve kullanılan algoritmaların karmaşıklığının azaltılması gibi optimizasyon çalışmaları olmuştur. Yapılan çalışmalarda OFDM tekniğinin işlem yükü en fazla olan birimi IFFT/FFT bloklarının gerçeklenmeleri üzerinde durulmuştur.

(Dick and Harris 2003)'de Xilinx firmasının MATLAB Simulink içerisinde bulunan ve IP-Core modüllerini kullanan şematik bloklar ile IFFT/FFT tasarımı gerçekleştirmişlerdir. Bu çalışmada 16-bit sayı formatı kullanılarak 64 noktalı IFFT/FFT modülleri başarı ile

tasarlanmıştır. Ancak 16-bit sayı formatının OFDM için istenilen hassasiyet açısından sorun teşkil edebileceği söylenebilir.

(Kaptan ve diğ. 2007)' de IEEE 754 32-bit kayan noktalı sayı formatını kullanarak esnek yapıda çalışabilen IFFT/FFT Kütüphanesi oluşturmuşlardır. Oluşturulan kütüphane hız açısından diğer oluşturulan kütüphanelere göre daha yavaş gözükse de kaynak kullanımının azlığı açısından öne çıkmaktadır. Sentezleme Virtex II- Pro üzerinde yapılmış ve ortalama 0.65µs sürmektedir.

IEEE 802.11.a kablosuz yerel ağları için temel bant OFDM vericisi Xilinx firmasının System Geneator aracı ile gerçeklenmiştir (Garcia and Complido 2005). FFT uzunluğu olarak 64 seçilmiş ve 16 uzunluklu dairesel ön ek kullanılarak tasarlanan sistem Virtex-II ailesinden XC2V3000 numaralı FPGA üzerinde gerçeklenmiştir. IFFT mimarisi Xilinx IP-Core kullanılarak gerçeklenmiş ve Radix-4 mimarisi kullanılmıştır. Sentezleme sonucunda 1678 Dilim Saklayıcı (Slice), 2353 d-mandalı (D-Flip Flop-FF), 2814 doğruluk tablosu (Look-Up Table –LUT) ve 12 Blok Rastgele Erişimli Bellek (Block Random Access Memory –BRAM) kullanılmıştır.

Temel bantta gerçeklenen sistemlerin RF bandında uygulamaları da mevcuttur. Francisco Martin Gutierrez 2009 yılındaki yüksek lisans çalışmasında Xilinx System Generator ile oluşturulan temel bant OFDM mimarisinin kablosuz ortamda analizlerini yapmıştır (Gutierrez 2009). 16 QAM ve 64-FFT kullanılarak gerçeklenen sistemde verileri kablosuz kanalda iletebilmek için üst dönüştürücü (upconverter) ve temel banda döndürmek için alt dönüştürücü (downconverter) kullanılmıştır. Yapılan analizler sonucunda Xilinx IP-Core ile oluşturulan IFFT/FFT bloklarının gecikmelere son derece hassas olduğu gözlenmiştir.

Kullanılan sayı formatları da sistem performansı açısından önemli bir etkendir. Sayı formatının uygun biçimde belirlenmesi kaynak kullanımı ve hız açısından son derece önemlidir (Şahin 2010). Bu çalışmada 802.16 WIMAX standardı için temel bant OFDM bloklarının farklı sayı formatları ile gerçeklenmeleri yapılarak kaynak kullanımları ve hızları karşılaştırılmıştır. Yapılan çalışmada en küçük işlem birimleri dâhil bütün bloklar için yeni VHDL kütüphaneleri oluşturularak firma bağımlılığı olmadan esnek yapıda tasarım gerçeklenmiştir. 16-bit ve 32-bit sabit noktalı sayı formatları kullanılarak 16 QAM ve 128 noktalı IFFT blokları gerçeklenmiştir. Virtex-5 ailesinin XC5VLX110T numaralı FPGA'sı

üzerinde yapılan sentezlemede 9335 adet Dilim Saklayıcı, 14290 FF ve 42429 LUT kullanılmıştır.

Donanım tanımlama dilleri kullanılarak yapılan çalışmalar kadar firmaların "HDL-Tasarlama" araçları ile yapılan çalışmalar da bulunmaktadır. (Mohamad 2013)' de Mentor Graphics firmasının HDL-Tasarlama aracını kullanarak temel bant bir OFDM mimarisinin prototipinin oluşturulabileceğini göstermiştir.

1.3 TEZ PLANI

Tez aşağıda özetleri verilen 6 bölümden oluşmaktadır.

Birinci bölümde haberleşme sistemlerine genel bir bakış verilmiş ve neden OFDM tekniği ve FPGA'nın teze konu olduğuna dair bilgiler sunulmuştur. Çalışmanın amacından bahsedilerek, literatürde yapılan çalışmalar ve tezin planı aktarılmıştır.

İkinci bölümde FPGA'ların genel özelliklerine, geleneksel işlemcilere göre avantajlarına ve kullanım alanlarına değinilerek mimarileri hakkında detaylı bilgiler sunulmuştur. Çalışmada kullanılan FPGA tanıtılmıştır. Ayrıca sayı sistemleri ve kullanılan sayı formatı tanıtılmıştır.

Üçüncü bölümde VHDL ile tasarım genel hatlarıyla çizilmiş ve VHDL kodlamada kullanılan en yaygın yapılar hakkında bilgi verilmiştir.

Dördüncü bölümde teze konu olan OFDM yapısının avantajlarını kavrayabilmek adına kablosuz kanal kısaca tanıtılmıştır. Daha sonra OFDM mimarisinin gelişim süreci anlatılarak OFDM sinyal yapısı ve blok diyagramı gösterilmiştir. OFDM blok diyagramında bulunan bütün blokların matematiksel ifadeleri detaylı şekilde açıklanmıştır.

Beşinci bölümde OFDM mimarisinde gerçeklenen bloklar ve tasarım aşamaları detaylı olarak anlatılmıştır. Gerçeklenen her bir modülün benzetim sonuçlarına, donanım kullanımlarına, devre şemalarına ve MATLAB sonuçları ile karşılaştırılmalarına ait görüntüler verilerek analizler yapılmıştır.

Altıncı bölümde ise yapılan çalışmadan elde edilen sonuçlar değerlendirilerek günümüz sistemlerinde kullanılması durumunda oluşacak avantajlar ve dezavantajlar ele alınmıştır. Ayrıca ileriye dönük yapılabilecek çalışmalardan bahsedilerek öneriler sunulmuştur.

BÖLÜM 2

KULLANILAN PLATFORM

Haberleşme sistemlerinde bulunan sayısal işaret işleme algoritmaları işlem yoğunluğu oldukça fazla olan tekniklerdir. OFDM mimarisinde bulunan TAFD ve AFD bunlara örnek olarak gösterilebilir. Bu yüzden mimarinin üzerinde gerçekleneceği gömülü sistem platformunun performansının yüksek olması gereklidir. Bu tez çalışmasında istenilen performans değerlerini karşılayabilmek adına FPGA kullanılmıştır. Ayrıca gerçekleştirilecek işlemler sırasında sonuçların hassas olarak elde edilmesi son derece önemlidir. Bu nedenle gömülü sistem üzerinde kullanılacak sayı formatı da doğru seçilmelidir (Şahin vd. 2008).

Bu bölümde gömülü sistem platformu olarak seçilen FPGA'ların genel olarak sınıflandırılması, mimarisi ve işlem kapasiteleri üzerinde durularak tez çalışmasında kullanılan FPGA tanıtılacaktır. Son olarak da sayı sistemleri hakkında bilgi verilerek seçilen sayı formatı hakkında bilgiler verilecektir.

2.1 FPGA'LARA GENEL BAKIŞ

FPGA "Sahada Programlanabilir Kapı Dizileri" anlamında İngilizce "Field Programmable Gate Array" sözcüklerinin kısaltmasıdır.

FPGA'ların belki de en önemli özellikleri sayısal(dijital) olarak yapılabilen tüm tasarımların tek bir entegre üzerinde gerçekleştirilebilmesine olanak sağlamalarıdır. Paralel işlem yapabilme kabiliyeti başta olmak üzere, sahada programlanabilme, tasarımların test edilip doğrulanabilmesi ve istenildiği durumda içerisine işlemci dahi gömülebilme özellikleri, FPGA'ları günümüzde popüler kılan yönleridir (Sarıtaş ve Karataş 2013).

Birbirleriyle neden-sonuç ilişkisi olmayan işlemlerin paralel olarak gerçekleştirilebilme kabiliyeti, özellikle görüntü işleme ve sinyal işleme gibi yoğun bilgi girişi olan ve elde edilen

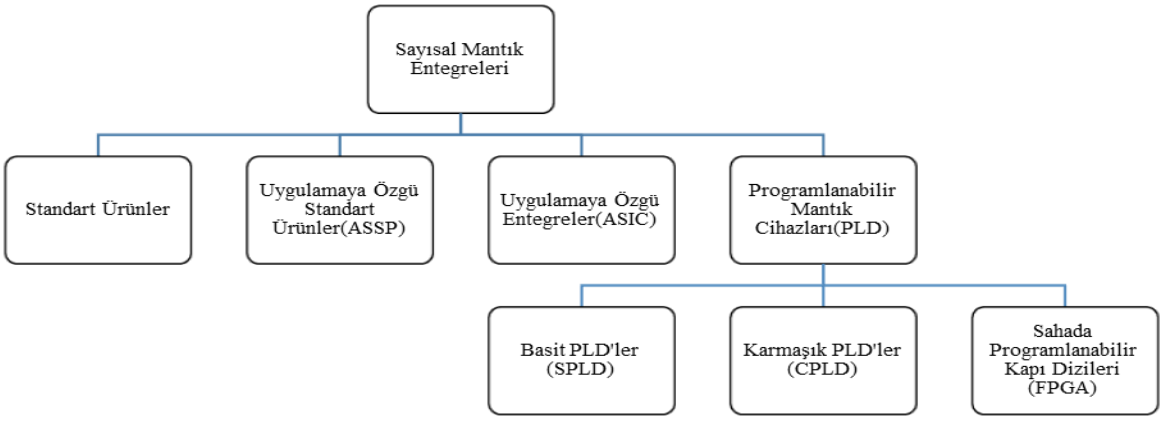
verinin kısa sürede işlenmesi gibi durumlar için FPGA'ları neredeyse rakipsiz yapmaktadır. Bu özelliği ile FPGA'lar geleneksel bilgisayar mimarilerine göre düşük frekanslarda çalışılmasına müsaade etmektedirler. Örneğin, geleneksel mimarilerde GHz'ler mertebesinde yakalanan işlem kapasitesi FPGA'larda 100-200 MHz frekanslarında yakalanabilmektedir (Çavuşlu ve Kösten 2015).

FPGA'ların bir diğer büyük avantajı ise düşük güç tüketimidir. Bu özelliği ise başta uzay ve havacılık sektörü gibi enerji kaynaklarının kısıtlı olduğu ve gerçek zamanlı işlenmesi gereken fazlaca verinin olduğu alanlarda FPGA'ları vazgeçilmez bir seçenek kılmaktadır.

Sahada programlanabilme özellikleri ile tasarımcıya son derece esneklik sağlayan FPGA'lar, bu özellikleri sayesinde de özellikle haberleşme sektörü gibi sürekli gelişmelerin yaşandığı, yeniliklerin getirildiği alanlarda da büyük bir kullanım oranına sahiptirler. Getirilen yenilikler, FPGA'nın donanım üzerinden sökülmeden tekrar programlanmasıyla donanıma aktarılabilmektedir.

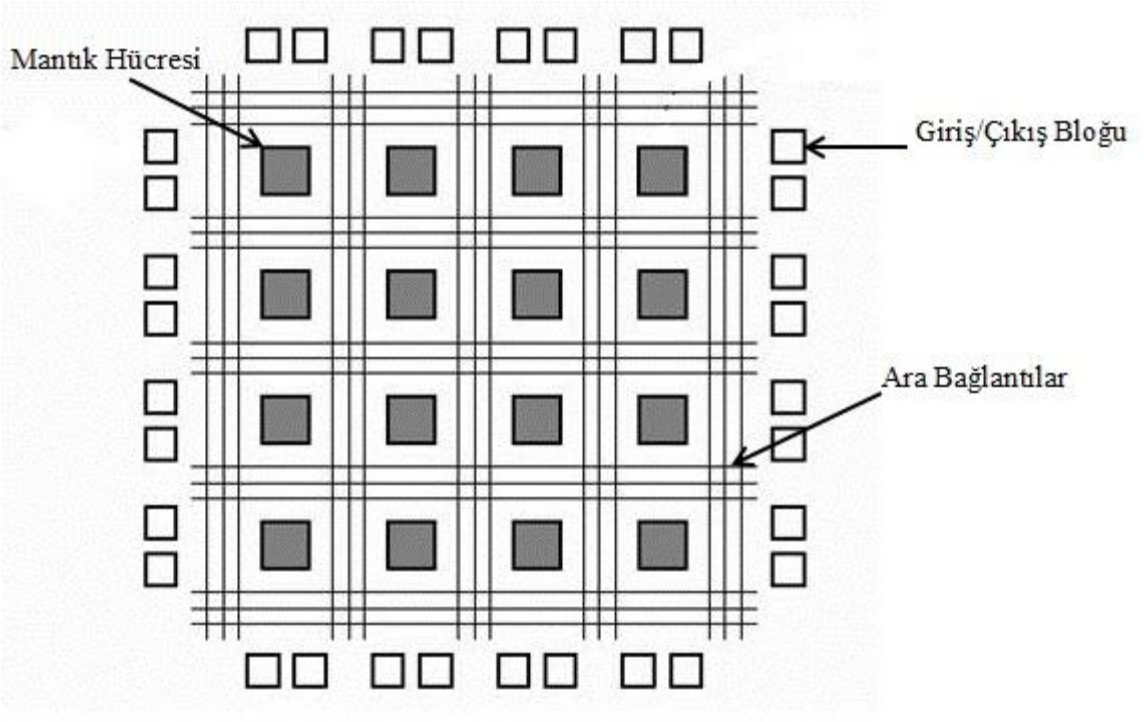
2.2 FPGA MİMARİSİ

FPGA'lar yapısı itibarıyla Sayısal Mantık Entegreleri ailesinin Programlanabilir Mantık Cihazları (Programmable Logic Device-PLD) sınıfının bir üyesi olarak gösterilse de içlerinde bulundurdukları elektronik elemanlar açısından diğer mantık cihazlarından oldukça farklılık göstermektedirler. Sayısal Mantık Entegrelerinin sınıflandırılması ve bu şema da FPGA'ların nerede bulundukları verilmiştir (Şekil 2.1).



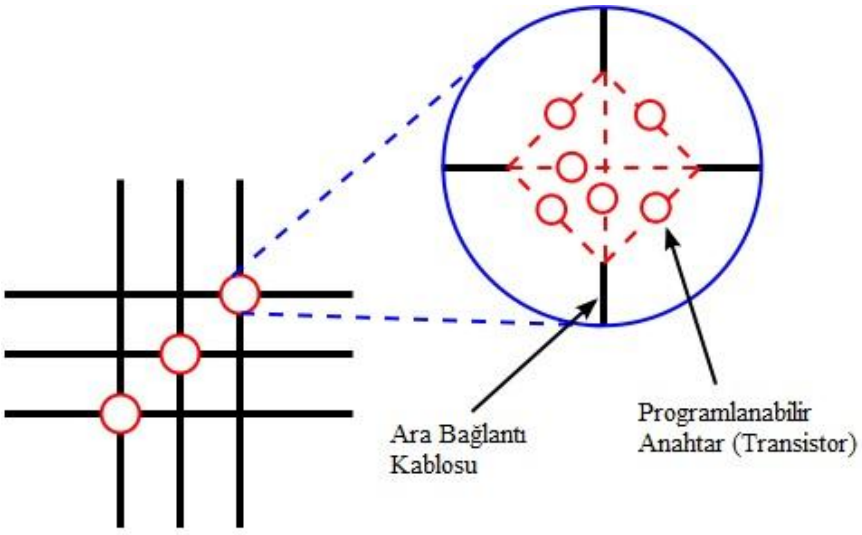
Şekil 2.1 Sayısal mantık entegrelerinin sınıflandırılması.

FPGA'lar diğer programlanabilir mantık cihazları gibi kapı seviyesinde (gate level) elemanlar içermezler (AND ve OR kapıları gibi) .Temel olarak FPGA'lar üç temel kaynaktan oluşurlar: mantık hücreleri/blokları, dış dünya ile bağlantı ve programlama pinlerini içeren Giriş/Çıkış blokları (I/O blocks) ve mantık hücreleri birbirine bağlayan ara bağlantı kabloları (Şekil 2.2).



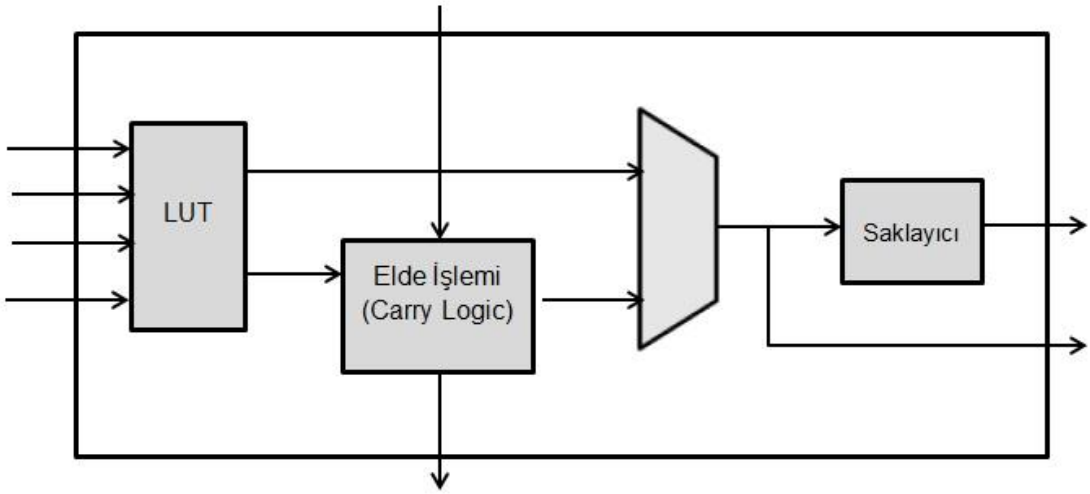
Şekil 2.2 FPGA mimarisi (URL-1 2013).

Mantık hücreleri iki boyutlu dizi şeklinde konumlandırılmışlardır ve ara bağlantı kabloları ise satır ve sütün şeklinde olan bu mantık hücreleri arasındaki bağlantı rotalarını belirlemek amacıyla yatay ve dikey olarak yerleştirilmişlerdir. Ara bağlantıların kesiştiği noktalarda programlanabilir anahtarlar bulunmaktadır. Her bağlantı noktasında bulunan anahtar yapısı 6 adet transistörden oluşmaktadır (Şekil 2.3). Ara bağlantıların kesişim noktalarında bulunan anahtarlar sayesinde mantık hücreleri birçok farklı şekilde birbirilerine bağlanarak sayısız mantık devresi kombinasyonları oluşturulabilmektedir. Bu bağlantılarla oluşturulan mantık devreleri binlerce mantık kapıları kullanılarak oluşturulabilecek olan devrelere eşdeğerdirler.



Şekil 2.3 Ara bağlantı matrisinin gösterimi (URL-2 2013).

FPGA'ların en temel yapı birimi olan mantık hücrelerinin yapıları üreticiden üreticiye hatta üreticinin farklı modelleri arasında bile değişiklik göstermekle beraber, bir mantık hücresi en yaygın kullanım olarak bir Doğruluk Tablosu (Look-Up Table –LUT), bir Çoklayıcı (Multiplexer –MUX) ve bir D mandalı (D Flip-Flop) saklayıcısından oluşmaktadır. Her hücre 0 veya 1'lerden oluşan bir mantık değeri (logic value) tutmakta ve çıkışa aktarmaktadır. Bu değerlerin sayısı LUT giriş sayısı arttırılarak arttırılabilir. Bizim kullandığımız FPGA'nın üreticisi olan ALTERA firmasına ait mantık elemanı (Logic Element- LE) 4 girişi olan LUT, bir adet tam-toplayıcı (full-adder), bir adet MUX ve bir adet saklayıcıdan (register) oluşmaktadır (Şekil 2.4).



Şekil 2.4 Mantık Elemanı (Logic Element-LE) Yapısı (Altera Corporation 2014).

FPGA içerisinde ayrıca birçok çarpma işleminin paralel olarak yapılabildiği Sayısal İşaret İşleme (Digital Signal Processing –DSP) bloklarının yanı sıra giriş saat bilgisine göre farklı frekans ve fazlarda saat sinyali üretebilen Faz Kilitlemeli Döngü (Phase Lock Loop –PLL) modülleri de bulunmaktadır. Bu modüller tasarlanan devre için gerekli olan osilatör sayısını ve kullanılan saat pini sayısını önemli ölçüde azaltmaktadırlar.

FPGA'ların mantık hücrelerindeki veriler uçucu (volatile) olmasından dolayı donanımı besleyen güç kesildiğinde saklanan bütün veriler kaybolacaktır. Bu nedenle FPGA'lar güç her uygulandığında tekrar programlanmak zorundadırlar. Bu işlemin yapılabilmesi için de FPGA'nın bulunduğu devre üzerinde programlama için gerekli verileri tutan Elektriksel Olarak Silinip Programlanabilen Sadece Okunan Hafıza (Electirically Erasable Programmable Read Only Memory –EEPROM) gibi küçük bir hafıza biriminin bulundurulması gerekmektedir. Böylece sisteme güç her uygulandığında veriler EEPROM'dan okunup tekrar programlama yapılabilecektir (Sarıtaş ve Karataş 2013).

FPGA'ların programlanması şematik tasarım ve Donanım Programlama Dilleri (Hardware Description Language) ile yapılmaktadır. En yaygın olarak kullanılan programlama dilleri Verilog ve VHDL'dir. Proje kapsamında kullanılan VHDL bir sonraki bölümde detaylı olarak anlatılacaktır.

2.3 GELİŞTİRME VE EĞİTİM KİTİ

Altera firmasının Geliştirme ve Eğitim Seti olarak sunduğu ailenin bir üyesi olan DE2-115 kartı hem temin kolaylığı hem de maliyeti açısından özellikle üniversite araştırmalarında sıklıkla kullanılan bir FPGA platformudur.

DE2-115 kartı Altera Cyclone IV ailesine ait 4CE11529C7 kodlu FPGA'yı barındırmaktadır. Geliştirme kartı üzerinde bulunan Giriş/Çıkış birimlerinin çeşitliliği sayesinde kullanıldığı uygulamalar oldukça fazladır (Terasic Technologies 2013). Üzerinde bulunan Giriş/Çıkış birimleri ve sayıları aşağıda verilmiştir:

1 adet Altera seri konfigürasyon aygıtı
1 adet USB Blaster programlayıcı
1 adet 2MB SRAM
2 adet 64MB SDRAM

1 adet 8MB Flash bellek

1 adet SD kart yuvası

4 adet buton

18 adet tek yönlü anahtar

18 adet kırmızı LED

9 adet yeşil LED

1 adet 50 MHz osilatör

1 adet 24-bit CODEC

1 adet 8-bit VGA DAC

1 adet TV kod çözücü

2 adet Gigabit Ethernet fiziksel katman konektörleri

1'er adet A ve B tipi USB konektörleri

1 adet RS-232 seri alıcı-verici konektörü

1 adet PS/2 fare/klavye konektörü

1 adet kızılötesi alıcı

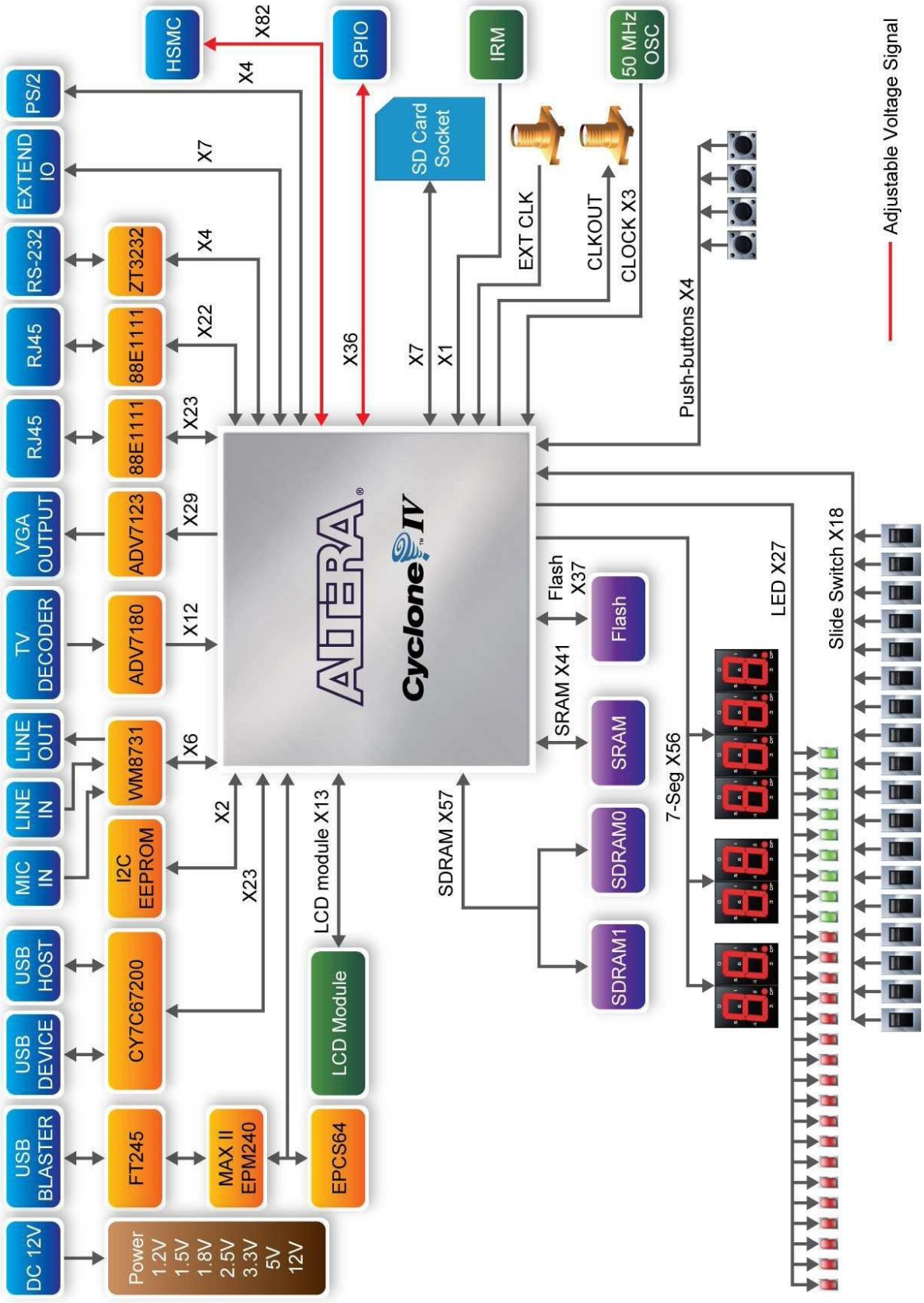
2 adet SMA konektörü (harici saat giriş/çıkış)

1 adet 40-pin genel giriş/çıkış (diyot korumalı)

1 adet Yüksek Hızlı Ara kart Konektörü (High Speed Mezzaine Card- HSMC)

1 adet 16x2 LCD

Geliştirme kartının üzerinde bulunan FPGA'nın ve Giriş/Çıkış bloklarının konumunu ve bağlantılarını gösteren blok diyagramı Şekil 2.5'te verilmiştir.



Şekil 2.5 DE2-115 geliştirme kartının blok diyagramı (Terasic Technologies 2013).

Geliştirme kartı üzerinde bulunan FPGA'nın içerdiği kaynaklar Çizelge 2.1'de verilmiştir.

Çizelge 2.1 DE2-115 Geliştirme kartı üzerinde bulunan FPGA'nın içerdiği kaynaklar.

4CE115 NOLU FPGA NIN İÇERDİĞİ KAYNAKLAR	
Cinsi	**Adet**
Mantık Elemanı (LEs)	114,480
Gömülü Hafıza(Embedded Memory) (Kbits)	3,888
Gömülü 18x18 Çarpıcılar	266
Genel-kullanım PLL	4
Global Saat Ağları	20
Kullanıcı G/Ç Kümesi	8
Maksimum G/Ç pini	528

Mantık elemanın ne olduğu daha önceden ayrıntılı bir şekilde açıklanmıştır. Gömülü hafıza birimleri ise tasarımda saklayıcı ve/veya hafıza birimlerine ihtiyaç duyulduğunda mantık elemanlardan oluşturmak yerine kullanıma sunulmuş hazır saklayıcılardır. Burada 18x18 çarpıcılar olarak adlandırılan kaynaklar, tasarımda özellikle çarpma, bölme, üs alma ve kök alma gibi işlem yoğunluğu olan operatörleri gerçekleştirme amacıyla kullanılmak üzere FPGA'nın içerisinde bulunan Sayısal İşaret İşleyici (DSP) bloklarıdır. 18x18 demek 18 bitlik işlemler için kullanıldığını ifade etmektedir. Toplamda bulunan 266 adet 18x18 çarpıcılar 532 adet 9x9 çarpıcı olarak da kullanılabilmektedirler. PLL'ler ise tasarımda ihtiyaç duyulan saat sinyallerinin üretilmesi amacıyla kullanılan birimlerdir. Tasarımda kullanılabilecek 20 adet global (evrensel) saat pini bulunmaktadır. Tasarımda kullanılabilecek Giriş/Çıkış kümeleri 8 adet ile sınırlandırılmıştır. Yani FPGA üzerinde bulunan G/Ç birimlerinden tek tasarımda 8 tanesi FPGA tarafından G/Ç bloğu olarak kullanılabilmektedir. FPGA üzerinde kullanılabilecek toplam 528 adet pin bulunmaktadır. Bu 528 adet pin saat, konfigürasyon ve programlama pinlerini de içermektedir. Bu nedenle hepsi kullanıcının denetimine açık değildir.

2.4 İKİLİK SAYI SİSTEMLERİ

Günlük hayatta sayılar onluk tabanda ifade edilirler. Bu tabanda sayılar 0,1,2,3,4,5,6,7,8 ve 9 değerlerini almaktadırlar. İkilik sistemlerde ise sayılar yalnızca 0 ve 1 değerlerini almaktadırlar. İkilik sayı sistemi, sayısal elektronik devrelerinde uygulanabilirliğinin kolay olması açısından vazgeçilmez hal almıştır. Hemen hemen bütün bilgisayar mimarileri bu sayı sistemine göre çalışmaktadırlar.

Temel ikilik sistemlerde iki tabanındaki her kelime pozitif tamsayı olarak yorumlanır. İkilik tabanda 32 bit uzunluklu kelimenin değeri olan "S" Denklem (2.1)'de gösterilmiştir. Burada en soldaki bit (b_{31}) en fazla değerlikli bit (Most Significant Bit –MSB), en sağdaki bit (b_0) ise en az değerlikli bit (Least Significant Bit –LSB) olarak adlandırılır.

$$S = b_{31} \times 2^{31} + b_{30} \times 2^{30} + \ldots\ldots + b_1 \times 2^1 + b_0 \times 2^0 \qquad (2.1)$$

Pozitif tam sayılar için yazılan bu denklem farklı kelime uzunlukları için değiştirilebilir. Ancak kullanılan sistemlerde her zaman pozitif tam sayılarla karşılaşılmamaktadır. Negatif tam sayıların kullanılma ihtimali gibi kesirli sayıların da kullanılma ihtimali vardır. Bu durumda kesirli sayılarında ikilik sayı sisteminde yazılması ihtiyacı vardır. Bunun için literatürde çeşitli yöntemler vardır. Bu yöntemler genel olarak iki gruba ayrılabilirler. Bunlar sabit noktalı sayı (fixed point number) sistemleri ve kayan noktalı sayı (floating point number) sayı sistemleridir.

2.4.1 Sabit Noktalı Sayı Sistemleri

Sabit noktalı sayı biçimlerinde sayıların bitleri arasına ikilik nokta koyulabilmektedir (Kim and Sung 1994). Böylece kullanılacak sayının büyüklüğü ve hassasiyeti uygulamaya göre değiştirilebilmektedir.

Sabit noktalı sayı sistemlerinin birçok örneği mevcuttur. Ancak hem yaygın kullanılma oranı hem de kullanılan diğer sayı formatı ile karşılaştırılabilme yapılabilmesi adıyla 32 bit kelime uzunluklu I-Q Math adı verilen sabit noktalı sayı formatı anlatılacaktır. Bu sayı formatında sayıların tam kısmı (integer part) "I" ile kesirli kısmı (fractional part) "Q" ile temsil edilmektedir. Kelime uzunluğu 32 bit olan sabit noktalı sayı I8Q24 olarak ifade edilir. Bu

gösterimde en değerlikli bit (MSB) işaret biti olarak adlandırılır. “0” ise sayı pozitif, “1” ise negatiftir. I8Q24 sabit noktalı sayı formatındaki N sayısının bit temsili Denklem (2.2)’de verilmiştir.

$$N = b_7 b_6 b_5 \ldots . b_1 b_0 b_{-1} b_{-2} \ldots \ldots \ldots . b_{-22} b_{-23} b_{-24} \quad (2.2)$$

Denklem (2.2)’ de gösterimi verilen N sayısının değeri, N_t tam sayı kısmını ve N_k kesir kısmını temsil etmek üzere:

$$N = N_t + N_k \quad (2.3)$$

olarak verilir Burada;

$$N_t = \sum_{t=0}^{6} b_t \times 2^t \quad (2.4a)$$

$$N_k = \sum_{k=1}^{24} b_{-k} \times 2^{-k} \quad (2.4b)$$

olarak tanımlanır. Sayısının işareti ise $(-1)^{b_7}$ ile elde edilir.

Denklem (2.3), Denklem (2.4a) ve (2.4b) de verilenlere göre örneğin 10’luk tabanda verilen 10.1875 sayısı;

$$10.1875 = 00001010.001100000000000000000000 \quad (2.5)$$

olarak gösterilecektir.

I-Q sayı formatında işlemler yapılırken eğer sayı negatif ise sayının 2’nin tümleyeni alınır ve işlemler 2’lik tabanda pozitif gibi devam ettirilir.

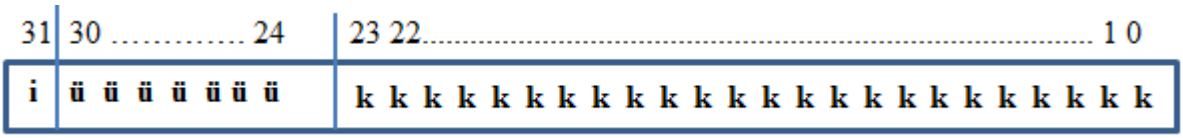
2.4.2 Kayan Noktalı Sayı Sistemleri

Gerçek dünyada sayılarla ile ilgili işlemler yapılırken bazı durumlarda sonsuza kadar giden değerler ile karşılaşmak mümkündür. Bu tarz işlemleri sayısal donanımlarda

gerçekleştirilirken uygulamanın ihtiyaç duyduğu büyüklük ve çözünürlüğe göre yaklaşık değerler ile temsil etmek gerekecektir. Bu temsillerin gerçeğe en yakın şekilde yapılmasını sağlayan sistemler kayan noktalı sayı sistemleridir.

IEEE 754 standartlarında belirlenmiş 4 adet kayan noktalı sayı formatı mevcuttur. Bunlardan en yaygın olarak kullanılanı 32-bit tek duyarlıklı kayan noktalı sayı (32-bit single precision floating point number) sistemidir.

Bu sayı formatında MSB yine işaret bitini ifade etmekte ve "0" ise sayı pozitif, "1" ise sayı negatif olmaktadır. Tek duyarlıklı kayan noktalı sayı formatının bit gösterimi Şekil 2.6'da gösterilmiştir.



Şekil 2.6 Kayan noktalı sayı formatı bit gösterimi.

Şekil 2.6'dan görüldüğü gibi işaret bitinden sonraki en değerlikli 7 bit üs biti olarak, diğer 24 bit ise kesir biti olarak adlandırılmaktadır. IEEE-754 standardında tek duyarlıklı N sayısının değerinin hesaplanması Denklem (2.6)'da verilmiştir.

$$N = (-1)^{N_i} \times (1 + N_k) \times 2^{N_{ü}-127} \tag{2.6}$$

Burada N_i işaret bitini, N_k kesir bitlerinin değerini ve $N_{ü}$ ise üs bitlerinin değerini göstermektedir. 127 sayısı tek duyarlıklı kayan noktalı sayı formatı için taban (bias) değeridir. Diğer sayı formatlarında farklı bir değer almaktadır. Denklem (2.6) doğrultusunda bir örnek yapılmak istenirse IEEE-754 formatındaki;

$$(10111110000010000000100000000000)_{IEEE-754} = -0.13284302 \tag{2.7}$$

değerine karşılık gelmektedir.

BÖLÜM 3

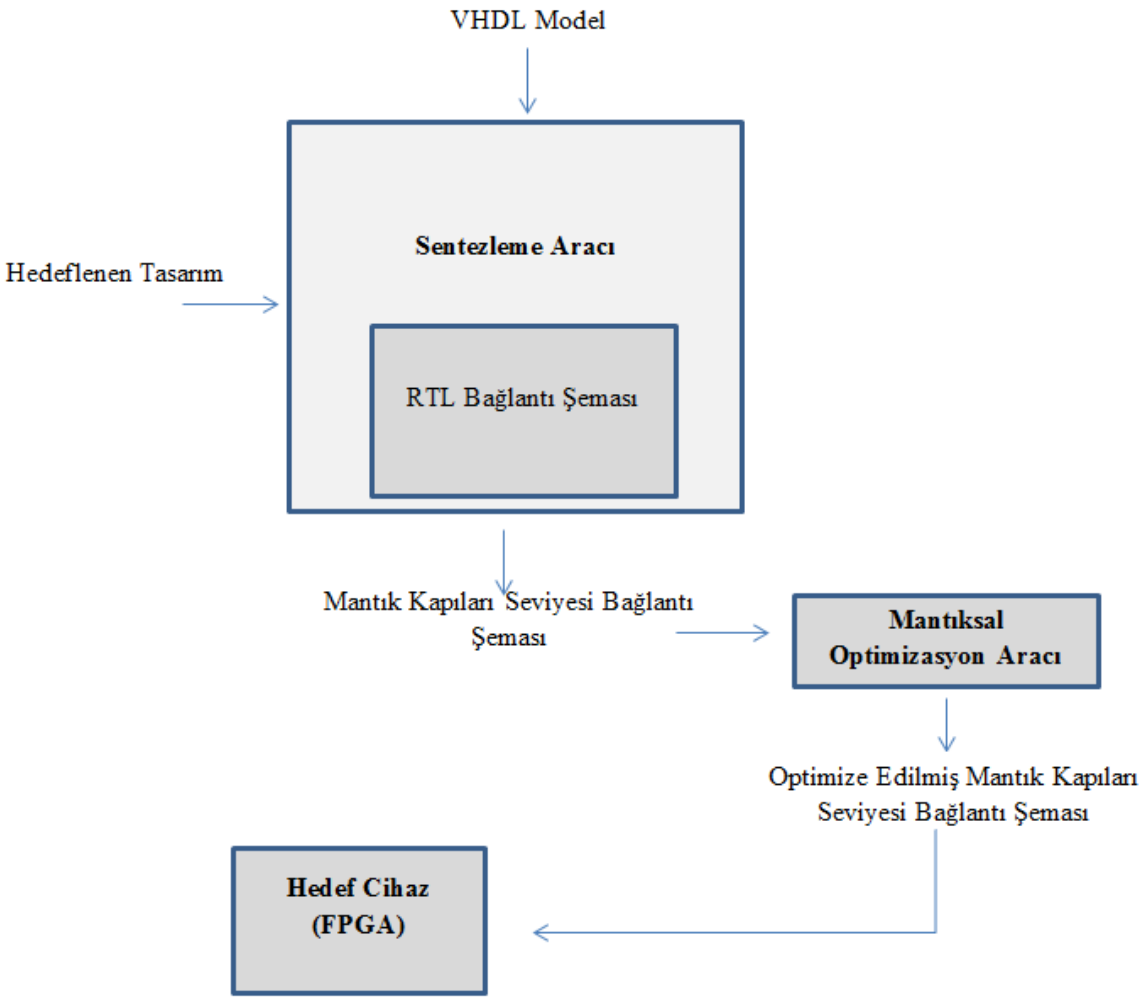
VHDL İLE TASARIM

VHDL anlam olarak Çok Yüksek Hızlı Entegre Devre (Very High Speed Integrated Circuit – VHSIC) ve Donanım Tanımla Dili (HDL) tümcelerinin kısaltmasından oluşmuş, FPGA programlamasında kullanılan bir programlama dilidir.

VHDL donanım üreticileri ile Uygulamaya Özgü Entegre Devre (Application Specific Integrated Circuit – ASIC) tasarım yapan firmaların işbirliği ile ABD Savunma Bakanlığı tarafından geliştirilmiştir. VHDL 'in çalışmaları 1970'lerin başlarına kadar dayanmakla beraber ilk olarak 1987 yılında Elektrik- Elektronik Mühendisleri Enstitüsü (Institute of Electrical and Electonics Engineering – IEEE) standardı olarak kabul edilmiştir. 1993 yılında eklentiler ve güncellemeler ile yeni standart halini almıştır (URL-5 2014).

VHDL donanım tasarımında kullanılan birçok programlama tekniğine müsaade etmektedir. Bunlar:
Sıralı ifadeler/tasarım (Squential language)
Eş zamanlı ifadeler/tasarım (Concurrent language)
Donanım düzen tasarımı (Net list language)
Zamanlama/Senkronizasyon (Timing specifications)
Dalga formu üretimi (Waveform generation)

VHDL ile davranışsal seviye gibi üst seviye modelleme aşamasından donanım düzeni seviyesi gibi düşük seviye tasarıma kadar geniş bir yelpazede tasarım yapmak mümkündür. Fakat düşük seviyelere inildikçe göz önünde bulundurulması gereken durumlar dolayısıyla da tasarımın karmaşıklığı artacaktır. VHDL ile tasarımı yapılan devre sentezleme araçları ile donanım seviyesine indirilebilmektedir (Şekil 3.1).



Şekil 3.1 VHDL ile tasarım akış şeması (Mohamad 2013).

Kullanılan sentezleme aracına göre Saklayıcı Transfer Seviyesi (Register Transfer Level – RTL) bağlantının farklılık göstermesi mümkündür. RTL tasarım mandallar, aritmetik mantık birimleri, çoklayıcılar ve aralardaki bağlantıları içermektedir. Tüm bunlar sentezleme aracı içerisindeki RTL bağlantılar oluşturulduktan sonra sentezleme işlemi sonucunda mantık kapıları seviyesinde bağlantılar oluşturulacaktır. Daha sonra mantık optimizasyon araçları ile optimize edilmiş mantık kapıları seviyesinde tasarım oluşturulacak ve bu bağlantı şekli FPGA içerisine gönderilebilecektir. Günümüzde kullanılan birçok sentezleme aracı bulunmaktadır. Bunlardan en yaygın olarak kullanılanları Xilinx firmasına ait "XST" ve Altera firmasına ait "Quartus II" sentezleme araçlarıdır. Bu çalışmada kullanılan FPGA Altera firmasına ait olduğu için sentezleme aracı olarak "Quartus II 13.1" kullanılmıştır.

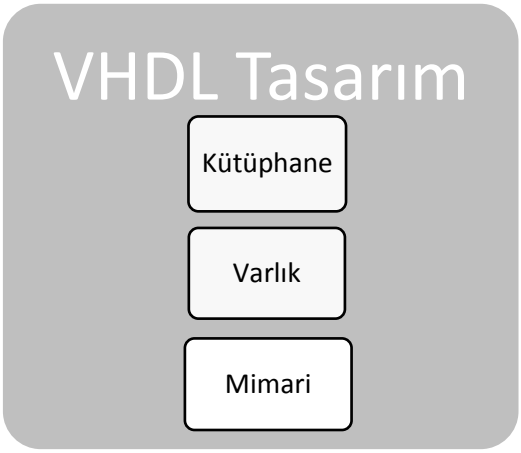
VHDL ile tasarım yapılırken her şeyden önce yazılacak kodun herhangi bir programlama dilinden farklı olarak yorumlanıp derlendiği göz önünde bulundurulmalıdır. VHDL ile yazılan

kodların karşılığında derleme sonucunda FPGA üzerinde işlemi yapacak bir mantık devresi sentezlenerek fiziksel bir devre elde edilmektedir. Bu nedenle yazılan VHDL kod parçacığının sentezlenebilir olması her daim önemli bir ölçüt olarak öne çıkmaktadır. Sentezlenebilir olmayan komutların donanım karşılıkları yoktur ve simülasyon amaçlı kullanılmaktadırlar, FPGA içerisine yüklenemezler.

VHDL kodları yazılırken diğer tüm yazılım dillerinde olduğu gibi göz önünde bulundurulması gereken kurallar vardır. VHDL kodunda kullanılacak değişken isimleri Türkçe ve özel karakter(alt tire hariç) içermemelidir. Değişkenler harf ile başlamalıdır. Bir diğer önemli husus ise VHDL' in büyük küçük harf duyarlılığı yoktur. Yani küçük harf ile tanımlanan bir değişken ya da komut büyük harf ile de kullanılabilir. Bu durum VHDL in kendi hazır komutlarında da geçerlidir.

VHDL kod bloğu temel olarak 3 temel kısımdan oluşmaktadır. Bu kısımlar;
Kütüphane (library) tanımlama kısmı
Varlık (entity) oluşturma kısmı
Mimari (architecture) oluşturma kısmı

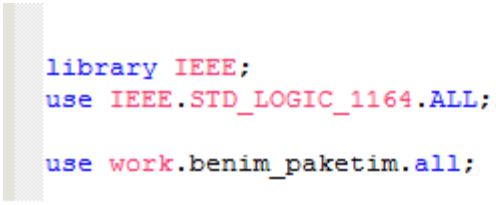
VHDL kod bloğu yukarıda verilen kısımlar haricinde de konfigürasyon (configuration), paket (package), bileşen (component) kısımlarını içerebilmektedir. VHDL kod bloğunu gösteren yapı Şekil 3.2'de verilmiştir.



Şekil 3.2 VHDL tasarım temel bloğu (Çavuşlu ve Kösten 2015).

3.1 KÜTÜPHANE KISMI

Bu kısımda tasarımda kullanılacak olan kütüphanelerin tanımlamaları yapılmaktadır. “library” etiketinden sonra kullanılacak kütüphanenin adı, “use” etiketinden sonrada ilgili kütüphanenin kullanılacak paketi tanımlanmaktadır. En yaygın olarak kullanılan kütüphane “IEEE” kütüphanesi olup, bu kütüphanenin “STD_LOGIC_1164” paketi en fazla kullanılan içeriktir. Daha sonraki bölümlerde anlatılacak olan kullanıcı tarafından oluşturulan paket (package) içeriği “work” kütüphanesinin paketi olarak burada tanımlanmalıdır. Kütüphane kısmına ait örnek tanımlamalar Şekil 3.3’te verilmiştir.



Şekil 3.3 Kütüphane kısmındaki tanımlamalara ait ekran görüntüsü.

Şekil 3.3’te “benim_paketim” adıyla kullanılan paket kullanıcının kendine özgü oluşturduğu kütüphanenin paket içeriğidir.

3.2 VARLIK KISMI

Tasarıma ait giriş ve çıkışların belirlendiği kısımdır. Varlık kısmına VHDL kurallarına uygun olması şartıyla istenilen ad verilebilir. Bu isimlendirme yapılırken “entity isim is” yapısı ile başlanır ve “end isim” yapısı ile sonlandırılır. .Varlığa ait tüm giriş ve çıkışlar varlık içerisindeki “port” adı verilen kısımda gerçekleştirilir. Giriş/çıkışların tanımlanabileceği 4 tür vardır. Bu türler aşağıda verilmiştir.

in: Tanımlanan portun giriş olduğunu gösterir
out: Tanımlanan portun çıkış olduğunu gösterir. “out” olarak tanımlanan porttan veri okuma işlemi gerçekleştirilemez.
inout: Tanımlanan portun hem giriş hem de çıkış olarak kullanılacağını ifade eder.
buffer: Çıkış portudur. Ancak “out” durumundan farklı olarak bu porttan okuma da yapılabilir.

Varlık içerisinde ayrıca jenerik (generic) olarak ifade edilen tanımlamalarda yapılabilir. Bu tanımlamalar varlık içerisinde kullanılan bazı parametrelerin değerlerinin dışarıdan değiştirilmesinde kullanılırlar. Böylece varlık içerisinde kullanılan portların boyutları ya da mimari içerisinde kullanılan vektörlerin uzunlukları ve döngü sayısı gibi özellikler dışarıdan belirlenebilmektedir. Bu özellik tasarımların sabit yapılı değil de değiştirilebilir olması açısından önemlidir. Varlık tanımlamalarına ait örnek Şekil 3.4'te verilmiştir.

```
entity deneme is

    generic ( dizi_uzunlugu:integer:=10);
    port(
    giris: in std_logic_vector(dizi_uzunlugu-1 downto 0);
    cikis_1: out boolean;
    cikis_2: buffer std_logic;
    cift_yon: inout std_logic_vector (2 downto 0)

    );
    end deneme;
```

Şekil 3.4 Varlık kısmındaki tanımlamasına ait ekran görüntüsü.

3.3 MİMARİ KISMI

Mimari kısmı yapılan VHDL tasarımının en önemli kısmını oluşturmaktadır. Bu kısımda kütüphanesi ve giriş çıkışları tanımlanan tasarımın davranışı –yapacağı işlem- tanımlanmaktadır. Mimari tanımlaması "architecture mimarinin_adi of varlik_adi is" yapısı ile başlamakta ve "end mimarinin_adi" kelime grubuyla bitmektedir. Mimari içerisindeki bütün işlemler bu iki kelime grubu arasında tanımlanmalıdır. "varlik_adi" tasarlanan mimari hangi varlık için tanımlanıyor ise onun ismi ile aynı olmalıdır. Mimari yapısı genel olarak iki kısımdan oluşmaktadır. Birinci kısım başlangıçtaki "is" kelimesinden hemen sonra yapılan sinyallerin, sabitlerin ve bileşenlerin tanımlandığı kısımdır. İkinci kısım ise mimari içerisindeki "begin" kelimesinden sonra başlayan işlemlerin ve sinyal atamalarının yapıldığı bölümdür. Mimari içerisinde yapılacak işlemler genel olarak iki sınıfa ayrılabilirler. Bunlar;

Eş zamanlı ifadeler: Paralel olarak yürütülen işlemlerdir. VHDL kodunda hangi sırada bulundukları önem arz etmemektedir.

Sıralı ifadeler: Diğer geleneksel yazılım dillerinde olduğu gibi sırayla yürütülen işlemlerdir. VHDL kodunda üstte olan işlem önce yürütülür.

Sıralı ve eş zamanlı ifadelerin detayları bir sonraki alt bölümde incelenecektir.

Mimari kısmında yapılan tasarıma ait örnek Şekil 3.5'te verilmiştir.

```
architecture davranis of deneme is

signal sonuc:std_logic_vector(7 downto 0);

component toplayici is

port (a: in std_logic_vector(7 downto 0),b:in std_logic_vector(7 downto 0),
      c: out std_logic_vector(7 downto 0));
end component;

begin

toplayici port map ( giris1, giris2 ,sonuc);
   process (clk)
      begin
         if rising_edge(clk)

         cikis<=sonuc;

end process;

end davranis;
```

Şekil 3.5 Mimari kısmındaki işlemlere ait ekran görüntüsü.

Şekil 3.5'te gösterilen mimari tasarımında "is" den sonra kullanılacak olan bağlantı için "sonuç" isimli sinyal ve işlemi yapacak olan bileşen (component) tanımlanmıştır. "begin" kısmından sonraki kısımda ise iki durum mevcuttur. Birinci durum "toplayici" adlı bileşenin işleminin yürütülmesidir. Bu işlem sürekli olarak yapılacak herhangi bir giriş değeri değiştirildiğinde sonuç değişecektir. Bu eş zamanlı ifadelere örnektir. "process" olarak tanımlanan bölümde is sonucun çıkışa aktarılması için "clk" adlı saat sinyalinin yükselen kenarı beklenmelidir. Saat sinyalinin yükselen kenarında sonuç çıkışa aktarılacaktır, bir sonraki yükselen kenara kadar "toplayici" adlı işlemin sonucu değişse bile çıkışa aktarılmayacaktır. "process" sıralı ifadelere örnektir.

3.3.1 Eş Zamanlı İfadeler

Eş zamanlı ifadeler adından da anlaşılacağı üzere mimari içerisinde eş zamanlı olarak yürütülürler. Burada komutların hangi sırada yazıldığının hiçbir önemi yoktur. Dolayısıyla bir komutun çıkışının diğer komutun giriş ifadesi olduğu gibi bağımlı durumlarda kullanılmaları pek mümkün değildir. Birbirinden bağımsız ifadelerin sonuçlarının aynı anda farklı sinyallere aktarılmasını sağlarlar.

Genel olarak mantık operatörleri bu modda kullanılmasına rağmen şartlı ifadeler için WHEN/ELSE ve WITH/SELECT/WHEN gibi komutlar mevcuttur. Döngüler için de GENERATE yapısı mevcuttur. Ancak, döngüler için paralel komutların kullanılması pek tercih edilen bir uygulama değildir. Eş zamanlı ifadelerde kullanılan operatörler Çizelge 3.1' de verilmiştir (Volnei 2005).

Çizelge 3.1 Eş zamanlı ifadelerde kullanıla işlemler.

İşlem Çeşidi	**İşlemler**
Mantıksal (Logical)	NOT,AND,NAND, OR, NOR, XOR, XNOR
Aritmetik	+, - , * , /, **
Karşılaştırma (Comparison)	=, /=, <, >, <=, =>
Kaydırma (Shift)	sll, srl, sla, sra, rol, ror
Bağlantı (Concatenation)	&, (, , ,)

3.3.2 Sıralı İfadeler

Sıralı olarak yürütülen işlemler sadece PROCESS, PROCEDURE ve FUNCTION bölümleri içerisindeki komutlardır. Bu bölümler içerisindeki komutlar üst satırdan başlayarak aşağı doğru yürütülürler. Ancak, her ne kadar bu bölümler içerisindeki komutlar sıralı bir şekilde yürütülseler de bu bölümlerin tamamından oluşan kod blokları birbirlerine ve tüm tasarımdaki diğer paralel komutlarla eş zamanlı olarak yürütülürler.

Sıralı ifade olarak en çok kullanılan kod bloğu “process” içerisine yazılan kodlarlardır. “process” bloğu parantez içerisinde verilen duyarlılık listesinde (sensitivity list) bulunan sinyallerden herhangi birinde değişim olduğu zaman yürütülmeye başlar (URL-4 2014). Duyarlılık listesinde en çok kullanılan sinyaller sıfırlama (reset) ve saat (clock) sinyalleridir. Böylece sistemin sıfırlama sinyaline bağlı olarak sıfırlanabileceği ve çıkışı saat sinyaline göre çıkışa aktaran mandal (flip-flop) yapısı tasarlanabilir.

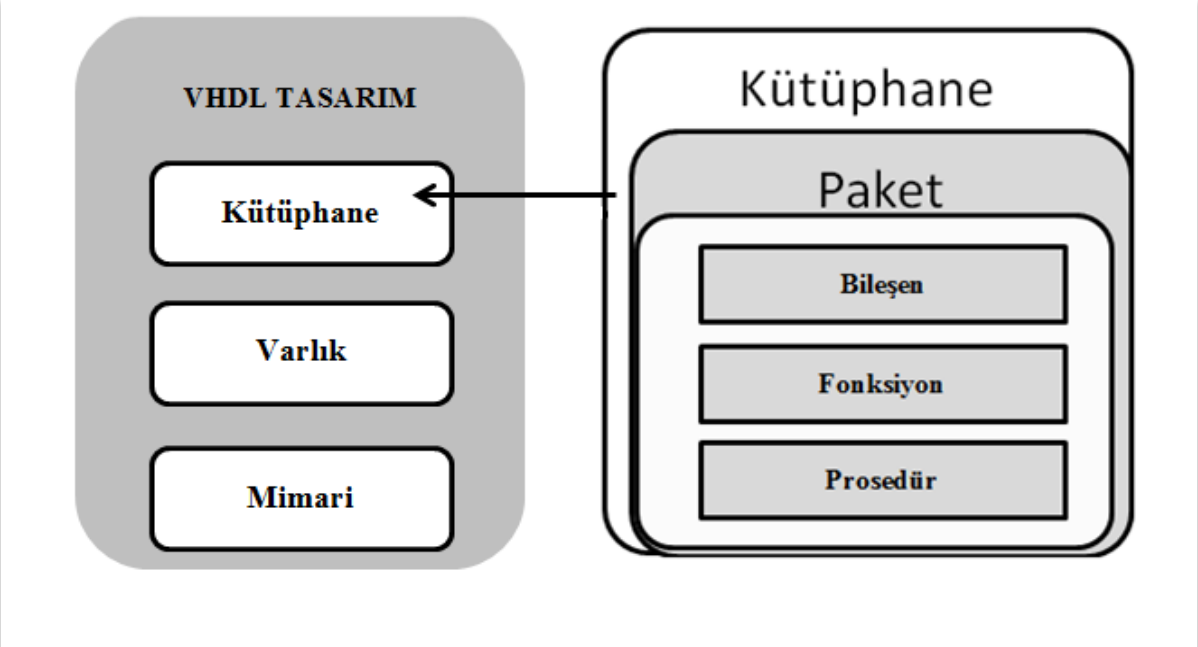
“process” bloğunda sıkça kullanılan komutlar IF, WAIT, CASE ve LOOP komutlarıdır. Bunların yanı sıra aritmetik ve mantıksal operatörler de “process” bloğu içerisinde kullanılabilir.

3.4 VHDL TASARIMINDA KULLANILAN DİĞER KISIMLAR

VHDL tasarımı yapılırken tasarımı kolaylaştırmak adına 3 temel öğe olan kütüphane, varlık ve mimari kısımlarına ek olarak bazı ek kısımlarda kullanılabilir. Bu kısımlardan en çok kullanılanları:

- Paket
- Bileşen
- Fonksiyon
- Prosedür

kısımlarıdır. Bu 4 kısmın temel öğeler ile en yaygın kullanım ilişkisini gösteren diyagram Şekil 3.6’da verilmiştir.



Şekil 3.6 VHDL tasarımında yan kısımların temel kısımlarla ilişkisi.

Şekil 3.6'da gösterilen diyagramda tasarlanan paket kısmının kütüphane tanımlamaları içerisinde olması gerektiği görülmektedir. Paket kısmının temel olarak kullanımı kullanıcının kendine özgü işlemlerini gerçekleştiren kütüphanesini (VHDL dosyası) oluşturarak bu kütüphanenin diğer VHDL tasarımlarında kullanılmasına yöneliktir. Paket tanımlaması, paket içerisinde kullanılacak kütüphane tanımlamaları yapıldıktan sonra "package paket_ismi is" etiketi ile başlar ve "end paket_ismi" ile sonlanır. Paket gövdesi içerisinde farklı bileşenler, fonksiyonlar ve prosedürlerin işlemlerinin tanımlamaları yapılabilir. Paket gövdesi "PACKAGE BODY paket_ismi IS" yapısıyla başlar ve "END paket_ismi" yapısı ile sona erer. Paket tanımlamasına ait örnek Şekil 3.7'de verilmiştir. Paketin kısmının kütüphane kısmında nasıl çağırılacağı Şekil 3.3'te gösterilmişti.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


PACKAGE benim_paketim IS

Function topla (signal s1: std_logic_vector(3 downto 0),

                signal s2: std_logic_vector(3 downto 0)) return std_logic_vector(3 downto 0);
end benim_paketim;



PACKAGE BODY benim_paketim IS

   function topla(signal s1: std_logic_vector(3 downto 0),signal s2: std_logic_vector(3 downto 0))
                return std_logic_vector(3 downto 0) IS
      begin

      return (s1+s2)
   end topla;

end benim_paketim;
```

Şekil 3.7 Paket tasarımına ait ekran görüntüsü.

Bileşen (component) ise paket içerisinde tanımlandığı gibi ana kod içindeki mimari içerisinde de tanımlanabilir. Bileşen bir başka VHDL tasarımı olan dosyanın tasarıma dâhil edilme işlemidir. Bileşen olarak kullanılacak olan VHDL tasarımı olağan şekilde tasarlandıktan sonra kullanılmak istenen tasarımın içerisinde bileşen (component) olarak tanıtılır ve bileşenin özellikleri kullanılır. Çoğu sentezleme araçlarında bileşen olarak kullanılacak tasarımın ana tasarımla aynı dizinde olması istenmektedir. Ana tasarımın mimari kısmında bileşen tanımlanması ve kullanılması Şekil 3.5'te mevcuttur.

Fonksiyon (function) ve prosedür (procedure) yapıları itibariyle "process" alt bölümüne oldukça benzemektedirler. Fonksiyon ve prosedür bir alt program olarak düşünebilirler ve içerisindeki komutlar sıralı olarak yürütülürler. Fonksiyon ve prosedür yapıları da tıpkı bileşen de olduğu gibi paketlerin yanında ana programın mimarisi içerisinde de konumlandırılıp kullanılabilirler (URL-3 2013). Fonksiyon tanımlamasına ait örnek Şekil 3.7'de gösterilmiştir. Prosedür tanımlamasına ait örnek ise Şekil 3.8'de gösterilmiştir. Prosedür ile fonksiyon arasındaki en temel fark fonksiyonun bir değer döndürürken prosedür işleminin birden çok sonuç döndürebilmesidir.

```
architecture davranis of varlik is

   procedure siralama ( signal giris1, giris2 : in integer range 0 to 1000;
                        signal buyuk , kucuk : out integer range 0 to 1000) is
      begin

      if(giris1 > giris2) then
         buyuk <= giris1;
         kucuk <=giris2;
      else
         kucuk <= giris1;
         buyuk <= giris2;
      end if;
   end siralama;

   begin
      process (tetikleme)
      begin
         if (tetikleme ='1') then
         siralama(giris1,giris2,buyuk_cikis,kucuk_cikis);
         end if;
      end process;
end davranis;
```

Şekil 3.8 Prosedür bloğunun mimari içerisinde tanımlanmasına ait ekran görüntüsü.

Şekil 3.8'de "varlik" adlı entity' nin mimarisinde "sıralama" adlı prosedür tanımlanmıştır. Tanımlanan prosedür mimari kod bloğu içerisinde kullanılan "process" içerisinde çağırılmıştır. Yukarıdaki kod parçacığında "tetikleme" sinyali "1" olduğunda prosedür çağırılacaktır. Prosedür bloğunun girişleri varlığın giriş portları olarak kullanılmış, çıkışları ise varlığın çıkış portları olan "buyuk_cikis" ve "kucuk_cikis" değişkenlerine aktarılmıştır. Prosedür bloğunun içerisinde bulunan "buyuk" ve "kucuk" sinyalleri yerel değişkenleri ifade etmektedir. Prosedür bloğu çağırılırken sinyallerin tanımlanma sırası blok içerisindeki tanımlama ile aynı olmalıdır.

3.5 VHDL TASARIMINDA KULLANILAN DEĞİŞKEN TİPLERİ

Sentezlenebilen ve verimli bir VHDL tasarımı yapabilmek için hangi değişken tiplerinin kullanıldığının ve bunlardan hangilerinin sentezlenebilir olduğunun bilinmesi önemlidir. VHDL, IEE 1076 ve IEEE 1164 standartları ile belirtilen değişken tiplerinin yanı sıra kullanıcı tarafından da değişken türetilmesine olanak sağlamaktadır.

VHDL tasarımı yapılırken “std” ve “ieee” kütüphaneleri içerisinde bulunan tanımlı değişken tipleri, bunların sentezlenebilir olup olmadıkları ve alabildikleri değerler aşağıda verilmiştir (Volnei 2005).

BIT: Sentezlenebilir. Bir bitlik binary değer.
BIT_VECTOR: Sentezlenebilir. Belirtilen uzunlukta binary değer.
BOOLEAN: Sentezlenebilir. Doğru (True) ya da yanlış (false).
INTEGER: Sentezlenebilir. 32-bitlik integer (-2,147,483,647 ile +2,147,483,647 aralığında).
NATURAL: Sentezlenebilir. Pozitif integer (0 ile +2,147,483,647 aralığında).
REAL: Sentezlenemez. -1.0E38 ile +1.0E38 aralığında
Fiziksel tanımlamalar: Sentezlenemez. “time” (zaman) “voltage” (gerilim) gibi fiziksel değerler.
Karakter tanımlamaları: Sentezlenemez. Tek karakter veya karakter dizisi
STD_LOGIC: Sentezlenebilir. Bir uzunlukta IEEE 1164 standardında belirtilen 8 durum.
STD_ULOGIC: Sentezlenebilir. Bir uzunlukta IEEE 1164 standardında belirtilen 9 durum.
STD_LOGIC_VECTOR: Sentezlenebilir. Belirtilen uzunlukta IEEE 1164 standardında belirtilen 8 durum.
STD_ULOGIC_VECTOR: Sentezlenebilir. Belirtilen uzunlukta IEEE 1164 standardında belirtilen) durum.
SIGNED: Sentezlenebilir. STD_LOGIC_VECTOR ile aynı fakat ilk bit işaret biti negatif değerler alabilir. Aritmetik işlemlere tabi tutulabilir.
UNSIGNED: Sentezlenebilir. SIGNED ile aynı fakat il bit işaret değil sadece pozitif değerler.

STD_LOGIC ve STD_LOGIC_VECTOR için IEEE 1164 standardında belirlenen 8 durum aşağıdaki gibidir.

“X”: Bilinmeyene zorlayan (Forcing Unknown)
“0”: Düşük seviyeye zorlayan (Forcing Low)
“1”: Yüksek seviyeye zorlayan (Forcing High)
“Z”: Yüksek empedans (High Impedance)
“W”: Zayıf bilinmeyen (Weak Unknown)
“L”: Zayıf düşük seviye (Weak Low)
“H”: Zayıf yüksek seviye (Weak High)
“-“: Önemli değil (Don’t Care)

Bu durumlardan “X”, “0”, “1” ve “Z” sentezlenebilirdir. STD_ULOGIC ve STD_ULOGIC_VECTOR’ de ise ekstra “U” durumu mevcuttur. Bu durum “Çözümlenmemiş (unresolved) ” durumunu ifade etmektedir.

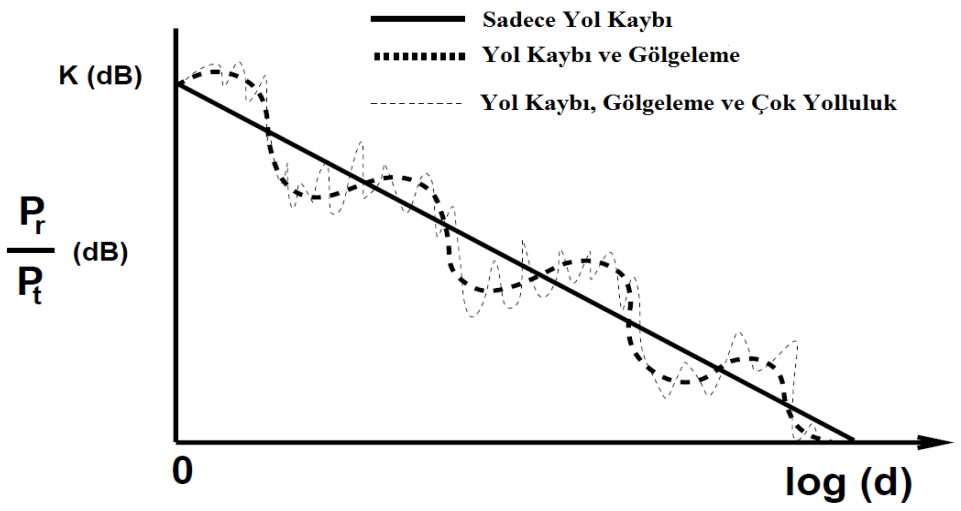
BÖLÜM 4

DİK FREKANS BÖLMELİ ÇOĞULLAMA (OFDM) YÖNTEMİ

OFDM, kelime anlamı olarak Dik Frekans Bölmeli Çoğullama manasına gelen "Orthogonal Frequency Division Multiplexing" İngilizce kelimelerin baş harflerinden oluşan bir kısaltmadır. Bu teknikte amaç çoğullama yaparken frekans eksenini daha verimli kullanmaktır. OFDM tekniğini ve avantajlarını anlamak için öncelikle kablosuz kanal modelinin anlaşılması gerekmektedir.

4.1 KABLOSUZ KANAL

Kablosuz haberleşmede veriler vericiden alıcıya elektromanyetik dalgaların atmosferde yayılımı yoluyla iletilir. Vericiden gönderilen sinyaller doğrudan görüş yolundaki (Line of Sight - LOS) engeller sebebiyle yansıma, kırılma ve saçılma gibi etkilerden dolayı hedefe birçok yoldan farklı güç ve zamanlarda ulaşmaktadır. Vericiden gönderilen sinyal gücünün alıcıda alınan sinyal gücüne oranının mesafe ile ilişkisi logaritmik eksende Şekil 4.1'de gösterilişitir.



Şekil 4.1 Kablosuz kanalda işaret gücünün mesafe ile ilişkisi (Goldsmith 2005).

Burada K alıcı ile verici arasındaki güç kazancını, P_t vericideki sinyal gücünü, P_r alıcıdaki sinyal gücünü ve d verici ile alıcı arasındaki mesafeyi göstermektedir. Şekil 4.1' de görüldüğü üzere vericiden gönderilen sinyalin alıcıda alınmasına mesafeye bağlı olarak 3 temel bozulma etki etmektedir. Yol kaybı (Path Loss) elektromanyetik dalgaların atmosferde yayılırken sinyal gücünün mesafeye bağlı olarak azalmasıdır. Gölgeleme (Shadowing) elektromanyetik dalgaların verici ile alıcı arasındaki engellerden kaynaklı olarak sinyal gücünün azalmasıdır. Çok yolluluk (Multipath) ise vericiden gönderilen sinyalin kırılma, saçılma ve yansıma gibi etkenlerden dolayı alıcıya birçok yoldan ulaşması sonucu oluşan sönümlemedir (fading).

Yol kaybı ve gölgeleme kullanılan modülasyon tekniğinden bağımsız olarak elektromanyetik atmosferde dalgaların yayılmasından ve ortam şekillerinden kaynaklıdır. Çok yollu sönümlemenin etkisi ise kullanılan modülasyon tekniklerine ve alıcı tasarımlarına göre azaltılabilmektedir. OFDM tekniğinin en büyük avantajlarından biri de çok yollu sönümlemelere karşı dirençli olmasıdır. Bu nedenle çok yollu sönümleme ayrıntılı bir şekilde incelenecektir.

4.1.1 Sönümlemeli Kanal

Kablosuz ortamda işaret, iletilirken bitki örtüsünden ve ortamdaki binalar gibi engellerden dolayı yansıma, kırılma ve saçılma gibi bozucu etkilere maruz kalır. Bu bozucu etkilerden kaynaklı olarak vericiden gönderilen işaret alıcıya birden çok yoldan zaman gecikmeli olarak ulaşır. Bu zaman gecikmeleri iletilen işaretin frekansına ve ortam şekillerine göre değiştiğinden kablosuz haberleşme ortamını zamanla değişen bir filtre gibi modellemek mümkündür (Rappaport 2002). Alıcıya farklı yollardan gelen sinyaller farklı genlik ve fazlara sahiptirler. Bu fazların zıt yönlü olması durumunda alıcıdaki sinyalde yıkıcı etki meydana gelir ve bu etki sönümleme olarak adlandırılır.

Alıcıda oluşan sönümleme iki farklı boyutta incelenebilir. Zaman boyutunda, kanalın zamanla değişimine göre sönümleme hızlı veya yavaş sönümleme olarak sınıflandırılırken, frekans boyutunda ise işaretin bant genişliğinin kanalın evre uyumlu bant genişliğine oranına göre sönümleme frekans seçici veya düz sönümleme olarak sınıflandırılır.

Kablosuz kanalın zamanla değişen bir filtre olarak kabul edilmesinden yola çıkılarak alıcıda alınan işaret ile vericiden gönderilen işaret arasındaki bağıntı:

$$r(t) = s(t) * h(t) + n(t) \tag{4.1}$$

şeklinde tanımlanır. Burada $r(t)$ alıcıda alınan işaret, $s(t)$ vericiden gönderilen karmaşık zarflı işaret, $h(t)$ kanalın karmaşık sönümleme katsayısı, $n(t)$ tüm elektronik devrelerde oluşan ısıl gürültü olarak da tanımlanan Toplanabilir Beyaz Gauss Gürültüsü (TBGG, Addictive White Gaussian Noise –AWGN) ve $*$ evrişim (konvolüsyon) işlemidir. Kanal dürtü tepkisi $h(t)$

$$h(t) = \alpha(t)e^{j\emptyset(t)} \tag{4.2}$$

olarak modellenir. Burada $\alpha(t)$ ifadesi sönümleme katsayısı $h(t)$ 'nin zarfı, $\emptyset(t)$ ise zarfıdır. $\alpha(t)$ kullanılan haberleşme kanalının yapısına göre farklılık gösteren rastgele süreçtir. Haberleşme sistemlerinde $\alpha(t)$ rastgele sürecini modellemek amacıyla en çok kullanılan Rayleigh, Rician ve Nakagami- m olasılık yoğunluk fonksiyonlarına sahip rastgele süreçlerdir. Zaman gecikmesini ifade eden $\emptyset(t)$ ise olasılık yoğunluk fonksiyonu [0-2π] aralığında tekdüze (uniform) dağılımlı rastgele süreçtir (Proakis and Salehi 2005).

4.1.1.1 Frekans Seçici Sönümleme

Sönümlemeli kablosuz kanalına vericiden gönderilen bir dürtü (darbe) işaretinin alıcıda ilk alındığı an ile farklı yollardan yansıyarak gelen kabul edilebilir bir güç seviyesindeki son sinyal arasındaki geçen süreye maksimum zaman gecikmesi adı verilir. Maksimum zaman gecikmesi T_m ile gösterilir. Kanal evre uyumlu bant genişliği ise kanalın iletilen işarete aynı $\alpha(t)$ sönümleme katsayısının ve doğrusal fazda $\emptyset(t)$ nin uygulandığı frekans aralığı olarak tanımlanır. Kanal evre uyumlu (coherence) bant genişliği B_c ile tanımlanır. Maksimum zaman gecikmesi ile kanal evre uyumlu bant genişliği arasındaki ilişki

$$B_c \approx \frac{1}{T_m} \tag{4.3}$$

olarak tanımlanır (Rappaport 2002).

İletilen işaretin bant genişliği W, kanal evre uyumlu bant genişliğinden daha büyük ise ($W \gg B_C$)kanal frekans seçici (frequency selective) olarak adlandırılır. Bu durumda iletilen işaret frekans boyutunda düşünüldüğünde W bant genişliği boyunca farklı sönümlemelere maruz kalacaktır. Denklem (4.2)'de verilen kanal dürtü tepkisi frekans seçici durumda

$$h(t) = \sum_{l=1}^{L} \alpha_l(t) e^{j \phi_l(t)} \delta(t - \tau_L) \quad (4.4)$$

olacaktır. Burada L alıcıda çözülen yol sayısı olarak adlandırılır. Bu durum sadece gönderilen işaretin genlik ve fazının değişmesi olarak düşünülemez, frekans seçici durumda "t" anında gönderilen sembolün zaman gecikmeli halleri "t+τ_L" anındaki sembol alıcıya gelene kadar alıcıya gelmeye devam edecektir. Bu da ciddi semboller arası karışım (SAK Inter Symbols Interference ISI) oluşmasına neden olacaktır (Chen and Guizani 2006). Semboller arası karışım çok yollu iletimden kaynaklanan kanalın maksimum zaman gecikmesinin iletilen işaretin sembol süresinden daha fazla olmasından kaynaklanmaktadır. Frekans seçici sönümlemeli kanal

$$T_s < T_m \quad (4.5)$$

durumunda ortaya çıkacaktır. Burada T_s iletilen işaretin sembol süresini ifade etmektedir. Frekans seçici kanalın neden olduğu SAK'ın giderilmesi için alıcıda kanal denkleştirme uygulanır.

4.1.1.2 Düz Sönümleme

Kanalın maksimum zaman gecikmesinin kanalda iletilen işaretin sembol süresinden çok daha küçük olduğu durumlardaki sönümleme düz (flat) sönümleme olarak tanımlanır ($T_m \ll T_s$).

Bu durumda kanalın maksimum zaman gecikmesinin tersi olarak tanımlanan kanal evre uyumlu bant genişliği B_C, kanalda iletilen işaretin bant genişliği W'dan çok daha büyük olacaktır. Bu da iletilen işaretin W bant genişliği boyunca kanal tarafından aynı sönümlemeye maruz bırakılacağı anlamına gelmektedir. Düz sönümleme durumunda kanal dürtü tepkisi Denklem (4.2)'deki gibi modellenir.

İletilen işaretin sembol süresi maksimum zaman gecikmesinden çok daha büyük olduğu için gönderilen sembolün çok yollu etkiden dolayı gecikmeli olarak gelen kopyaları bir sonraki sembolle örtüşmeyecektir. Bu da SAK oluşumu engelleyecektir. Ancak, alıcıda alınan işaretin gücü çok yollu etkiden kaynaklanan kanal kazanç dalgalanmasından dolayı zamanla değişecektir ve derin sönümleme (bayılma) (deep fading) oluşma ihtimali doğacaktır. Bu etkinin önüne geçmek için düz sönümlemeli kanallarda gönderilen işaretin gücü 20-30 dB arttırılarak alıcıya iletilir (Chen and Guizani 2006).

4.2 OFDM YÖNTEMİNİN GELİŞİMİ

Son yıllarda kablosuz haberleşme alanında oldukça popülerleşmesine rağmen ilk OFDM çalışmaları 1950'li yıllara dayanmaktadır. Bu yıllarda bir frekans çoğullama tekniği olan OFDM askeri projelerde denenmiştir. 1966 yılında Robert W. Chang OFDM sisteminin çoklu sayıda osilatör ile bilginin eş zamanlı gönderilebileceğini öne sürerek bunu matematiksel olarak göstermiş ve 1970 yılında OFDM sisteminin patentini Amerika Birleşik Devletleri'nden almıştır (Li and Stüber 2006).

Alt taşıyıcıların birbirinden bağımsız çok sayıda osilatör kullanılarak gönderilmesi işlemi son derece karmaşık ve uygulanabilirliği zor olan bir sistemdir. Sistemin daha kolay uygulanabilir hale gelmesi ve osilatör kullanımını azaltmak amacıyla çalışmalara devam edilmiş ve Ayrık Fourier Dönüşümü (AFD, Discrete Fourier Transform –DFT) kullanılarak bunun sağlanabileceği önerilmiştir (Weinstein and Eber 1971).

Bu yıllarda OFDM tekniğinin uygulanmasında bir diğer problem ise alt taşıyıcılar arası karışımdı.1980 yılında Peled ve Ruiz OFDM sembollerinin alıcıda örtüşmesini engellemek amacıyla önemli bir çalışma yapmışlardır (Peled and Ruiz 1980). Bu çalışmada vericide uygulanan TAFD'den sonra OFDM sembollerine dairesel ön ek eklenmesi önerilmiştir. Dairesel ön ek eklenme işlemi iletilen OFDM sembolünün belirli uzunluktaki son örneklerinin sembolün önüne kopyalanarak gönderilmesi işlemidir. Böylece OFDM tekniğinin günümüzde kullanılan haliyle tanımlanması yapılmıştır.

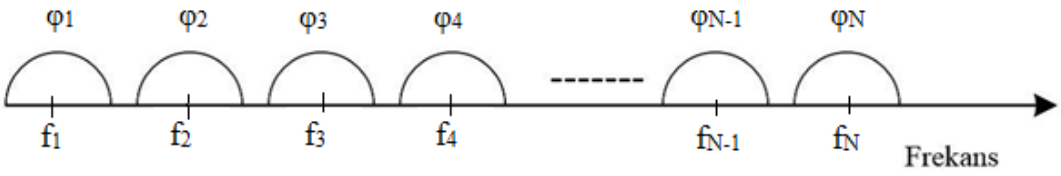
Bu gelişmeler doğrultusunda OFDM'in ilk ticari uygulaması 1987 yılında gerçekleştirilmiştir. 1994 yılında ise OFDM standartlarda yerini almıştır (Prasad 2004). 1990'lı yıllarda OFDM teknolojisi bant verimliği, frekans seçici sönümlemeye karşı direnci ve SAK sorunun ortadan

kalkması gibi özelliklerinden dolayı popülerleşmeye başlamış ve HDSL, ADSL, VDSL gibi sayısal haberleşme sistemlerinde kullanılmıştır.

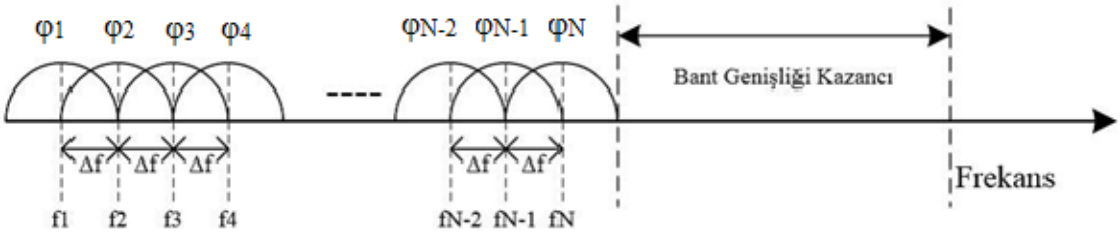
Günümüzde OFDM en çok tercih edilen haberleşme tekniği olarak göze çarpmaktadır. Hala Sayısal Video Yayıncılığı (DVB), Kablosuz Yerel Ağlar (WLAN) ve 4.nesil kablosuz sitemler WIMAX ve LTE için standartları bulunan OFDM teknolojisinin 5.nesil kablosuz sistemlerinde altyapısını oluşturması beklenmektedir (Zhang and Chen 2008).

4.3 OFDM SİNYAL YAPISI VE BLOK DİYAGRAMI

Haberleşme sistemlerinde farklı sinyallerin aynı zaman ve/veya frekans aralıklarında gönderilebilmesi için çoğullama teknikleri kullanılmaktadır. Bu teknikler zaman bölmeli çoğullama (Time Division Multiplexing -TDM), frekans bölmeli çoğullama (Frequency Division Multiplexing -FDM) ve kod bölmeli çoğullama (Code Division Multiplexing-CDM)'dır. Çok taşıyıcılı modülasyon tekniği olan OFDM temel olarak FDM yapısını kullanmaktadır. Vericiden iletilen semboller aynı zaman aralığında farklı frekans bandını kullanarak alıcıya aktarılmaktadır. Geleneksel FDM sistemlerinde kullanılan alt kanallar birbiriyle örtüşmezler ve arada koruma bandı olarak adlandırılan boşluklar vardır (Şekil 4.2). Şekil 4.2'de her bir komşu kanal φ_k'lar (k=1,2,…N) arasında mesafe olduğu gözükmektedir bu durum kullanılan frekans bandının verimliliğini düşürmektedir. Burada f_k (k=1,2,…N) alt kanallara ait taşıyıcı frekanslarını göstermektedir. OFDM tekniğinde ise FDM tekniğinden farklı olarak komşu kanallar arasında bir boşluk yoktur ve komşu kanallar birbirleriyle örtüşmektedirler (Şekil 4.3). Alt taşıyıcıların birbirine dik olarak seçilmesiyle alt taşıyıcılar arası karışımında önüne geçilmiş olur. Şekil 4.3'te de görüldüğü gibi alt taşıyıcılar birbirleri ile örtüşürler fakat dikgenlik dolayısıyla bir alt taşıyıcının merkez frekansında diğer alt taşıyıcıların genlikleri sıfırdır. Alt taşıyıcılar arası mesafe dikgenlik şartını sağlayan Δf olarak ayarlanarak alt taşıyıcıların örtüşmesi sağlanır ve bant genişliği kazancı elde edilmiş olur. Böylece FDM ile aynı sembol hızına sahip veriler daha az bir bant kullanılarak iletilebilir.



Şekil 4.2 FDM tekniğinde komşu kanalların frekans gösterimi.



Şekil 4.3 OFDM tekniğinde alt taşıyıcıların frekans gösterimi.

OFDM tekniğinde $0 < t < T$ zaman aralığının herhangi bir anında N adet komşu alt işaretten k. işaretin karmaşık bant geçiren zarfı

$$\widehat{s_k}(t) = S[k]g(t - kT)\exp(j2\pi(f_c + k\Delta f)t), \quad 0 \le k \le N - 1 \tag{4.5}$$

olarak tanımlanır. Burada, $S[k]$ modülasyon sonucu oluşan karmaşık katsayı, $g(t)$ $0 < t < T$ aralığındaki darbe şekillendiricisini, T $(T = NT_S)$ alt taşıyıcı uzunluğunu, f_c OFDM işaretlerinin merkez frekansını ve Δf alt taşıyıcıların frekansları arasındaki mesafeyi ifade etmektedir (Proakis and Salehi 2007). Alt taşıyıcıların frekansları arasındaki mesafe Δf

$$\Delta f = f_{k+1} - f_k = \frac{1}{T}, \; k = 1,2, \dots \dots N \tag{4.6}$$

olarak tanımlanır. N tane alt taşıyıcıdan oluşan bir OFDM sembolü N tane $\widehat{s_k}(t)$'lerin $(k = 0,1, \dots \dots, N - 1)$ toplamından oluşur ve

$$\hat{s}(t) = \sum_{k=0}^{N-1} \widehat{s_k}(t) = \sum_{k=0}^{N-1} S[k]g(t - kT)\exp(j2\pi(f_c + k\Delta f)t) \; 0 < t < T \tag{4.7}$$

ile verilir. Şayet Denklem (4.6)'da verilen T süresi sembol süresine eşit ise Denklem (4.7)'de verilen alt taşıyıcılar birbirine dik olacaktır.

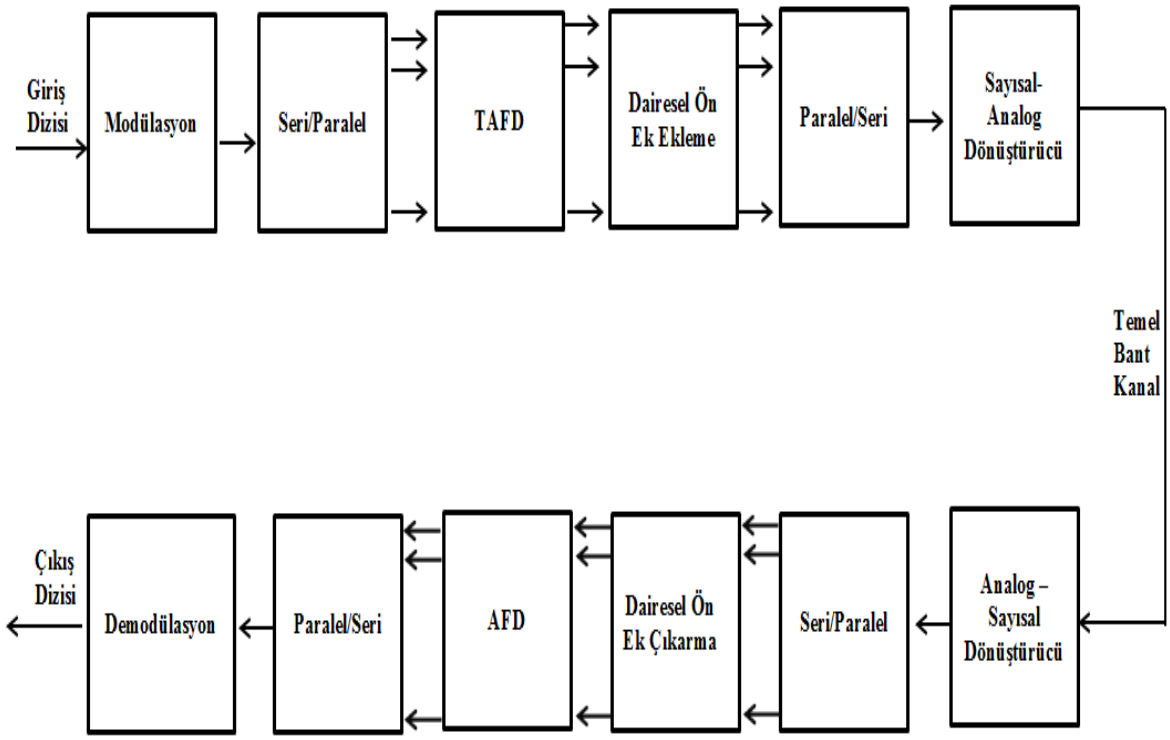
Denklem (4.7)'deki bant geçiren işaretin temel bant karşılığı sürekli zaman $s(t)$ işareti dikdörtgen darbe şekillendirici için sembol süresince N kez örneklendiğinde $t = nT_S$ $(n = 0,1, \dots . N-1)$ ayrık zaman temel bant işareti

$$s[n] = \sum_{k=0}^{N-1} S[k]\exp(\frac{j2\pi kn}{N}), \quad n = 1,2, \dots \dots N-1 \tag{4.8}$$

olarak ifade edilebilir (Haykin and Moher 2005).

Denklem (4.8)'de gösterilen temel bant ayrık işaret formları ters ayrık fourier dönüşüm denkleminin aynısıdır. Bu durumda modülasyon sonrası işaretlerin TAFD'si alınarak dikgenlik şartı bozulmadan alt taşıyıcılara bölünebileceği görülmektedir.

Modülasyon sonucunda oluşan dalga formlarının dikgenlik şartının bozulmadan ve birden çok osilatöre ihtiyaç duyulmadan alt taşıyıcılara bölünebileceği Denklem (4.7) ve Denklem (4.8)'den gözükmektedir. Bu durumdaki temel bant OFDM mimarisini gösteren blok diyagramı Şekil 4.4'te verilmiştir.



Şekil 4.4 OFDM blok diyagramı.

Şekil 4.4'te gösterilen temel bant OFDM blok diyagramında giriş verisi olarak gelen sayısal işaret (0 ve 1 bit dizisi) seçilen modülasyon tekniğinin işaret-yıldız kümesine göre haritalanır ve daha sonra TAFD uygulanması için alt taşıyıcısı sayısı kadar paralel verilere dönüştürülür. Ters ayrık fourier dönüşümü (TAFD) alındıktan sonra semboller arası karşımın (SAK) önüne geçmek amacıyla alt taşıyıcılara dairesel ön ek (CP) eklenir ve tekrar seri verilere dönüştürülür. Sayısal-Analog Çevirici (Digital Analog Converter –DAC) ile analog sürekli işarete dönüştürülen ayrık işaret temel bant kanaldan gönderildikten sonra alıcı kısmında Analog-Sayısal Çevirici (ADC) yardımıyla tekrar sayısal veriye dönüştürülür. Alıcıdaki sayısal veri alt taşıyıcı sayısı kadar paralel veriye dönüştürülür. Her bir alt taşıyıcı için ayrık fourier dönüşümü (AFD) paralel olarak alınmadan önce vericide eklenen dairesel öne ekler alt taşıyıcılardan çıkarılır. AFD işlemi Denklem (4.9)'da gösterilmiştir. Ayrık fourier dönüşümü sonrasında tekrar seri hale getirilen veriler ($s[n]\ n = 0,1, \dots, N-1$) modülasyon bloğundaki haritalama göre demodülasyon işlemine tabi tutulup ve gönderilmiş olunan sayısal bit dizisi elde edilmiş olur.

$$S[k] = \frac{1}{N}\sum_{n=0}^{N-1} s[n]e^{-\frac{j2\pi nk}{N}},\ k = 0,1,2 \dots, N-1 \qquad (4.9)$$

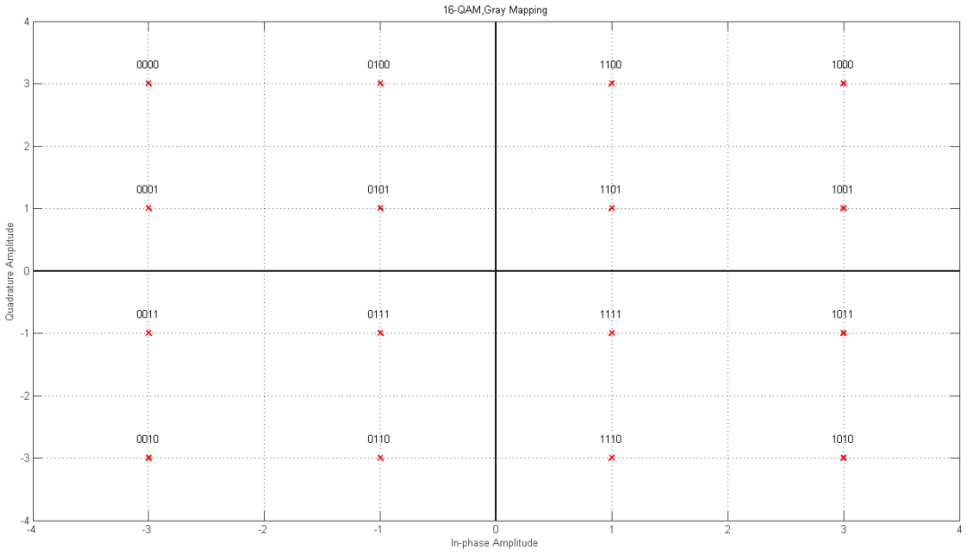
4.3.1 Temel Bant Modülasyon Yöntemleri

Modülasyon modern iletişim tekniklerinin vazgeçilmez bir parçasıdır. Giriş verisinin analog ya da sayısal olmasına göre analog modülasyon ve sayısal modülasyon teknikleri olarak iki temel kategoriye ayrılabilir. Modülasyon işleminde amaç bilgi işaretinin taşıyıcının genlik, faz ya da frekans bileşeninde iletilmesidir. Bilgi işaretinin taşıyıcının hangi özelliğini değiştirdiğine göre de sınıflandırılırlar: genlik modülasyonu, faz modülasyonu, frekans modülasyonu. OFDM sayısal giriş dizisinin olduğu bir sistem olduğu için kullanılacak modülasyon sayısal modülasyon tekniklerinden biri olmalıdır. Temel bant sayısal modülasyon tekniklerinde 0/1 bit dizisinden oluşan giriş verisi karmaşık düzleme haritalanır. Bu haritalama sonucu yatay eksen(I eşfaz-genlik), dikey eksen(Q dikfaz) olarak isimlendirilir (Sklar 2001).

OFDM sistemlerinde en fazla kullanılan modülasyon teknikleri M'li Dördün Genlik Modülasyonu (M-DÖGM, M-ary Quadrature Amplitude Modulation- MQAM) ve M'li Faz Kaydırmalı Anahtarlama (M-FzKA, M-ary Phase Shift Keying – MPSK) teknikleridir. Burada M modülasyon seviyesini ifade eder ve haritalama sonucunda karmaşık eksen M adet farklı değer bulunur. Giriş bit dizisi $k=\log_2(M)$'lık gruplar halinde değerlendirilerek bu gruplar için haritalama yapılır. Burada k sembol başına iletilecek bit sayısı (bit/sembol) olarak isimlendirilir.

Bu çalışmada 16-DÖGM modülasyonu kullanılmıştır. 16-DÖGM'de $k = \log2(16)$'dan elde edilerek "4" olarak bulunur. Bu her sembol başına 4-bit'lik verinin iletileceğini ve modülasyon sırasında karmaşık eksene haritalanacak verilerin 4'er bitlik gruplar halinde olduğunu ifade etmektedir. Modülasyon tekniği olarak M-DÖGM seçilmesinin nedeni iki boyutlu haritalama yapılma özgürlüğüdür. M-PAM'da sadece genlik boyutunda haritalama yapılırken bütün semboller için farklı bir genlik seviyesi seçilmelidir. Bu da modülasyon seviyesi arttıkça ortalama gücü arttıracaktır. M-PSK'da ise sadece faz ekseninde haritalama yapılır genlik seviyeleri bütün semboller için eşittir. Bu da modülasyon seviyesi arttıkça işaret-yıldız kümesindeki noktaların birbirine yakınlaşmasına ve alıcıda hata yapılmasına neden olmaktadır. Modülasyon tekniklerinde bu haritalama işlemi yapılırken Gray kodlama kullanılmaktadır. Gray kodlama işaret-yıldız kümesindeki iki komşu sembol arasında bir bitlik değişim olarak haritalanmasıdır. Bu durumda alıcıda hata yapılması durumunda yalnızca bir bitlik hata yapılacaktır (Stüber 2000).

Şekil 4.5'te kullanılan 16-DÖGM için işaret-yıldız kümesi gösterilmektedir. Burada komşu semboller arası bir bitlik değişimin olduğu gözükmektedir. 4 farklı genlik ve 4 farklı faz seviyesi olan işaret yıldız kümesinde komşu semboller arası mesafe aynıdır.



Şekil 4.5 16-DÖGM Gray kodlamalı işaret-yıldız kümesi.

4.3.2 Hızlı Fourier Dönüşümü

TAFD ve AFD işlemleri karmaşık sayılar üzerinde uygulandığından işlem yükü olarak son derece yoğun matematiksel ifadelerdir. Özellikle bu işlemlerin bilgisayar gibi işlem kapasitesi oldukça yüksek platformlarda değil de gömülü platformlarda hesaplanacağı göz önüne alınırsa gerçekleştirilmeleri oldukça zor olacaktır. TAFD ve AFD işlemlerinin bu işlem yoğunluğundan dolayı verimli olarak hesaplanabilmeleri adına birçok algoritma geliştirilmiştir. Bunlardan bazıları, böl-yönet yaklaşımı, doğrusal süzme yaklaşımı ve hızlı fourier dönüşümü algoritmalarıdır (Ertürk 2009).

Bu tez çalışmasında TAFD ve AFD hesaplanırken Hızlı Fourier Dönüşümü (HFD, Fast Fourier Transform- FFT) yöntemi kullanılmıştır.

FFT işlemi AFD işleminin işlem karmaşıklığının azaltılarak daha hızlı gerçekleştirilmesini sağlar. FFT işlemi için literatürde birçok algoritma mevcuttur. Bu tez çalışmasında da kullanılan en yaygın algoritma ise Cooley-Tukey algoritmasıdır (Cooley and Tukey 1965). Bu

algoritmada temel amaç bazı matematiksel kısaltmalar ve işlem kolaylıkları sağlayarak N noktalı AFD işlemini 2 ve 4 noktalı AFD işlemlerini kullanarak gerçekleştirmektir. 2 noktalı AFD'ler gerçekleştirilen algoritmaya Taban-2/Kelebek-2 (Radix-2) algoritması adı verilmektedir. Bu tez çalışmasında Radix-2 algoritması kullanılmıştır. Radix-2 algoritmasını kullanabilmek için AFD si alınacak N uzunluklu giriş verisinin 2'nin kuvveti olması gerekmektedir ($N = 2^v, v$ bir pozitif tamsayı). Örneğin N uzunluklu $x[n]$ ayrık zaman işaretin FFT'sini almak için önce DFT ifadesi aşağıdaki gibi yazılır.

$$X[k] = \frac{1}{N}\sum_{n=0}^{N-1} x[n]e^{-\frac{j2\pi nk}{N}}, k = 0,1,2 \ldots., N \quad (4.10)$$

Burada verilen $e^{-\frac{j2\pi k}{N}}$ ifadesi faz faktörüdür ve W_N ile gösterilebilir. W_N katsayıları N ile periyodik ve N/2' ye göre simetriktir. N uzunluklu giriş dizisi çift ve tek olarak N/2 adet giriş dizisine ayrılır ve N/2 noktalı AFD hesaplanırsa

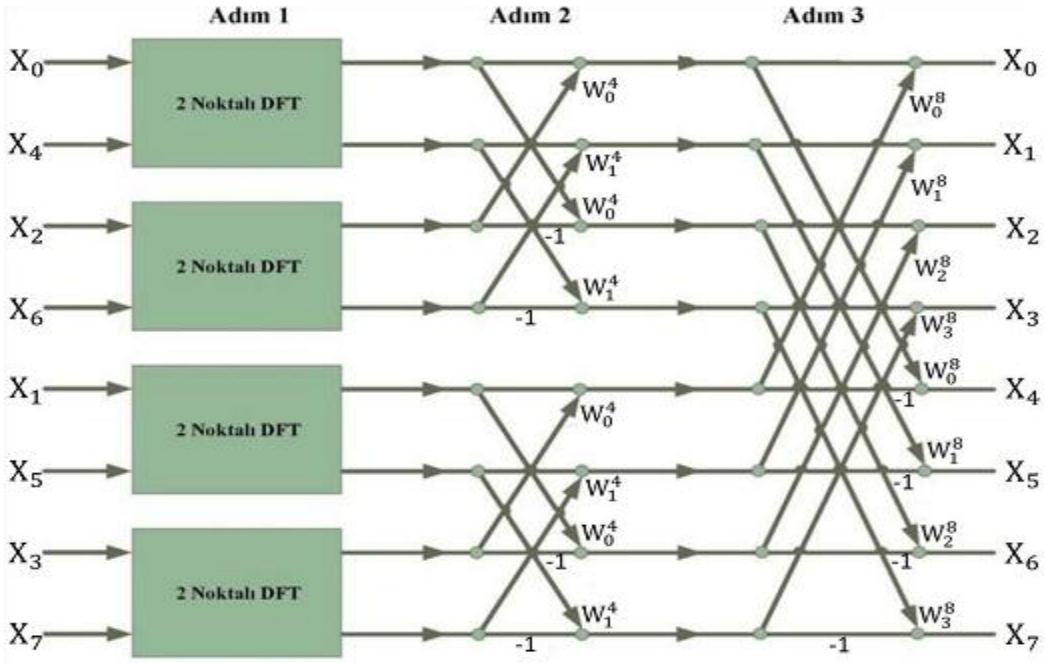
$$X[k] = \sum_{m=0}^{\frac{N}{2}-1} x(2m)W_{\frac{N}{2}}^{2mk} + \sum_{m=0}^{\frac{N}{2}-1} x(2m+1)W_{\frac{N}{2}}^{k(2m+1)} \quad (4.11)$$

olarak hesaplanır.$W_N^2 = W_{N/2}$ ifadesi yerine konulursa;

$$X[k] = F_1[k] + W_N^k F_2[k] \quad (4.12)$$

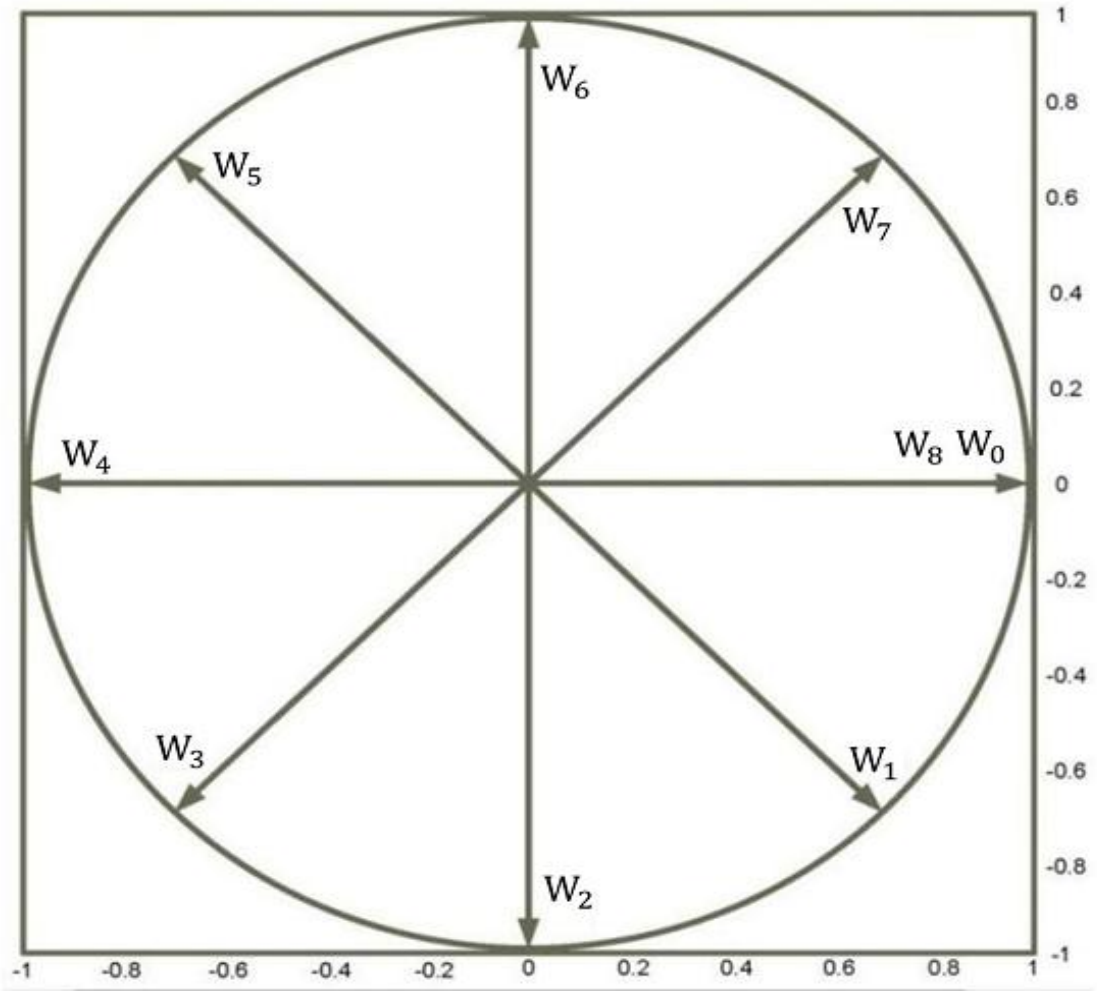
olacaktır. Burada $F_1[k]\ ve\ F_2[k]$ sırasıyla çift ve tek olarak ayrılan dizilerin N/2 noktalı ayrık fourier dönüşümleridir.

Bu işlem giriş veri dizileri tek noktalı hale gelinceye kadar tekrarlanabilir. Bu durumda yapılacak olan 2 noktalı AFD hesaplanması olacak ve çıkış dizilerinin faz faktörleri ile çarpılıp toplanarak birleştirilmesi işlemidir. Örnek olarak 8- noktalı AFD işleminin Radix-2 algoritması kullanılarak gerçekleştirilmesine ait diyagram Şekil 4.6'da verilmiştir. Bu yönteme kelebek (butterfly) yöntemi denir (Proakis and Monalakis 2007).



Şekil 4.6 8 noktalı AFD için Radix-2 işlem basamakları (Proakis and Monalakis 2007).

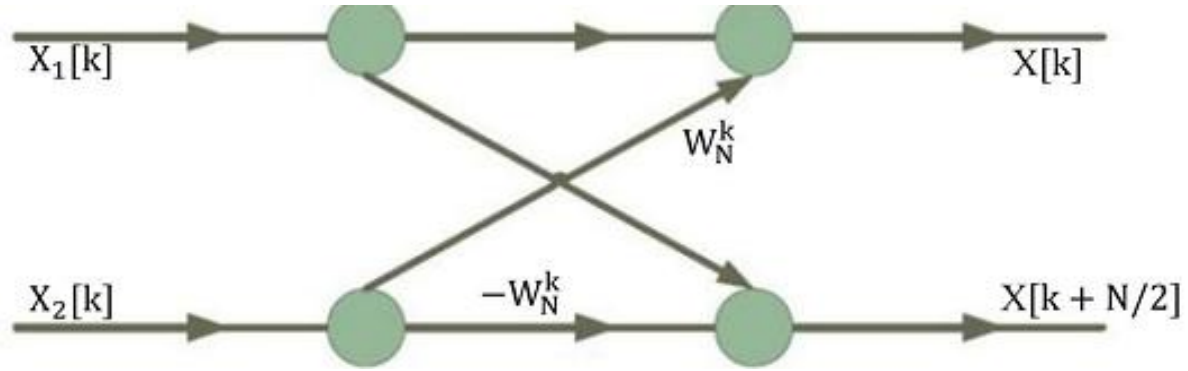
8-noktalı AFD işlem basamaklarında kullanılan faz faktörü katsayıları Şekil 4.7'de verilmiştir.



Şekil 4.7 8 noktalı FFT için faz faktörü katsayıları (Şahin 2010).

Şekil 4.7'de 8 noktalı AFD işlemi için oluşturulan W_N faz faktörü katsayılarının N=8 ile periyodik olduğu görülmektedir ($W_8=W_0$). N/2=4 sayısına göre simetrik oldukları gözükmektedir. Bu durumda 8 noktalı AFD işlemi için giriş dizisi tek ve çift veri dizilerine ayrılır ve N/2=4 noktalı AFD işlemleri hesaplanır. Çift diziler faz faktörü katsayıları ile çarpılarak 8 noktalı AFD işlemi gerçekleştirilmiş olur. 4 noktalı AFD işlemi içinse her iki veri dizisi de tekrar kendi arasında tek çift diye ayrılır ve bu dizler için 2 noktalı AFD işlemi hesaplanır. Bu durum Şekil 4.6'da gözükmektedir.

Temel kelebek yöntemi Şekil 4.8'de gösterilmiştir. 2 noktalı AFD işleminin hesaplanmasında W_N faz faktörü katsayıları "1" ve "-1" olacağından bu işlemi gerçekleştirmek oldukça kolaylaşacaktır.



Şekil 4.8 Temel Radix-2 yapısı.

4.3.3 Dairesel Ön Ek Eklenmesi/Çıkarılması

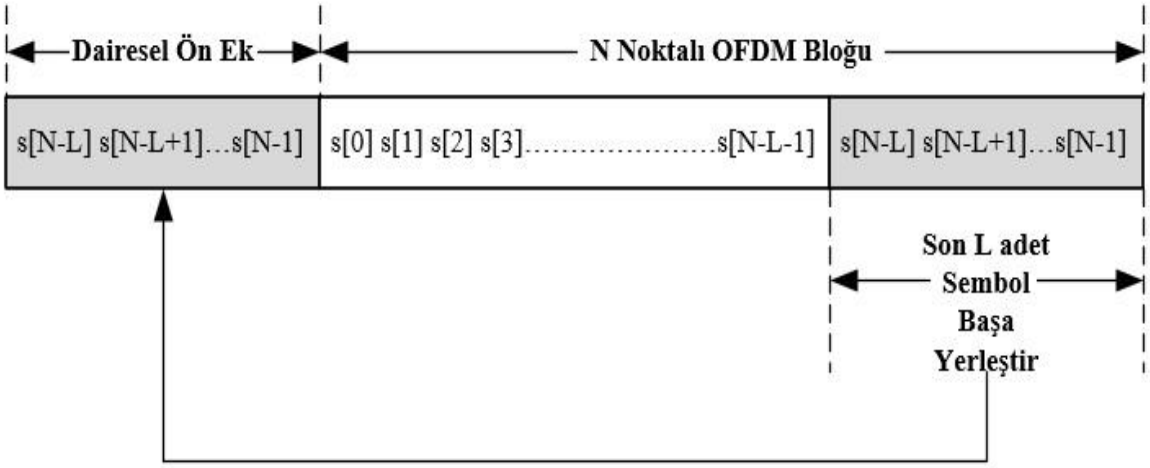
Kablosuz kanalda çok yollu etkiden dolayı alıcıya gelen sinyaller orijinal işaretin zamanda gecikmiş kopyalarının toplamına eşittir. OFDM'in alt bant genişliklerinin kanalın evre uyumlu bant genişliğinden küçük olacak şekilde ayarlanmasıyla kanalın frekans seçiciliği önlenir ve alıcıda frekans boyutunda tek tap'lı denkleştiriciyle kanal denkleştirme yapılır.

Ancak Şekil 4.4'te verilen OFDM verici-alıcı yapısında görüldüğü gibi alıcıda önce A/D çevrim yapılmaktadır. Bu durumda her bir alt taşıyıcıya karşılık gelen düz sönümlemeli kanal tepkisi $h(t)$, ayrık dürtü cevabı L uzunluklu $h[n]$'ye dönüşmektedir.

OFDM sembollerinde işaret alt taşıyıcılara bölündüğünden kanalın ayrık dürtü cevabı h[n], alt taşıyıcı süresince ayrık doğrusal zamanla değişmeyen sistem (Linear Time-Invariant) olarak ele alınabilir. Bu durumda kanal çıkışında alıcıda elde edilecek olan ayrık işaret y[n];

$$y[n] = s[n] * h[n] \tag{4.13}$$

olarak elde edilir. Burada s[n] kanal girişindeki ayrık OFDM sembolü, * matematiksel sembolü ise doğrusal konvolüsyon işlemidir. Dolayısıyla alıcıda SAK oluşmaması adına kanal uzunluğu olan L'nin dikkate alınması gereklidir. IFFT sonucunda oluşan sembollere L kanal uzunluğunda bir dairesel ön ek (cyclic prefix- CP) eklenerek bu problemin önüne geçilmiş olur. Eklenen dairesel ön ek N uzunluğundaki OFDM alt taşıyıcısının son L adet verisinden oluşmaktadır (Şekil 4.10).



Şekil 4.10 N uzunluğundaki sembole L uzunluğunda CP eklenmesi.

Dairesel ön ek eklenerek elde edilen yeni OFDM alt sembolü $\tilde{s}[n]$ olsun, bu durumda kanalın etkisine maruz kalarak alıcıda alınan işaret $\tilde{y}[n]$;

$$\tilde{y}[n] = \tilde{s}[n] * h[n] \tag{4.14}$$

olarak ifade edilir. Denklem (4.14)'te elde edilen $\tilde{y}[n]$ giriş sembolünün dairesel ön ek eklenmemiş hali olan $s[n]$ ile kanalın dürtü cevabı $h[n]$' nin dairesel konvolüsyonuna eşittir (4.15). Böylece alıcıda elde edilen $\tilde{y}[n]$'den ilk L adet sembol çıkarılarak $s[n]$ elde edilebilir.

$$\tilde{y}[n] = s[n] \otimes h[n] \tag{4.15}$$

$\otimes$ dairesel konvolüsyon işlemidir.

CP eklenerek SAK'ın önüne geçilmiş olur. Ancak veri hızını olumsuz yönde etkileyecektir. Bir N+L uzunluğundaki sembol boyunca N adet anlamlı veri gönderilmektedir. Bu durum verimin düşmesine sebep olacaktır.

Alıcıda DFT işleminden sonra, alınan işaret frekans boyutunda

$$\tilde{Y}(f) = S(f) \,.\, H(f) \tag{4.16}$$

şeklindedir. Kanalın alıcıda bilinmesi durumunda her bir sembol için tek tapl'lı denkleştirme ile iletilen işaretin kestirimi gürültüsüz bir kanalda

$$\tilde{S}(f) = \frac{\tilde{Y}(f)}{H(f)} = S(f) \tag{4.17}$$

olarak elde edilir.

4.4 OFDM AVANTAJ VE DEZAVANTAJLARI

OFDM tekniğinin avantajları aşağıda sıralanmıştır.

Toplam hız sabit olmak üzere veriler daha düşük hızlı alt taşıyıcılar ile gönderilmektedir.

Alt taşıyıcıların birbirine dik seçilmesi ile örtüşmeleri sağlanarak daha az bir bant genişliği kullanılır. Bant verimliği arttırılmış olur. Alt taşıyıcılar için sembol süresinin artması ile sistem frekans seçici sönümlemelere karşı dirençli hale gelir. CP kullanılmasıyla SAK ortadan kaldırılır. TAFD ve AFD sayısal teknikleri kullanılarak çok taşıyıcılı sistemlerin karmaşıklığından ve maliyetinden kaçınılır ve sistem daha kolay uygulanabilir hale gelir.

OFDM tekniğinin avantajlarının yanında bazı dezavantajları da bulunmaktadır. Bunlar da aşağıda verilmiştir.

Alıcıda senkronizasyon problemi doğabilmektedir. Sistemin taşıyıcı frekans kaymalarına karşı direnci zayıftır. Sistemde kullanılan TAFD'nin çıkışında yüksek bir işaret elde edilmesine bağlı olarak yüksek tepe ortalama güç oranı (PAPR) mevcuttur. Kullanılan yükselteçlerin doğrusal olmaması durumunda dikgenlik bozulabilmektedir. Dolayısıyla doğrusal yükselteçlerle kullanılmalıdır.

BÖLÜM 5

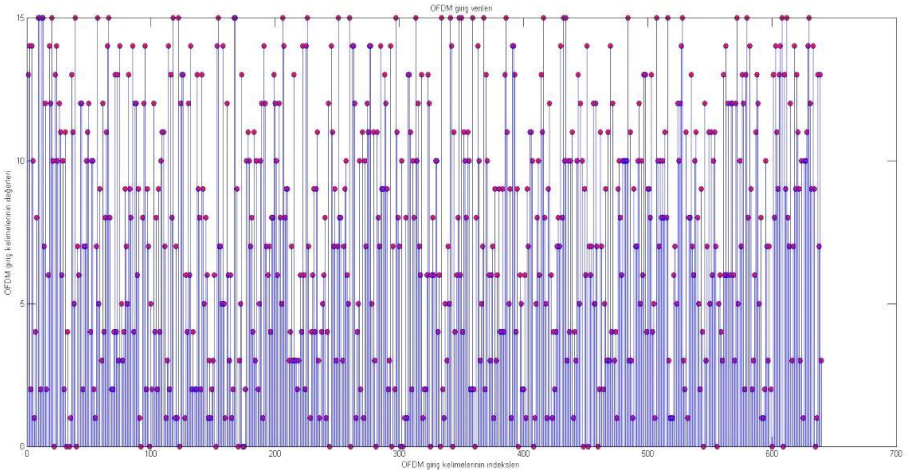
OFDM BLOKLARININ GERÇEKLENMESİ

5.1 GİRİŞ

OFDM teknolojisinin alıcı/verici sistemini gerçekleştirme adına verici kısmında bulunan Modülasyon ve TAFD blokları ile alıcı kısmında bulunan AFD ve Demodülasyon bloklarının VHDL kodları yazılarak sentezlemeleri yapılmıştır. Seri/Paralel ve Paralel/Seri dönüşüm blokları için ayrı bloklar yazılması yerine modülasyon-TAFD ve AFD-demodülasyon blokları geçişi sırasında bu işlemler gerçekleştirilmiştir. Yazılan VHDL kod bloklarının testlerinin yapılması adına her bloğun Modelsim benzetimleri oluşturulmuş ve bu benzetim sonuçları MATLAB ortamında gerçekleştirilen sonuçlarla karşılaştırılmıştır. Modelsim benzetimleri için "Modelsim Altera 10.1" kullanılmıştır. Benzetimleri yapılan blokların sentezlenmesi amacıyla "Quartus II 13.1" kullanılmıştır.

Verici-Alıcı tüm sistemin testinin gerçekleştirilmesi adına 10 adet alt taşıyıcı için 4-bit'lik kelimelerden oluşan 640 uzunluğundaki veri dizisi kullanılmıştır. Şekil 5.1'de gösterilen bu veri dizisi MATLAB ortamında üretilen düzgün dağılımlı rastgele değişkenlerden oluşmaktadır. Gerçekleştirilen her bir bloğun Modelsim benzetimlerinden ve sonra çıkış verileri MATLAB ortamına alınarak karşılaştırılmaları yapılmıştır.

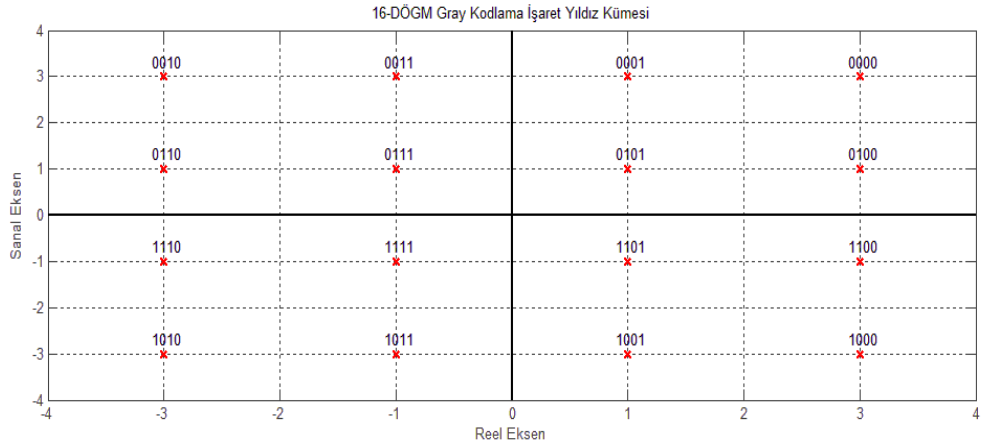
FPGA üzerinde yapılacak işlemler için sayı formatı olarak Elektrik Elektronik Mühendisleri Enstitüsü (Institute of Electrical Electronics Engineering-IEEE)' tarafından belirlenmiş 32-bit kayan noktalı (floating point) sayı formatı standardı seçilmiştir. Bunun sebebi kayan noktalı sayı formatında işlemlerin sabit noktalı (fixed point) sayı formatına göre daha hassas olarak yapılabilmesidir. Sayı formatının belirlenmesinde bir diğer faktör ise; Bölüm 5.3'te anlatılacak TAFD/AFD bloklarının 32-bit kayan noktalı sayı formatında çalışmasıdır. Sayı formatı dönüşümü yapmak yerine diğer bloklar da bu sayı formatında tasarlanmıştır.



Şekil 5.1 MATLAB ortamında oluşturulan giriş veri dizisi.

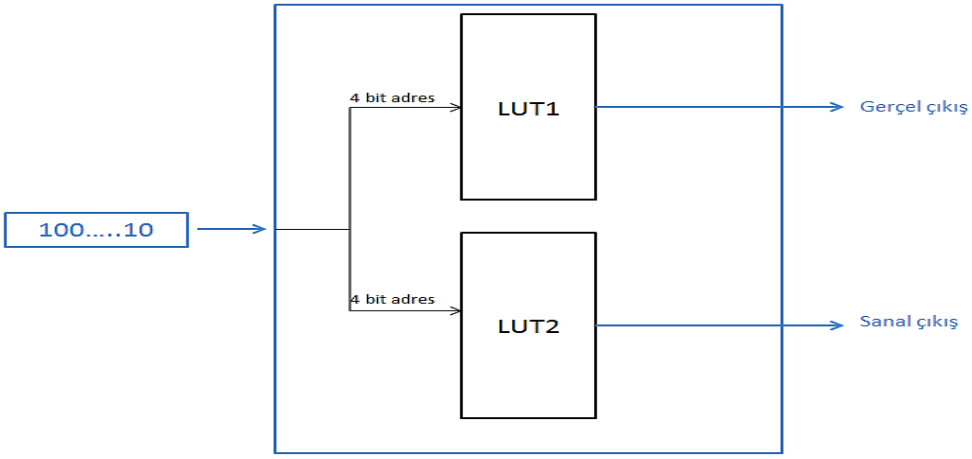
5.2 16-DÖGM (DÖRDÜN GENLİK MODÜLASYONU)'NİN GERÇEKLENMESİ

OFDM mimarisine tabi tutulacak sayısal verinin öncelikle temel bant modülasyon tekniklerinden biri ile modüle edilmesi gerekmektedir. Bu tez çalışmasında modülasyon çeşidi olarak 16-DÖGM (Dördün Genlik Modülasyonu – Quadrature Amplitude Modulation-QAM) seçilmiştir. DÖGM 'de modülasyon bloğu girişine gelen semboller (symbol) analitik düzlemde haritalanır. Bu düzlem x reel (genlik -amplitude) ve y sanal(faz -in-phase) eksenlerinden oluşmaktadır. Bu haritalama sırasında her bir sembol için farklı bir nokta belirlenmelidir. M-QAM modülasyonun da toplam M adet nokta bulunur iken, bu noktalara haritalanacak sembol uzunlukları $log2(M) = k$ bit'lik verilerden oluşmaktadır. Bu haritalama sonucu oluşturulan düzleme işaret-yıldız kümesi adı verilmektedir. 16-DÖGM için semboller 4 bit'ten oluşmaktadır ve işaret-yıldız kümesi 16 adet nokta içermektedir. İşaret-yıldız kümesi oluşturulurken Gray kodlama kullanılarak komşu semboller arasında yalnızca bir bitlik fark olması sağlanır. Böylece modülasyon-demodülasyon sonucunda oluşabilecek hatalar minimize edilmeye çalışılır. 16-DÖGM için Gray kodlama kullanılarak her 4 bit'lik sembolün farklı bir değere haritalandığı işaret-yıldız kümesi Şekil 5.2'de gösterilmiştir.



Şekil 5.2 16-DÖGM Gray kodlamalı işaret-yıldız kümesi.

İşaret-yıldız kümesine göre her bir sembolün haritalandığı noktanın değerlerinin bilinmesi gerekecektir. Bunun için gerçel ve sanal değerleri tutacak 2 adet doğruluk tablosu (LUT) bloğunun VHDL kodu yazılmıştır. LUT bloklarının oluşturulması için FPGA içerisinde 2 adet sabit hafıza (constant memory) blokları oluşturulmuştur. Bu bloklar içerisinde işaret-yıldız kümesindeki değerlerin 32-bit kayan noktalı (floating point) karşılıkları tutulmaktadır ve bu değerlerin tutuldukları adresler giriş sembollerinin değerleri olarak belirlenmiştir. 16-DÖGM bloğuna gelen 4-bit'lik semboller LUT bloklarına adres verisi olarak verilip, o adreslerdeki 32-bit'lik değerler 16-DÖGM bloğunun çıkışı olarak aktarılmıştır (Şekil 5.3). Böylece modülasyon bloğuna girilen her bir 4-bit'lik sembol, işaret-yıldız kümesindeki karşılığına göre haritalanmış ve çıkışa aktarılmıştır. LUT blokları içerisinde tutulan işaret-yıldız kümesi değerlerinin 32-bit kayan noktalı gösterimi 16lık tabanda (hexadecimal) olarak verilmiştir (Çizelge 5.1)

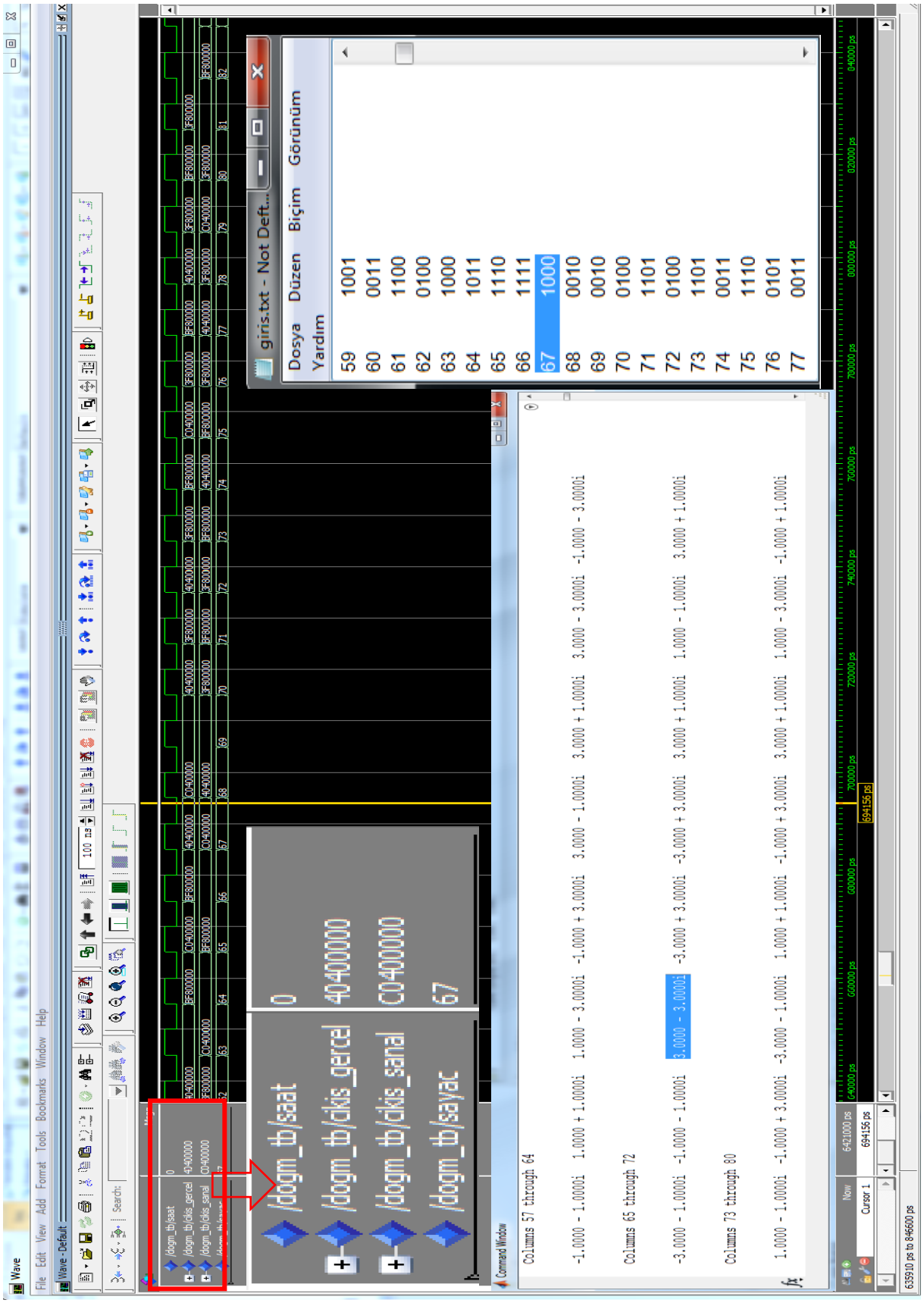


Şekil 5.3 16 DÖGM VHDL tasarımının blok gösterimi.

Çizelge 5.1 16-DÖGM için 4bitlik sembollerin adreslerinde tutulan 32-bit kayan noktalı sayı formatındaki işaret yıldız kümesi değerlerinin 16'lık tabanda gösterimi.

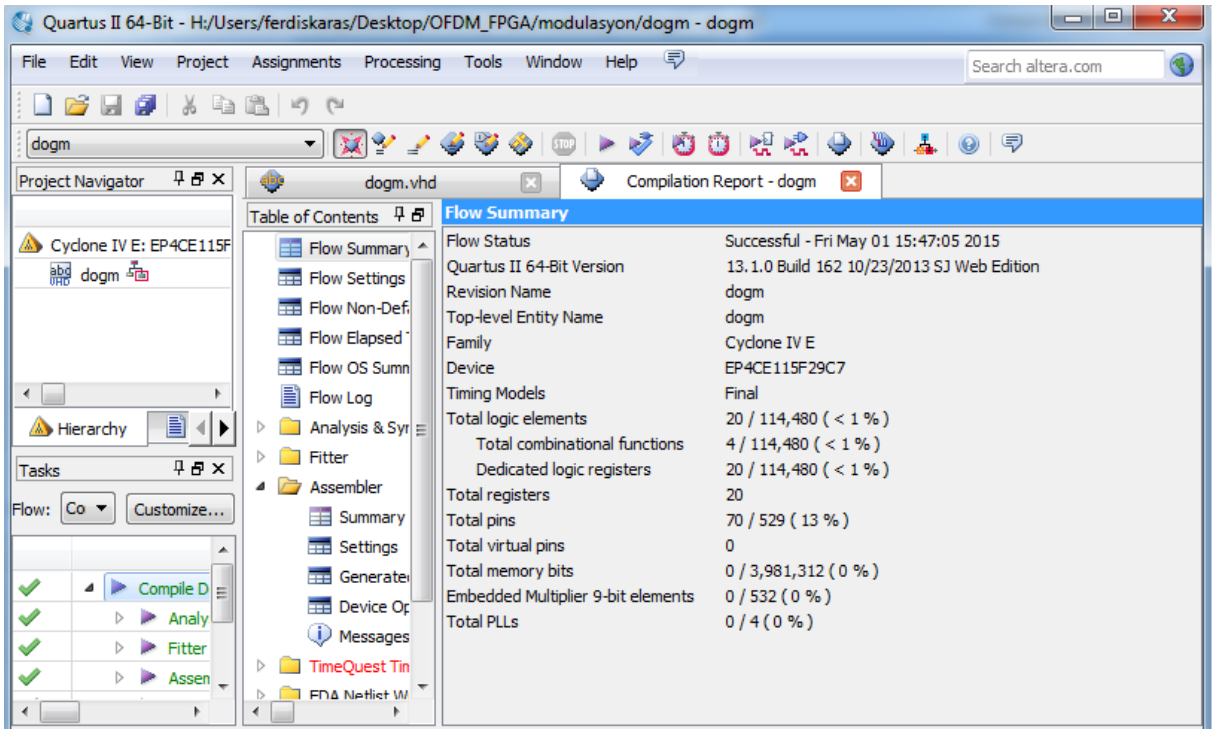
Sembol(Adres)	LUT1 (Gerçel Eksen)	LUT2 (Sanal Eksen)
0000	40400000	40400000
0001	3F800000	40400000
0010	C0400000	40400000
0011	BF800000	40400000
0100	40400000	3F800000
0101	3F800000	3F800000
0110	C0400000	3F800000
0111	BF800000	3F800000
1000	40400000	C0400000
1001	3F800000	C0400000
1010	C0400000	C0400000
1011	BF800000	C0400000
1100	40400000	BF800000
1101	3F800000	BF800000
1110	C0400000	BF800000
1111	BF800000	BF800000

VHDL olarak tasarımı yapılan modülasyon bloğunun testini yapabilmek için yazılan test dosyası (testbench) "Modelsim 10.1" kullanılarak test edilmiş ve bit seviyesi sonuçlar Şekil 5.4'te gösterilmiştir. Ayrıca testlerin doğru çalıştığını görmek amacıyla MATLAB da bulunun "qammod" kodu kullanılarak karşılaştırmalar yapılmıştır. Örneğin; modülasyon bloğuna giriş olarak verilen "67" numaralı sembol için tasarlanan modülasyon bloğu sırasıyla gerçel ve sanal kısım için "40400000" ve "C0400000" çıkışlarını vermektedir. Giriş verilerinin tutulduğu "giriş.txt" dosyasında "67" numaralı sembolün "1000" olduğu ve bu sembolün adresinde gerçel ve sanal kısımlar için "40400000" ve "C0400000" verilerinin saklandığı Çizelge 5.1'den görülmektedir. Ayrıca MATLAB'da çalıştırılan "qammod" kodu sonucunda "67" numaralı sembol için "-3+3i" çıkışı gözlenmiştir. Bu sonuç da 32-bit kayan noktalı sayı formatına dönüştürüldüğünde sonuçların örtüştüğü gözükmektedir



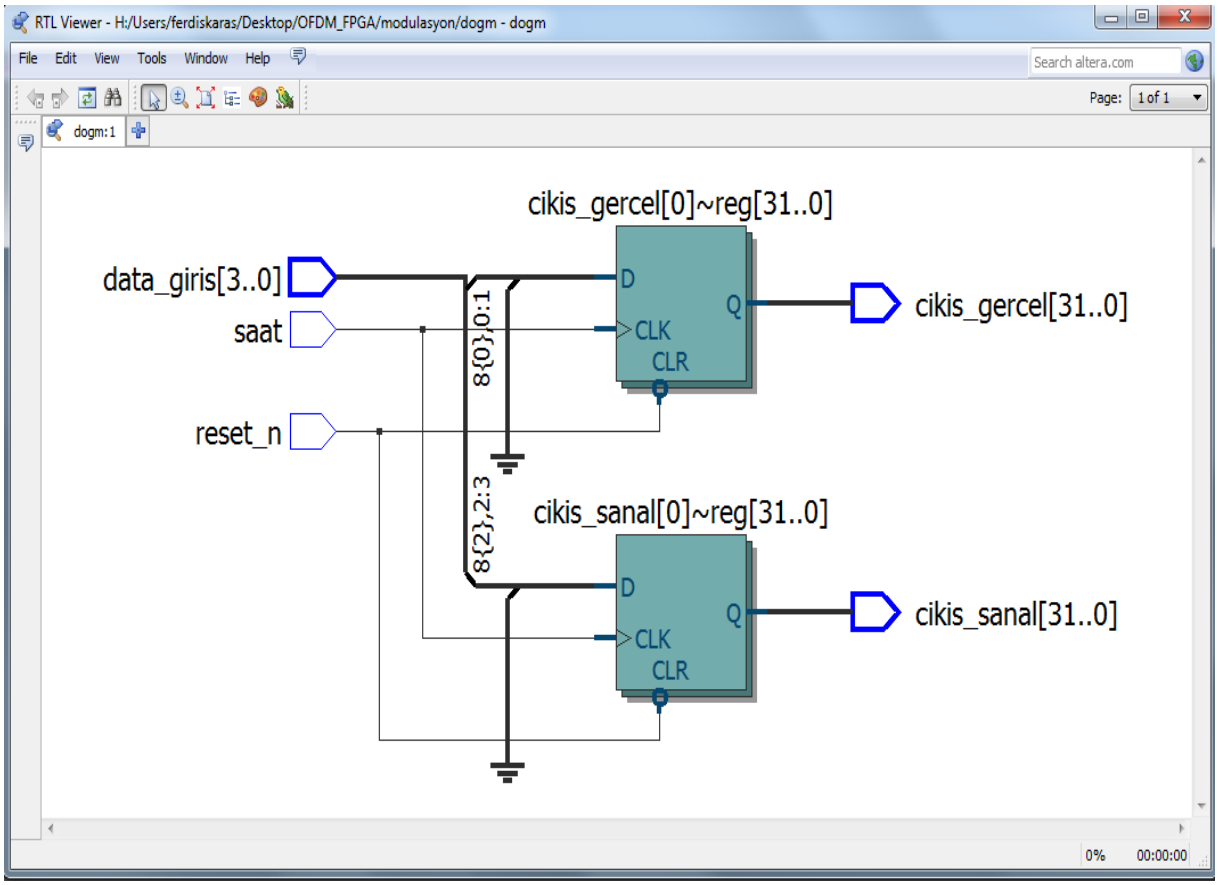
Şekil 5.4 16-DÖGM bloğunun Modelsim benzetimine ve MATLAB sonuçlarına ait ekran görüntüsü.

Modelsim benzetimi ile MATLAB sonuçları ile örtüşmesi sonrası 16-DÖGM bloğu "Quartus 13.1" kullanılarak derlenmiş ve başarılı olan derleme sonucuna ait ekran görüntüsü Şekil 5.5'te verilmiştir. Sentezleme sonucu oluşan makine kodu kullanılan FPGA üzerine yüklenerek modülasyon bloğunun gerçeklenmesi yapılmıştır.

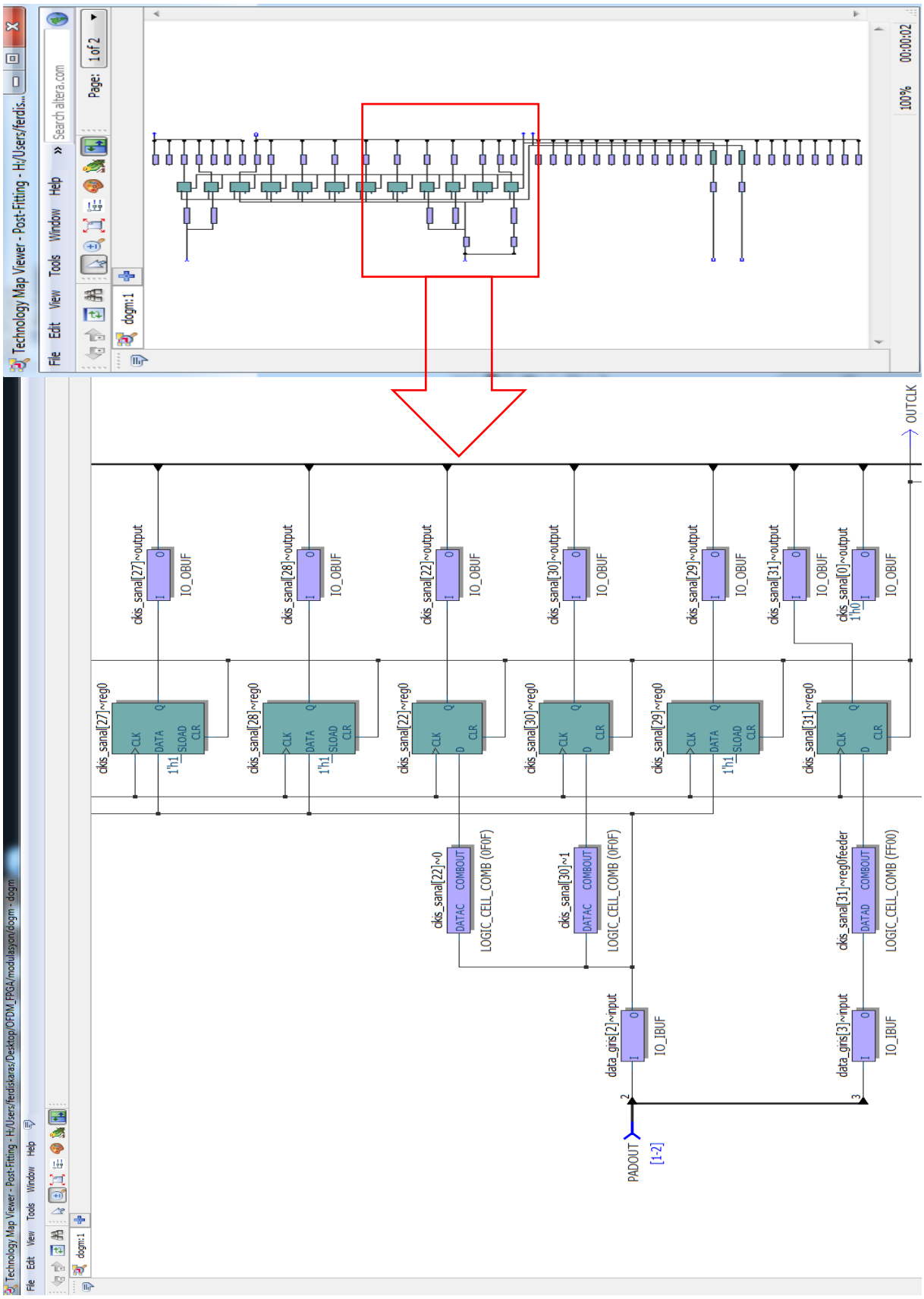


Şekil 5.5 16-DÖGM bloğunun Quartus projesine ait ekran görüntüsü.

16-DÖGM bloğunun sentezlenmesi sonucu oluşturulan devrenin sadece 20 adet Mantık Elemanı ile gerçekleştirilebileceği gözükmektedir. Bu da Cyclone IV ailesine ait 4CE11529C7 kodlu FPGA'nın kaynaklarının %1'den bile daha azını kullanıldığını göstermektedir. Şekil 5.6'da sentezlenen bloğun saklayıcı transfer seviyesi (RTL) devre şeması gösterilmiştir. Şekil 5.7'de ise sentezleme ve optimizasyon sonucu oluşan kapı seviyesi (gate level) devre şeması verilmiştir.



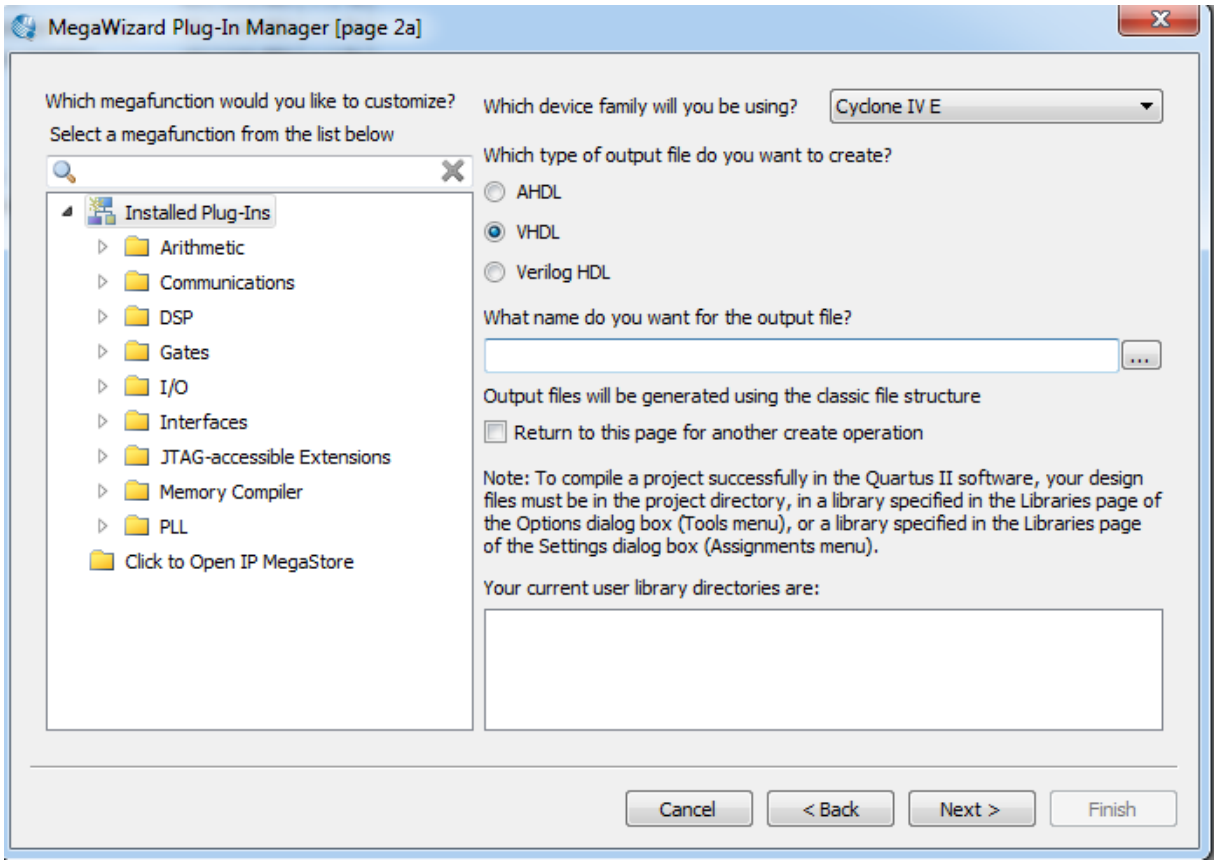
Şekil 5.6 16 DÖGM bloğunun sentezleme sonucu oluşan RTL şemasına ait ekran görüntüsü.



Şekil 5.7 Quartus ile sentezlenen 16-DÖGM bloğu donanımına ait ekran görüntüsü.

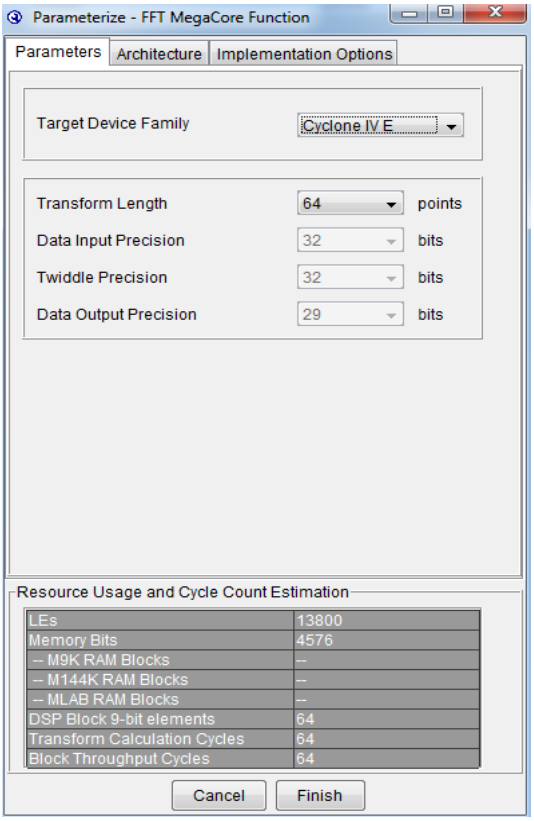
5.3 TAFD/AFD BLOKLARININ GERÇEKLENMESİ

OFDM mimarisinin bir sonraki adımı modülasyon bloğu sonucunda elde edilen karmaşık verilerin (gerçel ve sanal değerler) belirlenen alt-taşıyıcı uzunluğunda TAFD'sinin alınmasıdır. Bu Ters Ayrık Fourier Dönüşümleri her bir alt taşıyıcı için hesaplanmalı ve çıkışlar bir sonraki bloğa gönderilmelidir. TAFD matematiksel olarak son derece yoğun işlemlerin olduğu bir bloktur. Bu nedenle TAFD bloğunun VHDL kodunun yazılması yerine "Quartus MegaWizard Plug-In Manager" aracının sunduğu "FFT MegaCore" kütüphanesi kullanılmıştır. Bu tezde alt-taşıyıcı uzunluğu N=64 olarak seçilmiş ve FFT MegaCore kütüphanesinden buna göre "MegaFunction" oluşturulmuştur. "MegaWizard Plug-In Manager" aracının kullanıcının ihtiyacının olduğu kütüphaneyi seçmesine olanak sağlayan ara yüz Şekil 5.8'de gösterilmiştir. Bu ara yüzden "DSP" seçeneğinin altından "FFT 13.0" fonksiyonu seçilmiştir. Bu seçim sonrası açılan "FFT Megacore Function" 'a ait ara yüz ve kullanıcının parametreleri girebildiği ara yüzler Şekil 5.9'da gösterilmiştir.

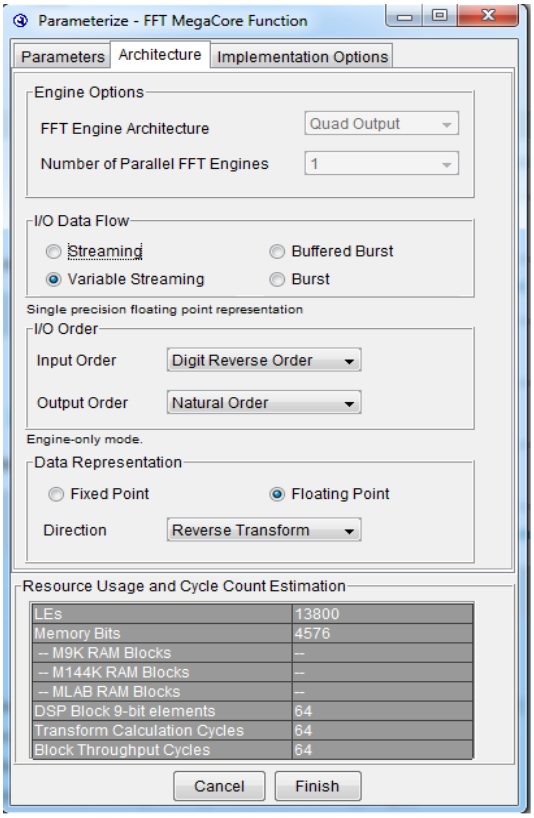


Şekil 5.8 MegaWizard Plug-In Manager'a ait ekran görüntüsü.

a) FFT Megacore kullanıcı ara yüzü



b)IFFT uzunluğunun ayarlandığı ara yüz

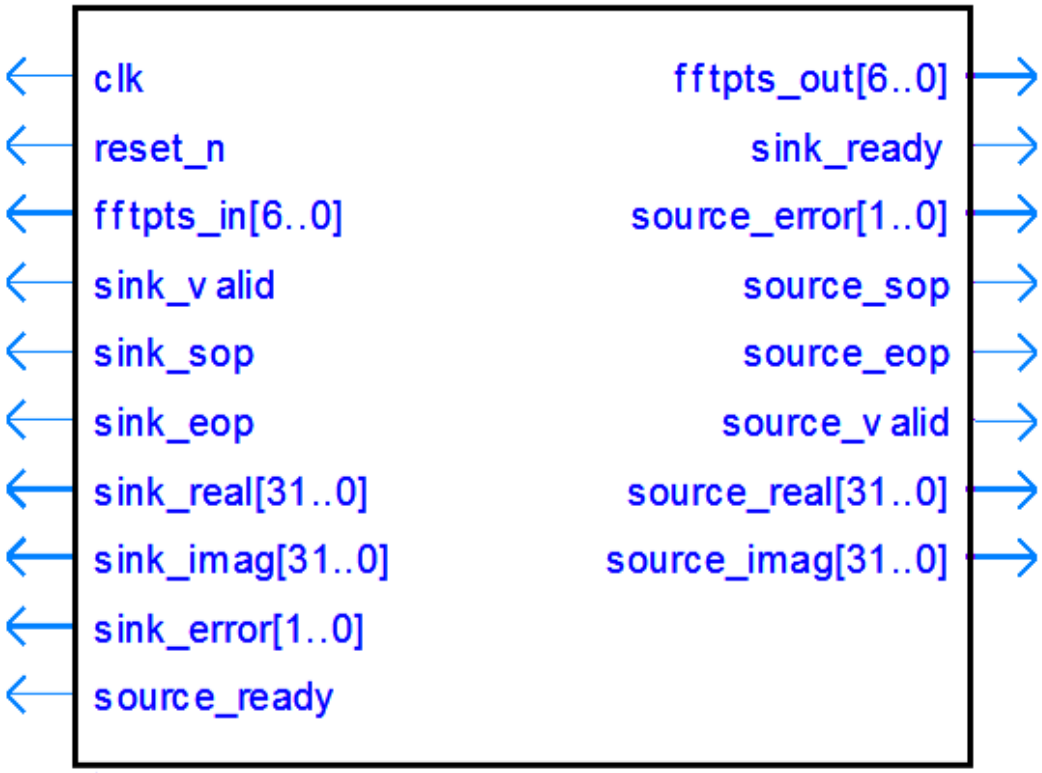


c)Mimarinin ve sayı formatının ayarlandığı ara yüz

Şekil 5.9 FFT Megacore Function'a ait kullanıcı ara yüzleri.

Şekil 5.9.a'daki resimden "Parameterize" seçeneği tıklanarak açılan pencerede Şekil 5.9.b'deki "Parameters" sekmesinde hedef FPGA cihazı seçilerek dönüşüm uzunluğu "64" olarak belirlenmiştir. Şekil 5.9.c'deki "Architecture" sekmesinden kullanılan sayı formatı "Floating Point" dönüşüm yönü "Inverse Transform" olarak ayarlandıktan sonra Şekil 5.9.a'daki resimden "Generate" seçeneği tıklanarak IFFT fonksiyonu oluşturulmuştur.

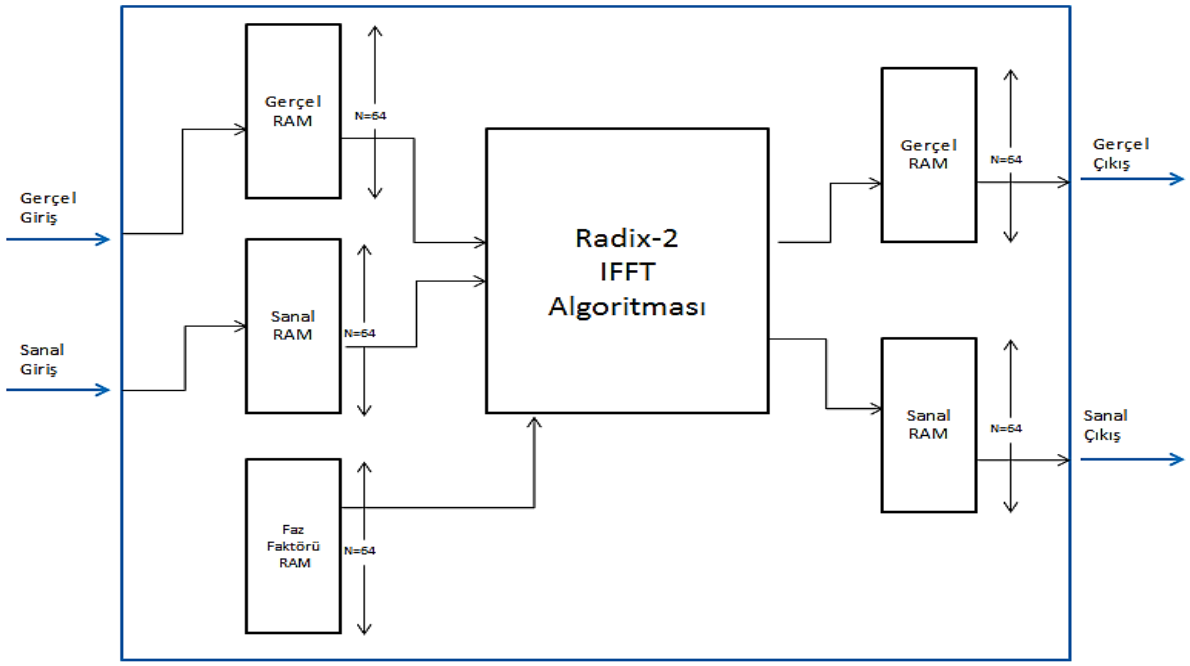
FFT Megacore Function "Avalon-ST" adı verilen bir ara yüz oluşturmakta ve IFFT/FFT fonksiyonunun çalıştırılması için bu ara yüzdeki giriş (input) sinyallerinin doğru bir şekilde sürülmesi gerekmektedir. Oluşturulan ara yüz Şekil 5.10'da verilmiştir.



Şekil 5.10 Avalon-ST ara yüzü.

Oluşturulan Avalon-ST ara yüzünün içyapısını gösteren blok şeması Şekil 5.11'de verilmiştir. Burada en dıştaki blok Avalon-ST ara yüzüdür ve sadece giriş ve çıkış veri setlerinin portları gösterilmiştir. Ara yüz veri girişlerinden gelen gerçel ve sanal veriler öncelikli olarak RAM bloklarına yazılır. Daha sonra RAM bloklarından okunan veriler faz faktörü katsayılarının tutulduğu RAM bloğundan gelen veriler ile birlikte Bölüm 4.3.2'de anlatılan Kelebek-2

algoritmasına göre IFFT hesaplanır ve sonuçlar çıkış katmanındaki RAM bloklarına yazıldıktan sonra çıkış portlarına aktarılır.



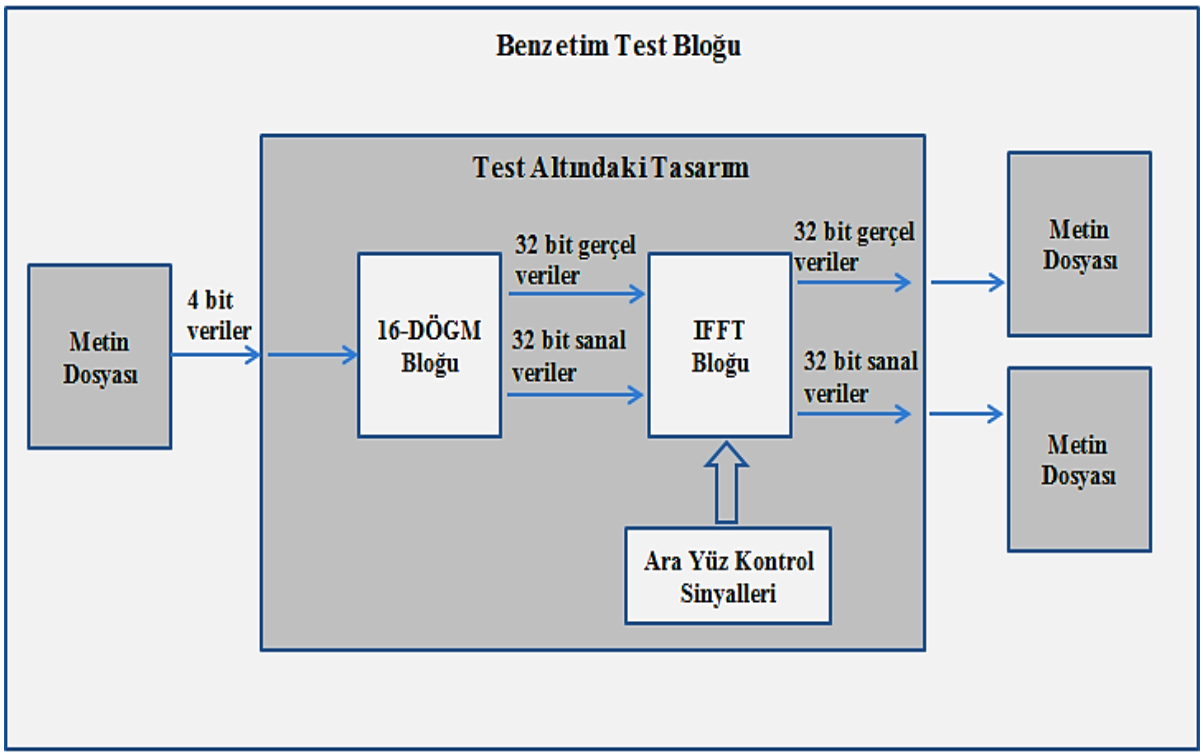
Şekil 5.11 Megacore IFFT Function mimarisini gösteren blok şeması.

Avalon-ST ara yüzünde bulunan "sink_sop" girişi IFFT bloğuna gelen verilerin yeni bir çerçeve (frame) başlangıcı olduğunu bildiren sinyaldir, "sink_eop" girişi ise IFFT'ye gelen giriş verisinin bir önceki çerçevenin son verisi olduğunu haber veren sinyaldir. Dolayısıyla özellikle bu iki giriş sinyalinin IFFT'nin doğru alınabilmesi için doğru zamanda sürülmesi son derece önem taşımaktadır. IFFT fonksiyonu çalıştırılır iken "sink_real" ve "sink_imag" girişlerini süren modülasyon bloğunun sırasıyla "dogm_cikis_gercel" ve "dogm_cikis_sanal" çıkışlarının kaçıncı veri olduğu kontrol edilir ve bu bilgi "sayac_giris" sinyalinde tutulur. "sayac_giris" sinyali "0" değerinde iken "sink_sop" sinyali tetiklenir ve "sayac_giris" sinyali IFFT uzunluğunun bir eksiği olan "63" değerini aldığında "sink_eop" sinyali tetiklenerek "sayac_giris" sinyali tekrar "0" a çekilir ve "frame_in_index" değeri bir arttırılır. Bu değer IFFT bloğuna kaçıncı çerçevenin gönderildiği değerini tutmaktadır.

Avalon-ST ara yüzünün çıkış katmanında IFFT fonksiyonu tarafından üretilen "source_valid", "source_sop" ve "source_eop" sinyalleri sırasıyla bir sonraki bloğa IFFT işleminin sonuçlandığını, yeni çerçevenin başlangıcını ve çerçeve bitişini bildiren sinyallerdir.

Bu sinyaller kontrol edilerek blok çıkışına kaçıncı çerçevenin aktarıldığı bilgisi "frame_out_index" sinyalinde ve çıkıştaki sinyalin o çerçevenin kaçıncı sinyali olduğu bilgisi ise "sayac_cikis" sinyalinde tutulur.

IFFT bloğunun fonksiyonel testini yapmak amacıyla oluşturulan OFDM verici kısmının Modelsim benzetimine ait blok şeması Şekil 5.12'de verilmiştir. MATLAB'da oluşturulan rastgele veri dizisi metin dosyasından okunup modülasyona tabi tutulduktan sonra modülasyon çıkışları IFFT bloğunun giriş veri setleri olarak kullanılmış ve IFFT çıkışları sanal ve gerçel olmak üzere iki farklı metin dosyasına yazılmıştır. Benzetim için oluşturulan test dosyası (testbench) tezin sonunda bulunan ekler kısmında verilmiştir.



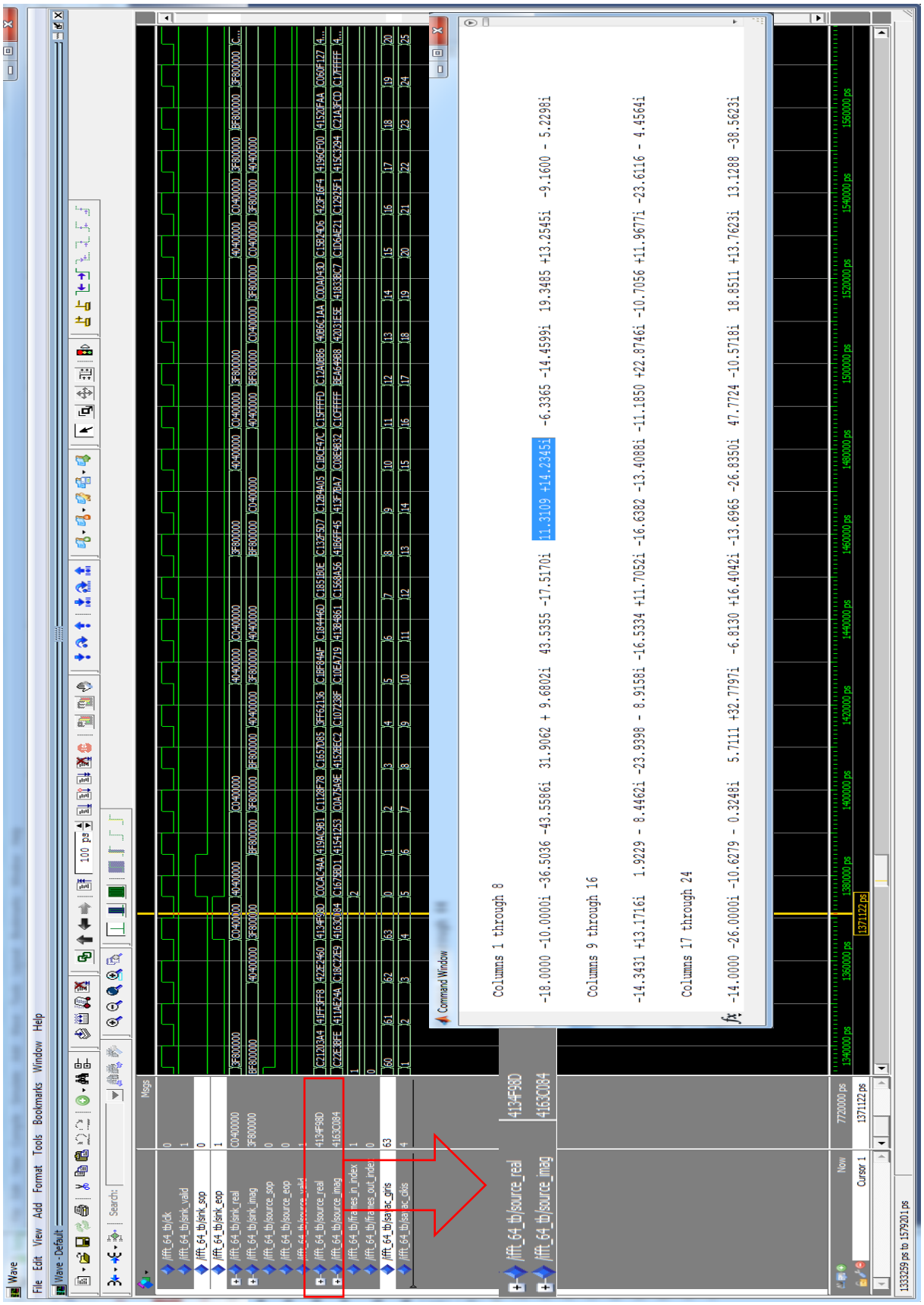
Şekil 5.12 OFDM verici kısmının Modelsim benzetimini gösteren blok şeması.

Şekil 5.12' de blok şeması verilen Modelsim benzetim sonuçları Şekil 5.13 ve Şekil 5.14'te gösterilmiştir. Şekil 5.13'te "sayac_giris" değerinin "63" e ulaşması durumunda "sink_eop" sinyalinin tetiklendiği, bir sonraki saat sinyalinde ise "sayac_giris" sinyalinin "0" a çekilerek "frame_in_index" sinyalinin bir arttırılarak "sink_sop" sinyalinin tetiklendiği gözükmektedir. Şekil 5.14'te ise "source_eop" sinyalinin üretildiği anda "sayac_cikis" sinyalinin "63" değerinde olduğu gözükmektedir. "source_sop" sinyalinin tetiklenmesi ile "sayac_cikis"

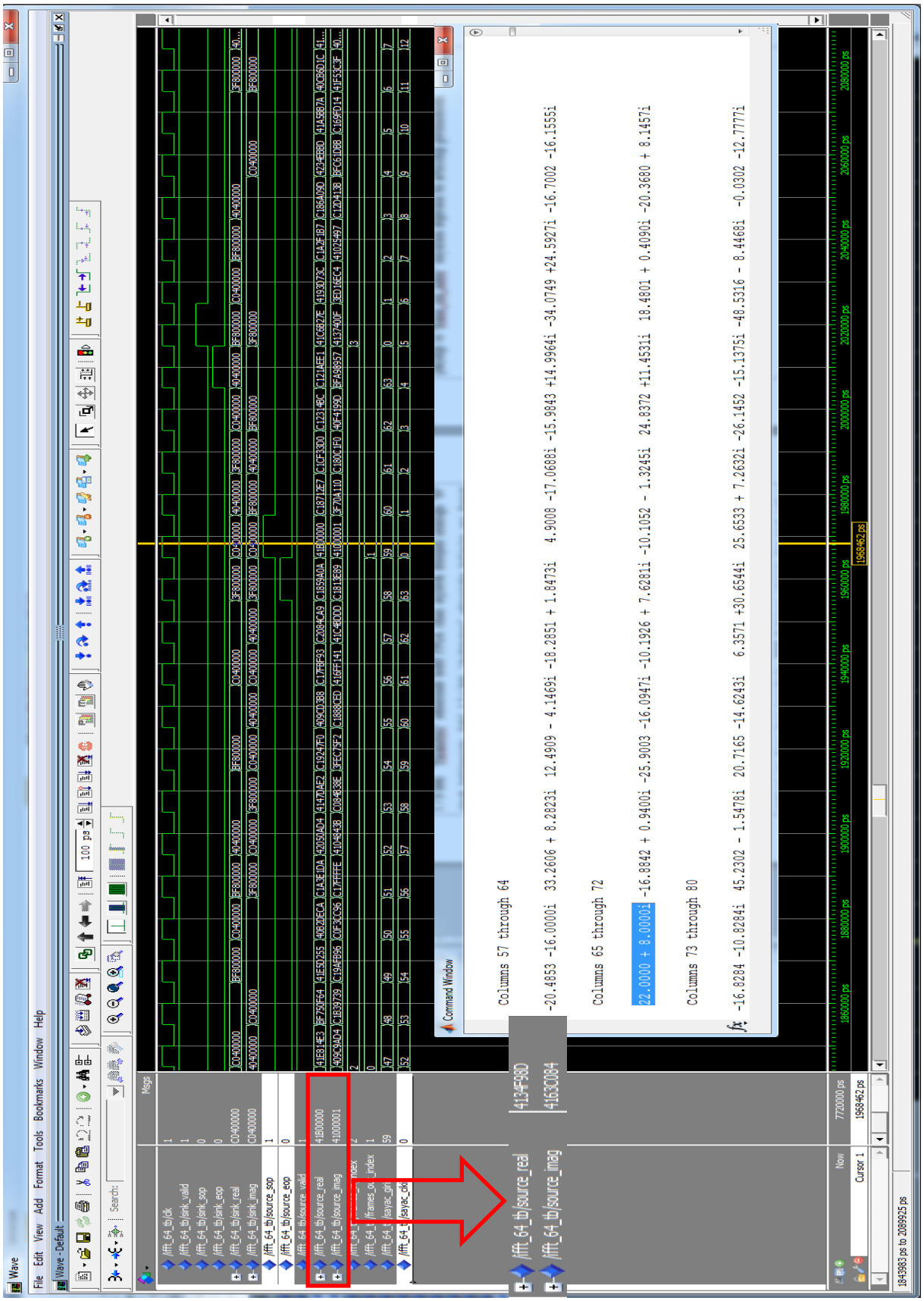
sinyalinin “0” a çekildiği ve “frame_out_index” sinyalinin değerinin bir arttırıldığı görülmektedir.

Modelsim benzetimlerindeki sonuçların doğruluğunu test amacıyla MATLAB da hesaplanan IFFT sonuçları da Şekil 5.13 ve 5.14’te gözükmektedir. Şekil 5.13’teki benzetimde “frame_out_index” sinyalinin değerinin “0”, “sayac_cikis” sinyalinin değerinin “4” olduğu durumdaki gerçel ve sanal çıkış değerlerinin sırasıyla “4134F98D” ve “4163C084” olduğu gözükmektedir. Bu değerler MATLAB ortamında gerçekleştirilen hesaplama sonucu oluşan 5. değerin 32-bit kayan noktalı sayı formatının 16’lık tabanda gösterimine karşılık gelmektedirler. Bir başka örnek ise; Şekil 5.14’teki benzetimde “frame_out_index” sinyalinin değerinin “1”, “sayac_cikis” sinyalinin değerinin “0” olduğu durumdaki gerçel ve sanal çıkış değerleri olan “41B00000” ve “41000001”, MATLAB ortamında gerçekleştirilen hesaplamanın “65” inci değeri olan “22.0000 + 8.0000i” karmaşık sayısının 32-bit kayan noktalı sayı formatının hexadecimal olarak gösterimleridir.

Şekil 5.9’da gösterilen “FFT Megacore” ara yüzünde 64 uzunluklu IFFT hesaplanması için gerekli sürenin yaklaşık 64 saat döngüsü olarak verildiği görülmektedir. Fakat Şekil 5.11’de verilen IFFT iç mimarisinde giriş/çıkış verilerinin öncelikli olarak RAM bloklarına yazıldığı görülmektedir. Bu da verilerin IFFT bloğu çıkışına aktarılma zamanını arttıracaktır. Modelsim benzetim sonuçlarında giriş portundan ilk verinin girildikten sonra çıkışa aktarılması arasında geçen süresinin –zaman gecikmesi- 122 saat döngüsü olduğu gözlenmiştir.

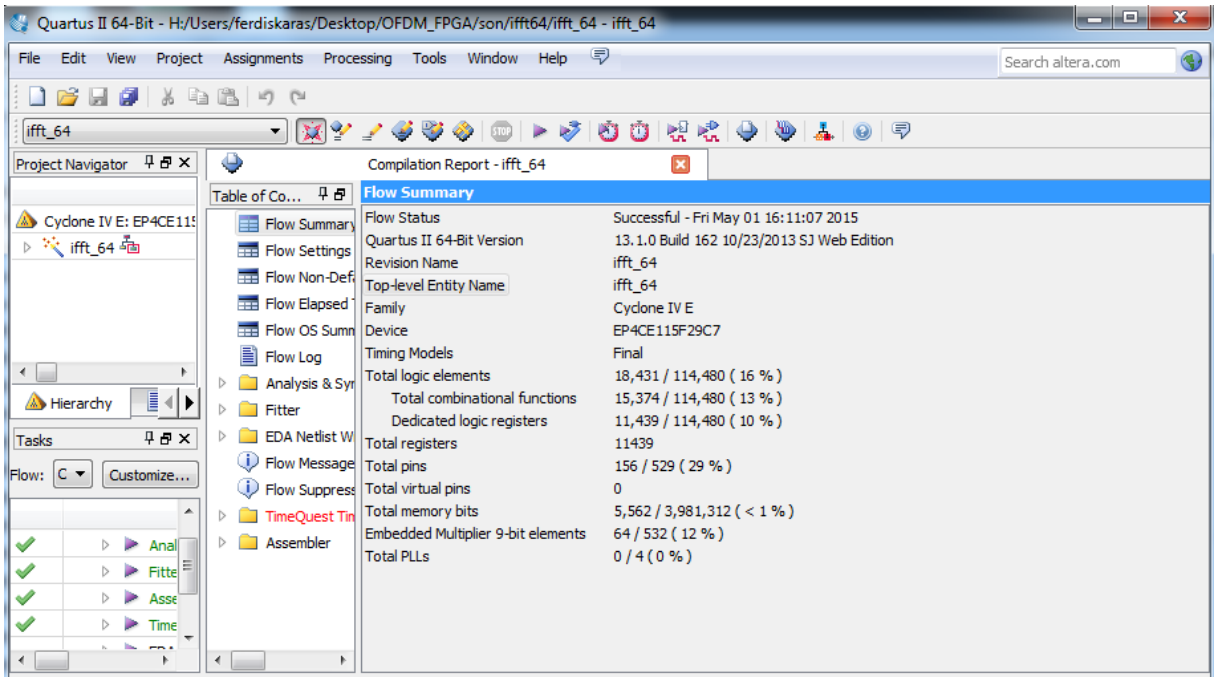


Şekil 5.13 IFFT Modelsim benzetiminin giriş çerçevesi değişimi anındaki sinyal tetiklemelerini ve sonuçlarını gösteren ekran görüntüsü.



Şekil 5.14 IFFT Modelsim benzetiminin çıkış çerçevesi değişimi anındaki sinyal tetiklemelerini ve sonuçlarını gösteren ekran görüntüsü.

Modelsim benzetimlerinin MATLAB'da hesaplanan TAFD ile uyumluluk göstermesi üzerine "FFT Megacore" ile oluşturulan VHDL kodunun sentezlenmesi "Quartus II 13.1" programı ile gerçekleştirilmiştir. Sentezlemenin başarılı olduğuna dair ekran görüntüsü Şekil 5.15'te verilmiştir.



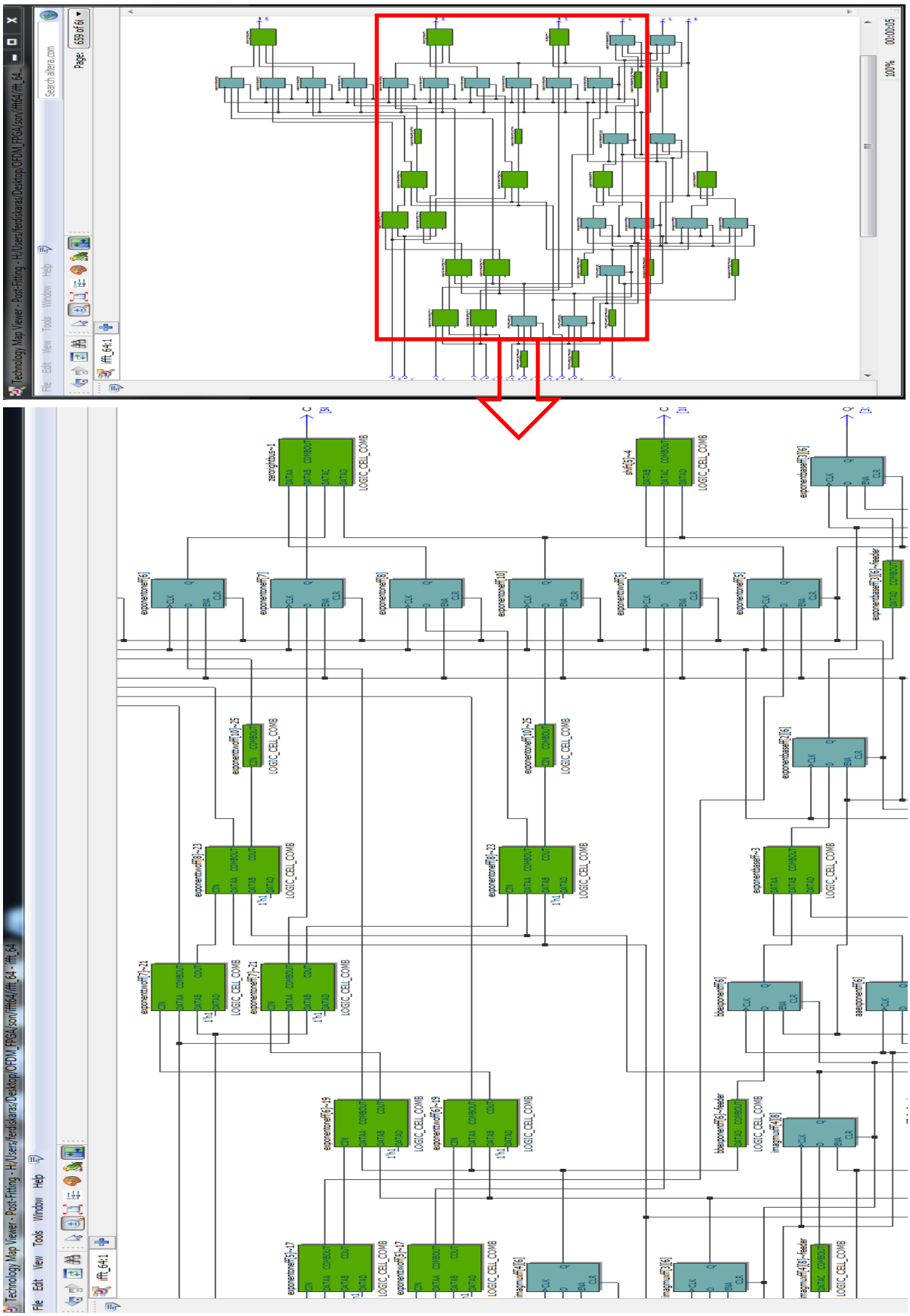
Şekil 5.15 IFFT fonksiyonun Quartus II ile sentezleme sonucu.

Sentezleme sonucu oluşturulan donanımın kullanılan platform olan DE2-115 Geliştirme kartı üzerindeki kaynak kullanımı Çizelge 5.2'de verilmiştir.

Çizelge 5.2 Sentezlenen IFFT bloğunun kaynak kullanımı ve oranı.

Kaynak Çeşidi	**Kullanılan Kaynak (IFFT)**	**Mevcut Donanım (DE2-115)**	**Kullanım Oranı**
Mantık Elemanı (LEs)	18431	114480	16 %
Hafıza(bellek) biti	5562	3981312	<1 %
DSP-9bit Bloğu	64	532	12 %

Sentezleme sonucu oluşturulan devre şeması Şekil 5.16'da verilmiştir.



Şekil 5.16 IFFT bloğunun sentezlenmesi sonucu oluşan devre şemasına ait ekran görüntüsü.

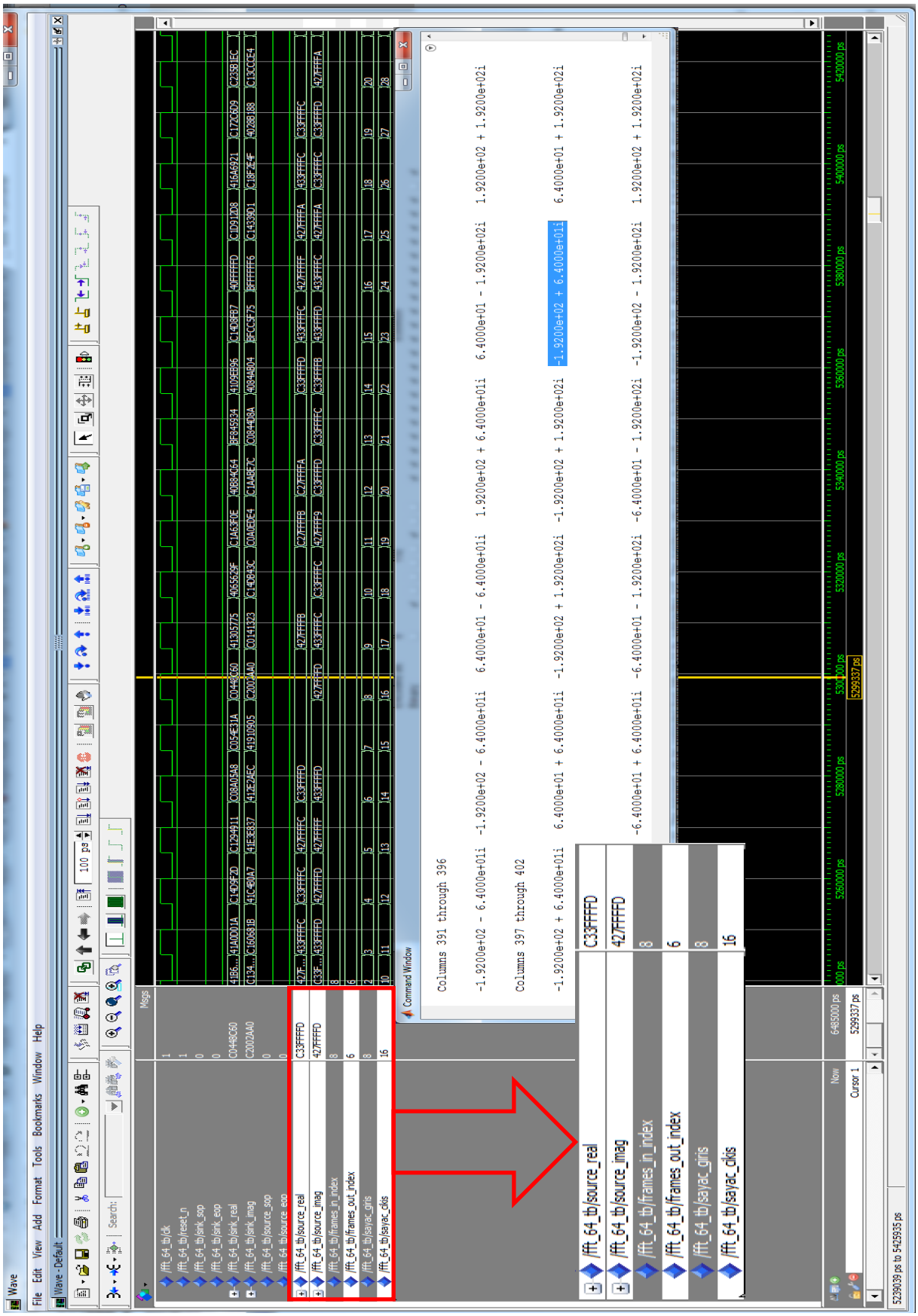
OFDM mimarisinin verici kısmında modülasyon ve TAFD işlemlerine tabi tutulan sembollerin alıcı kısmında aynı şekilde geri alınabilmesi için öncelikle AFD işleminin uygulanması gereklidir. AFD işlemi de tıpkı vericide uygulanan TAFD işleminde olduğu gibi "MegaWizard-Plug-In" aracının "FFT Megacore" kütüphanesi kullanılarak oluşturulmuştur.
Bu nedenle AFD için uygulanan basamaklar detaylı olarak anlatılmayacaktır. FFT işlemini oluşturulabilmek için Şekil 5.9'da verilen ara yüzlerden gerekli ayarlamalar yapılmıştır. IFFT bloğunun doğru bir şekilde çalıştırılması için gerekli olan "Avalon-ST" ara yüzündeki sinyal tetiklemelerinin uygun şekilde gerçekleştirilme işlemi FFT bloğu içinde geçerlidir. FFT bloğunun doğru bir şekilde çalışması için gerekli olan bütün adımlar IFFT bloğunun sentezlenmesinde anlatıldığı gibi uygulanmış ve FFT bloğu için de Modelsim benzetimi oluşturulmuştur. Bu benzetime ait sonuçların doğruluğunu test etmek amacıyla yine MATLAB ortamı kullanılmıştır. FFT bloğunun benzetimi için kullanılan sinyal girişleri MATLAB için de "fft" işleminin girişleri olarak belirlenerek yapılan hesaplamalara ait sonuçlar Şekil 5.17'de verilmiştir. Çerçeve indeksin 6,çıkış sayacının 16 değerinde olması o çıkışın 6×64+(16+1) = 401. çıkış sinyali olduğunu göstermektedir.

Modelsim benzetiminde FFT işleminin gerçekleştirilip blok çıkışına aktarılması için geçen sürenin – zaman gecikmesi – 119 saat döngüsü olduğu gözlenmiştir.

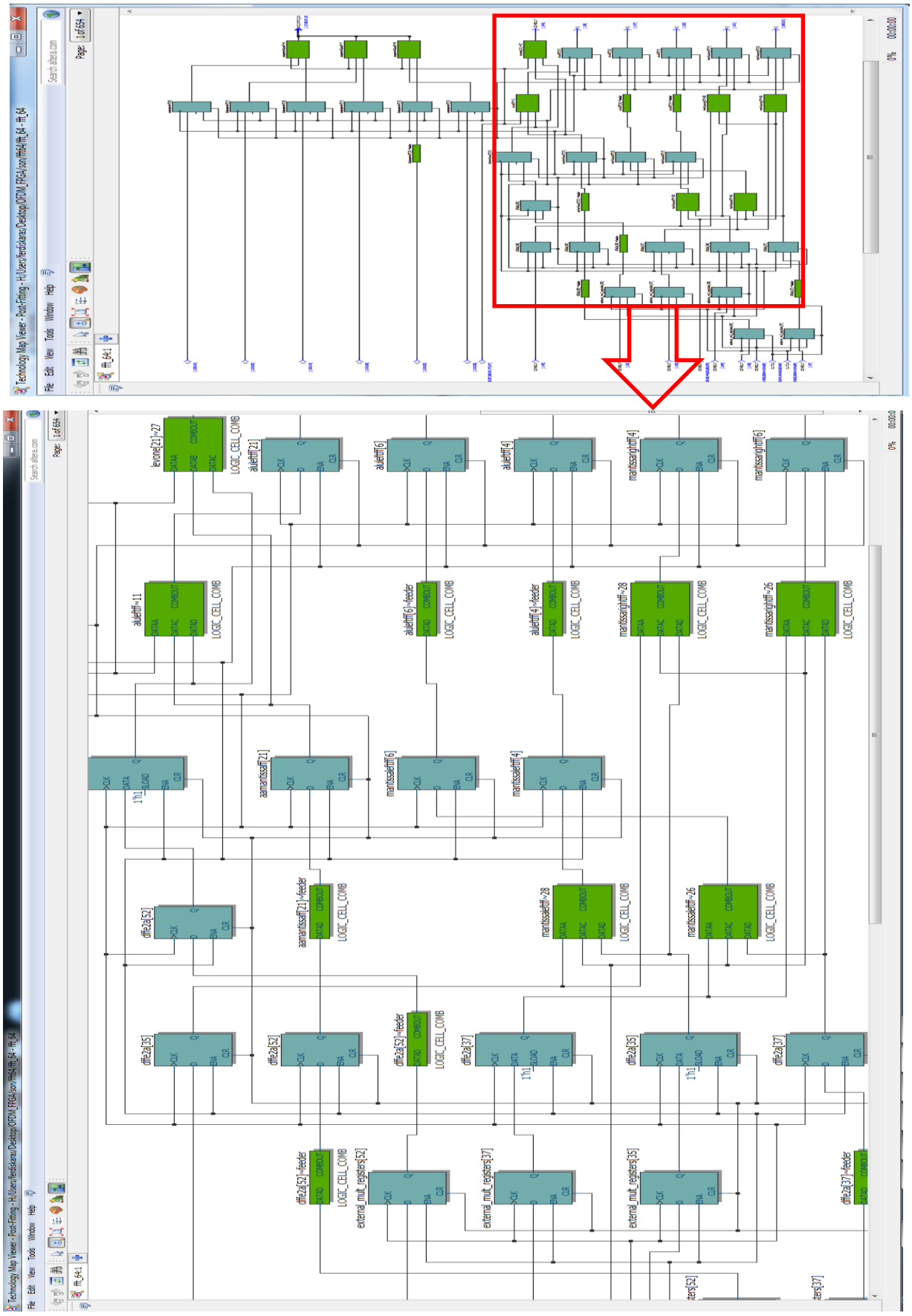
Modelsim benzetimlerinin doğruluğu test edildikten sonra FFT bloğunun sentezleme işlemi gerçekleştirilmiş ve sentezlenen devreye ait kaynak kullanımı Çizelge 5.3'te verilmiştir. Sentezleme sonucu oluşturulan devre şeması ise Şekil 5.18'de verilmiştir.

Çizelge 5.3 Sentezlenen FFT işleminin kaynak kullanımı ve oranı.

Kaynak Çeşidi	**Kullanılan Kaynak (FFT)**	**Mevcut Donanım (DE2-115)**	**Kullanım Oranı**
Mantık Elemanı (LEs)	18117	114480	16 %
Hafıza(bellek) biti	8732	3981312	<1 %
DSP-9bit Bloğu	64	532	12 %



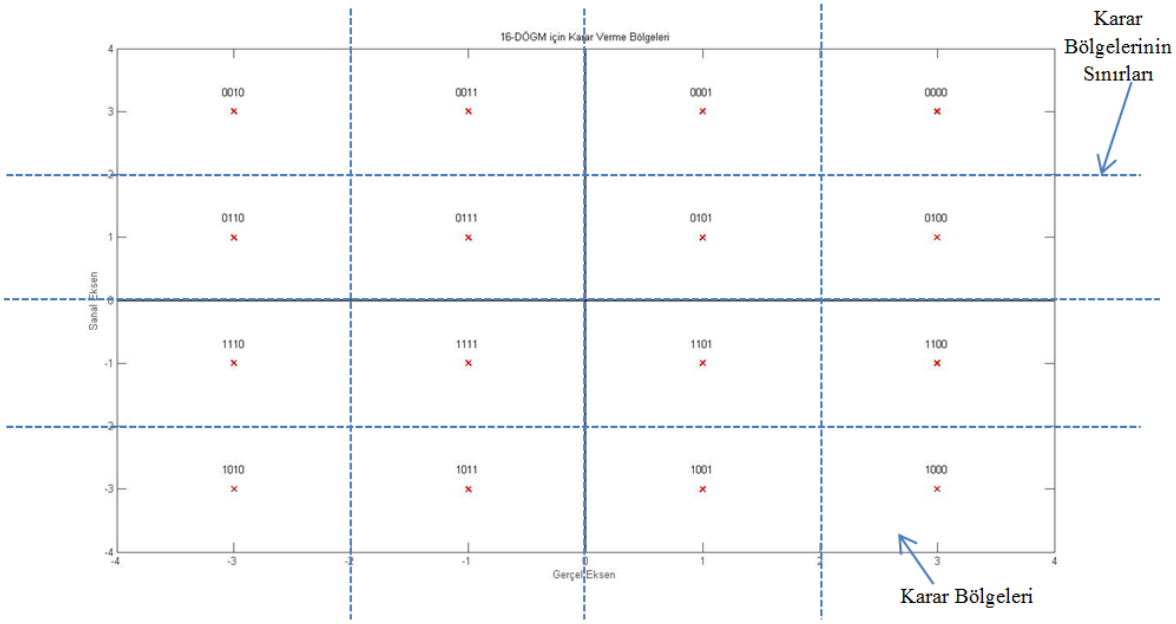
Şekil 5.17 FFT bloğunun Modelsim benzetiminin sonuçlarını gösteren ekran görüntüsü.



Şekil 5.18 FFT bloğunun sentezlenmesi sonucu oluşan devre şemasına ait ekran görüntüsü.

5.4 DÖRDÜN GENLİK DEMODÜLASYONU GERÇEKLENMESİ

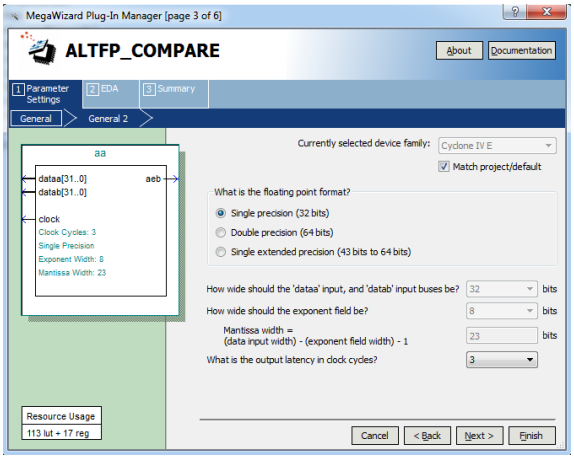
Modülasyon işleminde 4-bit'lik kelimeler reel ve sanal eksene haritalanmıştı. Demodülasyon bloğunda yapılması gereken işlem ise, FFT işlemi sonrası demodülasyon bloğu girişine gelen gerçel ve sanal sinyallerin blok çıkışına 4-bit'lik kelimelerle eşleştirilerek aktarılmasıdır. Bu eşleştirme işlemi gerçekleştirilirken göz önünce bulundurulması gereken, demodülasyon işlemine tabi tutulacak gerçel ve sanal girişlerin tam olarak sabit değerlerde olmaması durumudur. Gerek uygulanan işlemelerden kaynaklanan (IFFT ve FFT gibi) gerekse kanalda oluşabilecek gürültülerden dolayı demodülasyon bloğunun girişleri, modülasyon sonucu haritalanan gerçel ve sanal değerlerden farklılıklar gösterecektir. Bu durumda eşleşme işleminin yapılabilmesi için karar bölgeleri belirlenmelidir. Bu karar bölgeleri içinde kalan sinyaller o karar bölgesindeki kelimeye eşlenmelidir. Karar bölgelerini ve o bölgelerdeki kelimeleri gösteren işaret-yıldız kümesi gösterimi Şekil 5.19'da verilmiştir. Burada gerçel ve sanal değerler için ayrı ayrı belirlenen karar bölgeleri demodülasyon girişine gelen sinyallerle karşılaştırılarak, çıkışa karar verilecektir.



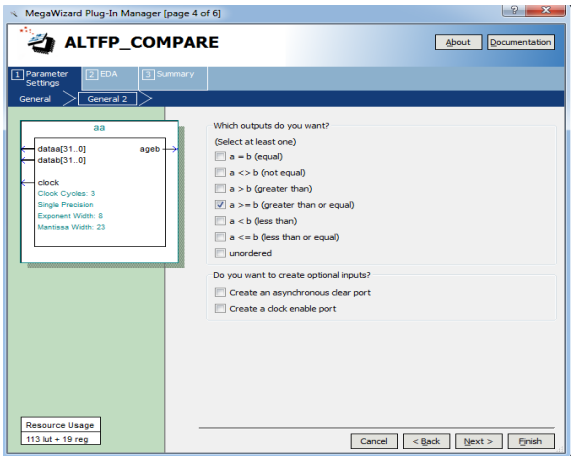
Şekil 5.19 Demodülasyon için karar verme sınırları ve bölgeleri.

FFT bloğu çıkışında elde edilen 32-bit kayan noktalı sayı formatındaki gerçel ve sanal ekseni temsil eden sinyallerin, karar bölgelerinin sınırları ile karşılaştırma işlemleri için "MegaWizard Plug-In" aracının "Floating-Point" kütüphanesi kullanılmıştır. Şekil 5.6'da gösterilen "MegaWizard Plug-In" ara yüzünün "Arithmetic" başlığının altında bulunan bu

kütüphaneden “ALTFP_Compare” fonksiyonu seçilmiştir. Bu seçim işlemi sonucu açılan kullanıcı ara yüzleri Şekil 5.20’de gösterilmiştir.



a)Kullanılacak FPGA ailesinin ve sayı formatının seçildiği ara yüz

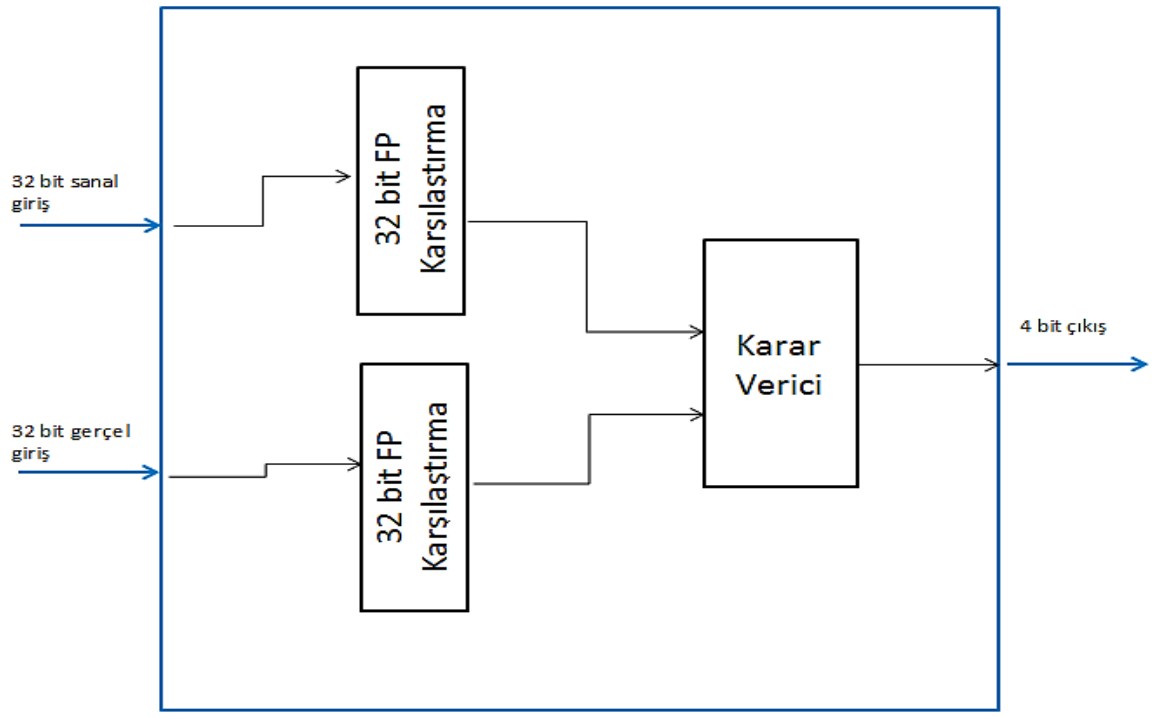


b)Fonksiyon çıkışına oluşturulacak sinyallerin belirlendiği ara yüz

Şekil 5.20 ALFP_COMPARE fonksiyonunun kullanıcı ara yüzlerine ait ekran görüntüsü.

Şekil 5.20.a’da gösterilen ara yüzden kullanılacak olan FPGA ailesi ve sayı formatı seçilmiştir. Şekil 5.20.b’de ise karşılaştırma sonucu fonksiyon çıkışında oluşturulacak sinyaller belirlenmiştir. Fonksiyonun girişi olarak gösterilen “dataa” sinyali demodülasyon girişine gelen 32-bit kayan noktalı sayı formatındaki gerçel veya sanal eksen sinyalleri olarak belirlenmiş, “datab” sinyali ise gerçel ve sanal bölge sınırlarının 32-bit kayan noktalı sayı formatı karşılıkları olarak ayarlanmıştır. Fonksiyon çıkışında elde edilen “ageb” sinyali “dataa” nın “datab” den sayısal olarak büyük veya eşit olması durumunda “1”, aksi durumda “0” değerini almaktadır. Bu “ageb” sinyalinin “1” ya da “0” olması durumuna göre demodülasyon girişine gelen sinyallerin hangi bölgede olduğuna karar verilmiş ve çıkış o bölgedeki kelimeye eşlenmiştir.

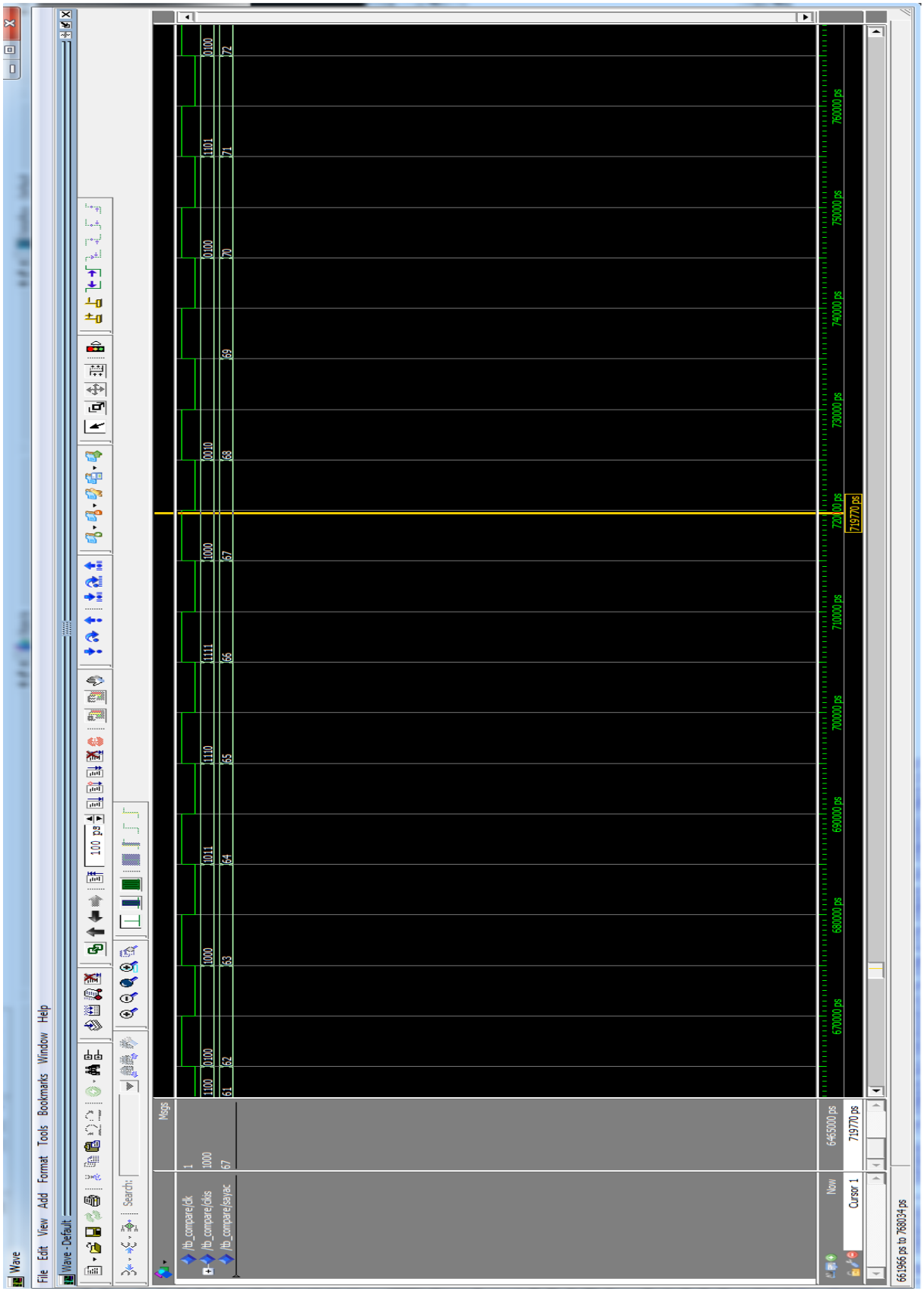
Demodülasyon bloğu için yapılan VHDL tasarımının blok gösterimi Şekil 5.21'de verilmiştir. Burada "FP Karşılaştırma" olarak verilen bloklar "MegaWizard Plug-In" aracı ile oluşturulan fonksiyonlar, "Karar Verici" bloğu ise karşılaştırma sonuçlarına göre 4-bit'lik kelimelere eşleştirme yapan VHDL kod bloğudur.



Şekil 5.21 Demodülasyon VHDL tasarımının blok şeması.

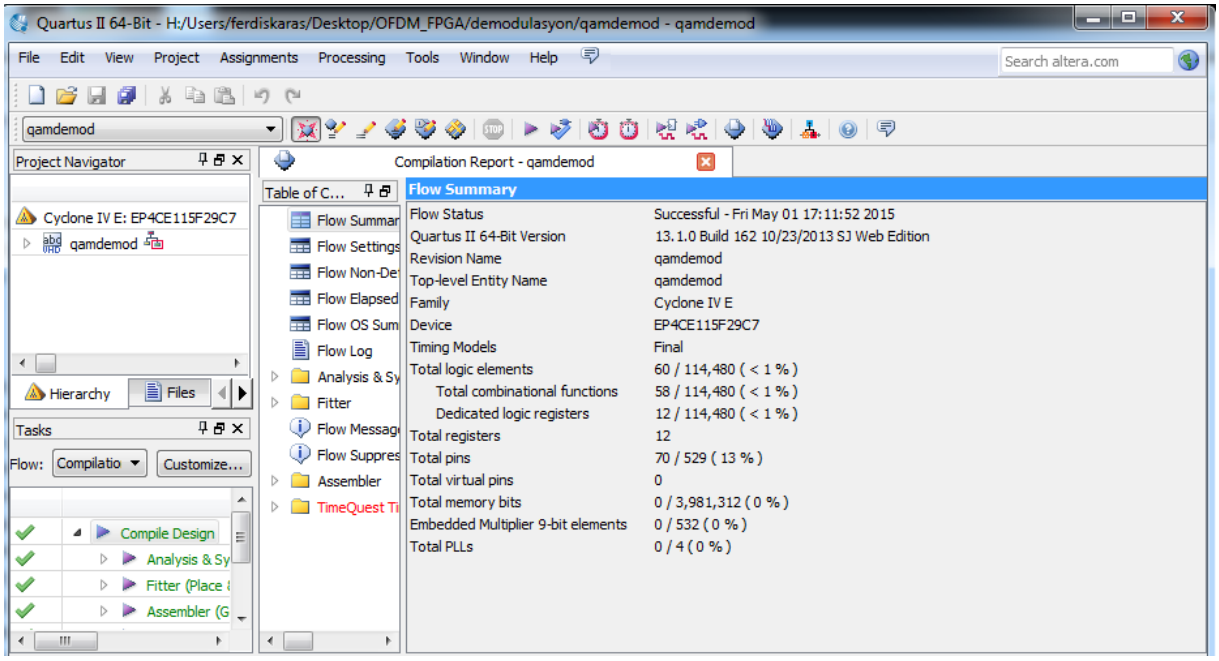
Demodülasyon için oluşturulan bloğun Modelsim benzetimi yapılarak fonksiyonel doğruluğu test edilmiştir. Bu benzetim Şekil 5.22'de gösterilmiştir. Modelsim benzetiminde çıkışa aktarılan verinin kaçıncı kelime olduğu "sayac" verisinde tutulmaktadır. Burada "67" numaralı kelimenin 4-bit'lik "1000" olduğu gözükmektedir. Modülasyon kısmının benzetiminin gösterildiği Şekil 5.4'te "67" numaralı kelimenin "1000" olduğu gözükmektedir. Bu da bize hem demodülasyon işleminin hem de bu basamağa kadar gerçekleştirilen bütün blokların doğru bir şekilde çalıştığını göstermektedir.

Modelsim benzetiminde demodülasyon bloğunda verilerin blok çıkışına aktarılmasının 4 saat döngüsü süresinde gerçekleştiği gözlenmiştir.



Şekil 5.22 Demodülasyon bloğunun Modelsim benzetimine ait ekran görüntüsü.

Modelsim benzetiminin test edilmesinden sonra demodülasyon bloğu sentezlenmiştir. Sentezleme sonucunu Şekil 5.23'te gösterilmiştir.



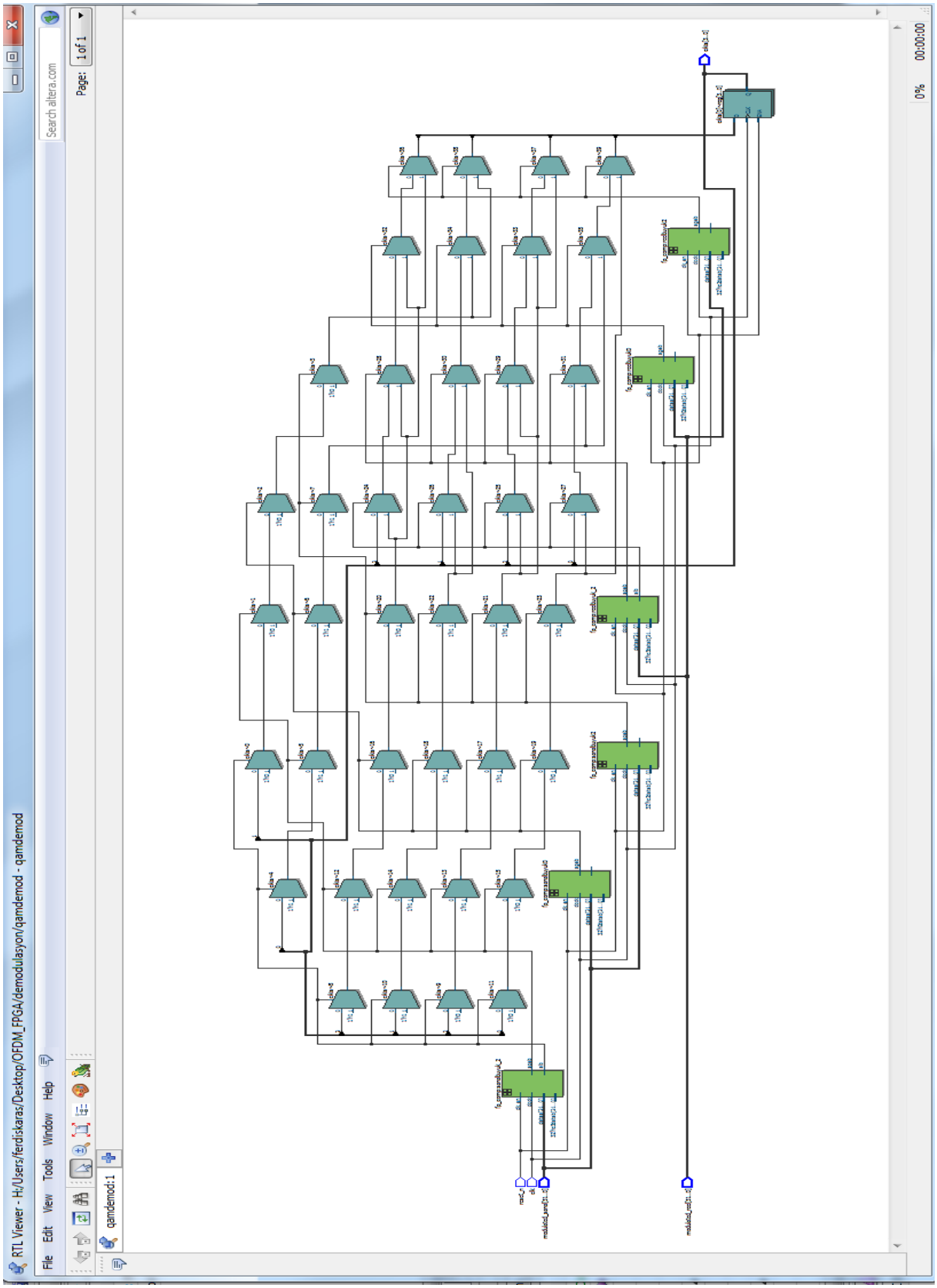
Şekil 5.23 Demodülasyon bloğunun Quartus II ile sentezleme sonucuna ait ekran görüntüsü.

Sentezleme sonucunda demodülasyon bloğunun kaynak kullanımı ve kullanılan platformun kaynaklarına oranı Çizelge 5.4'te verilmiştir.

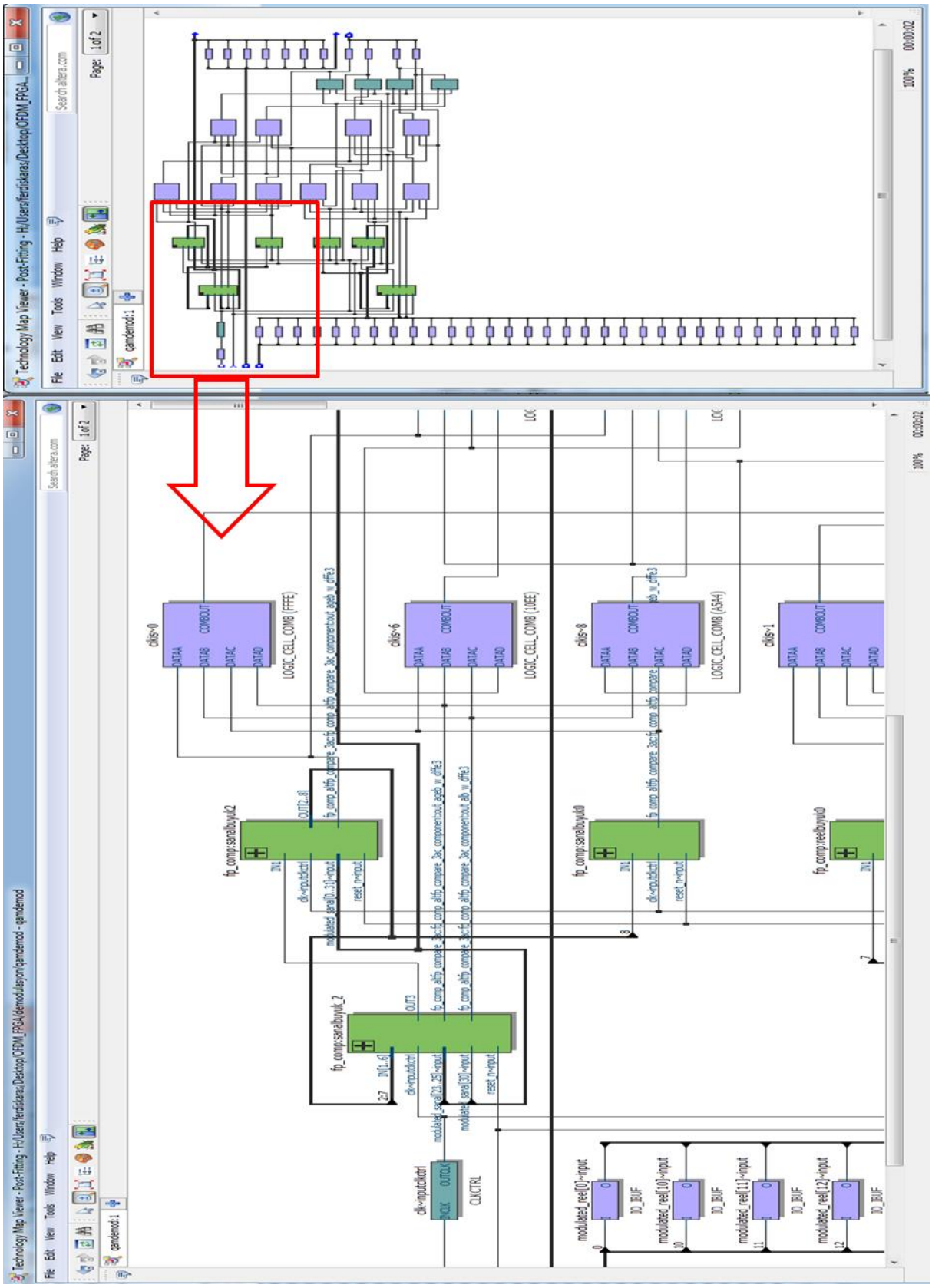
Çizelge 5.4 Sentezlene Demodülasyon bloğunun kaynak kullanımı ve oranı.

Kaynak Çeşidi	**Kullanılan Kaynak (FFT)**	**Mevcut Donanım (DE2-115)**	**Kullanım Oranı**
Mantık Elemanı (LEs)	60	114480	<1 %
Giriş/Çıkış Pini	70	529	13 %

Sentezleme sonucu oluşan saklayıcı transfer seviyesi (RTL) devre şeması Şekil 5.24'te verilmiştir. Optimizasyon sonrası oluşan kapı seviyesi (gate level) devre şeması ise Şekil 5.25'te verilmiştir.



Şekil 5.24 Demodülasyon RTL devre şemasına ait ekran görüntüsü.



Şekil 5.25 Demodülasyon bloğunun sentezlenmesi sonucu oluşan devre şemasına ait ekran görüntüsü.

BÖLÜM 6

SONUÇLAR

Bu tez çalışmasında temel bant OFDM tekniğinin verici ve alıcı kısmında bulunan blokların VHDL kodları yazılarak modülasyon ve demodülasyon bloklarının FPGA üzerinde gerçeklenmesi yapılmıştır. Tasarlanan blokların bit seviyesinde fonksiyonel olarak testlerinin yapılabilmesi için "Modelsim Altera 10.1" kullanılmıştır. Modelsim benzetimleri MATLAB da oluşturulan modellerle karşılaştırılmıştır. Test sonuçlarından emin olunan blokların sentezlenmesi "Quartus II 13.1" kullanılarak yapılmış ve oluşturulan "bitstream" dosyaları Altera firmasına ait DE2-115 Geliştirme ve Eğitim kartı üzerinde bulunan FPGA'ya yüklemiştir.

OFDM mimarisinin FPGA üzerinde gerçeklenmesi için sayı formatı olarak IEEE 32-bit kayan noktalı sayı formatı tercih edilmiştir. IFFT/FFT bloklarının tasarımı için Altera firmasının "MegaWizard Plug-In" aracının sunduğu kütüphaneler kullanılmıştır

Modülasyon çeşidi olarak 16-DÖGM ve OFDM alt taşıyıcı uzunluğu olarak 64 seçilmiştir. Bu parametrelerle sentezlenen sistemin verici ve alıcı kısmındaki toplam kaynak kullanımları ve Cyclone IV 4CE11529C7 FPGA üzerindeki donanım kaynaklarına oranları hesaplanmıştır. Bu değerler Çizelge 6.1'de verilmiştir. Bu çizelgeden Cyclone IV 4CE11529C7 FPGA üzerinde alıcı/verici sistemin gerçekleşebileceği sonucuna varılmıştır.

Verici ve alıcı iki ana blok halinde gerçekleştirilen fonksiyonel testler sonucunda verici giriş veri seti ile alıcı çıkış veri setlerinin birebir örtüştüğü gözlenmiştir.

Çizelge 6.1 OFDM mimarisinin verici alıcı kısımlarında kullanılan toplam kaynak kullanımı ve mevcut kaynaklara oranı.

Kaynak Çeşidi	**Kullanılan Kaynak Verici/Alıcı**	**Mevcut Donanım (DE2-115)**	**Kullanım Oranı Verici/Alıcı**
Mantık Elemanı (LEs)	18451 / 18177	114480	16 % / 16 %
Hafıza(bellek) biti	5562 / 8732	3981312	<1 %
DSP-9bit Bloğu	64 / 64	532	12% / 12 %

Sistemin FPGA kullanılarak tasarımı modülasyon seviyesi ve FFT uzunluğu gibi parametrelerin değiştirilebilmesi esnekliğini sunmaktadır. Fakat özellikle FFT uzunluğunun arttırılması gecikmeyi de arttıracaktır. Verici kısmında modülasyon ve 64-IFFT işlemleri toplam 123 saat döngüsü sürede, alıcı kısmında ise demodülasyon ve 64-FFT işlemleri 122 saat döngüsü sürede gerçekleştirilmektedir. Bu nedenle FFT bloğunun daha az gecikme ile gerçekleştirilmesi işlemi önem arz etmektedir. Bunun için iki yöntem önerilmektedir. Birinci yöntem; sayı formatının değiştirilerek gerekli hassasiyeti sağlayabilecek sabit noktalı bir sayı formatı belirlenmesidir. Böylece IFT/FFT hesaplamasında yapılması gereken işlem yükü azalacaktır. Bu da hem kaynak kullanımı hem de gecikme açısından yararlı olacaktır. İkinci yöntem ise; IFFT/FFT işleminin firmaların sunduğu kütüphaneler kullanılmaksızın, istenilen hassasiyette uygulamaya özgü VHDL kodunun yazılmasıdır.

İleriki çalışmalarda gerçeklenen temel bant OFDM mimarisinin akıllı antenler kullanılarak RF bloklarının da sisteme dâhil edilerek kablosuz ortamda gerçeklenmesi hedeflenmektedir. Ayrıca üretici firma bağımlılığını ortadan kaldırmak adına OFDM mimarisinde kullanılacak IFFT/FFT modüllerinin VHDL kodunun yazılması ve sisteme Hızlı Frekans Hoplamalı-Dik Frekans Bölmeli Çoğullama (Fast Frekans Hoping- Orthogonal Frequency Division Multiplexing- FFH-OFDM) algoritmasının da dâhil edilerek gerçeklenmesi hedeflenmektedir.

KAYNAKLAR

Aktürk A T (2014) OFDM Tabanlı Temel Bant WIMAX Fiziksel Katman Vericinin FPGA Üzerinde Gerçekleşmesi. *Yüksek Lisans Tezi*, İstanbul Teknik Üniversitesi Fen Bilimleri Enstitüsü Elektronik Haberleşme Mühendisliği Anabilim Dalı, İstanbul, 69 s.

Altera Corporation (2008) Implementing OFDM Modulation for Wireless Communications, Application Notes, 503 p.

Altera Corporation (2014) *Cyclone IV Device*, Handbook, 490 p.

Cooley J W and Tukey J W (1965) An Algorithm for the Machine Calculation of Complex Fourier Series, *Mathematics of Computation*, 19 (90): 297-301

Chen H and Guizani M (2006) *Next Generation Wireless Systems and Networks*, ISBN: 978-0-470-02434-8, John Wiley & Sons Ltd, 516 p.

Çavuşlu M A ve Kösten M M (2015) *VHDL ile Sayısal Tasarım ve FPGA Uygulamaları*, ISBN: 978-605-9118-46-0, KODLAB Yayıncılık, İstanbul 318 s.

Dick C and Harris F (2003) FPGA Implementation of an OFDM PHY, *Conference Record of the Thirty-Seventh Asilomar Conference*,1: 905-909

Dejan M D, Dejan R, Slobodan D and Veljko V (2004) FPGA-based Prototyping of IEEE 802.11a Baseband Processor, *Serbian Journal of Electrical Engineering*, 1(3): 125-136

Emir A (2014) Sayısal İşaret İşleyicilerde OFDM Tasarımı. *Yüksek Lisans Tezi*, Fen Bilimleri Enstitüsü, Elektrik-Elektronik Mühendisliği Anabilim Dalı, Zonguldak, 85 s.

Ertürk S (2009) *Sayısal İşaret İşleme*, ISBN: 978-975-511-309-8 Birsen Yayınevi, İstanbul, 350 s.

Garcia J and Cumplido R (2005) On the design of an FPGA-Based OFDM modulator for IEEE 802.16-2004, *International Conference on Reconfigurable Computing and FPGAs (ReConFig 2005)*

Goldsmith A (2005) *Wireless Communications*, ISBN: 978-0-521-83716-3, Cambridge University Press, London, 561 p.

Gutierrez F M (2009) Implementation of a Tx/Rx OFDM System in a FPGA. *Master Thesis*, Polytechnic University of Catalonia, Barcelona, 61 p.

KAYNAKLAR (devam ediyor)

Haykin S and Moher M (2005) *Modern Wireless Communications.* International Edition, ISBN: 978-0-130-22472-9, Pearson Prentice Hall, New Jersey 560 p.

Hirosaki B (1981) An Orthogonally Multiplexed QAM System Using the Discrete Fourier Transform. *IEEE Trans. Commuications*, 29: 982-989.

Kaptan H, Tangel A and Şahin S (2007) FPGA Implementation of FFT Algorithms Using Floating Point Numbers, *5th Int. Conf. Electrical and Electronics Engineers (ELECO-2007)*

Li Y and Stüber G L (2006) *Orthogonal Frequency Division Multiplexing for Wireless Communications,* ISBN: 978-0-387-30235-5, Springer, e-book, 306 p.

Mohamad M A (2013) FPGA Synthesis of VHDL OFDM System, *Wireless Personal Communications*, 4 (70): 1885-1909

Peled A and Ruiz A (1980) Frequency Domain Data Transmission using Reduced Computational Complexity algorithms. *Acustic, Speech and Signal Processing Conference, IEE ICASSP*, 5: 964-967.

Prasad R (2004) *OFDM for Wireless Communication Systems,* ISBN: 978-1-580-53796-4 Artech House, London, 294 p.

Proakis J G and Salehi M (2005) *Fundamentals of Communication Systems.* ISBN: 978-0-131-47135-1, Pearson Prentice Hall, New York, 858 p.

Proakis J G and Salehi M (2008) *Digital Communications.* 5th edition, ISBN: 978-0-072-95716-7, Mc Graw Hill, New York, 923 p.

Proakis J G and Monalakis D G (2007) *Digital Signal Processing Principles Algorithms and Application.* 4th edition, ISBN: 978-0-131-87374-2, Prentice Hall, New Jersey 1050 p.

Rappaport T (2002) *Wireless Communications: Principles and Practice,* 2nd edition, ISBN: 978-0-130-42232-3, Prentice Hall, New Jersey, 736 p.

Sarıtaş E ve Karataş S (2013) *Her Yönüyle FPGA ve VHDL*, ISBN: 978-605-355-129-4, Palme Yayıncılık, Ankara 411 s.

Sklar B (2001) *Digital Communications Fundamentals and Applications.* 2nd edition, ISBN: 978-0-130-84788-1, Prentice Hall, New Jersey, 1079 p.

Stüber G L (2000) *Principles of Mobile Communication.* 2nd edition, ISBN: 978-0-792-37998-0, Kluwer Academic Publishers, Massachusetts 776 p.

Şahin S (2010) FPGA Tabanlı OFDM Modem Gerçeklemesi. *Doktora Tezi*, Fen Bilimleri Enstitüsü, Elektronik-Haberleşme Mühendisliği Anabilim Dalı, Kocaeli, 138 s.

KAYNAKLAR (devam ediyor)

Şahin S, Dikmeşe Ş and Kavak A (2008) Which Number Format to Use for Baseband Wimax Modem Implementation on an FPGA ?, *Communication and Applications Conference, SIU 2008. IEEE 16th*, 1-4.

Terasic Technologies (2013) DE2-115 User Manual, 121 p.

Tunçkaya Y (2009) MQAM ve MPSK Modülasyonlu OFDM Sistemlerin Gürültülü ve Rayleigh Sönümlemeli Kanallarda Başarımlarının Benzetimle İncelenmesi. *Yüksek Lisans Tezi*, Fen Bilimleri Enstitüsü, Elektrik-Elektronik Mühendisliği Anabilim Dalı, Zonguldak, 111 s.

URL-1 <*http://chipdesignmag.com/display.php?articleId=386*>, Ziyaret Tarihi: 10.2.2013

URL-2 <*http://article.sapub.org/10.5923.j.msse.20120102.05.html*>, Ziyaret Tarihi: 10.2.2013

URL-3 <*http://www.seas.upenn.edu/~ese171/vhdl/vhdl_primer.html*>, Ziyaret Tarihi: 25.12.2013

URL-4 <*http://esd.cs.ucr.edu/labs/tutorial/* >, Ziyaret Tarihi: 15.1.2014

URL-5 <*http://www.ics.uci.edu/~alexv/154/VHDL-Cookbook.pdf* >, Ziyaret Tarihi: 12.2.2014

Volnei A P (2005) *Circuit Design With VHDL*, ISBN: 978-0-262-01433-5, MIT Press, Massachusetts, 363 p.

Weinstein S B and Ebert P M (1971) Data Transmission by Frequency Division Muktiplexing Using the Discrete Fourier Transform. *IEEE Trans. Communications*, 19: 628-634.

Zhang Y and Chen H (2008) *Mobile WiMAX Toward Broadband Wireless Metropoltian Area Networks.*ISBN: 978-0-849-32624-0, Auerbach Publications, New York, 472 p.

EK AÇIKLAMALAR

EK A: OFDM Verici Kısmı İçin Modelsim Benzetim Test Dosyası Kodları

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use std.textio.all;
use ieee.std_logic_textio.all;
entity verici_tb is
end verici_tb;

architecture tb of verici_tb is

  constant saat_araligi : time := 1 ns;

  signal clk     : std_logic;
  signal reset_n : std_logic;
  signal fftpts_in   : std_logic_vector (6 downto 0):="1000000";
  signal fftpts_out     : std_logic_vector (6 downto 0);
  signal sink_valid   : std_logic;
  signal sink_sop     : std_logic;
  signal sink_eop     : std_logic;
  signal sink_error   : std_logic_vector(1 downto 0);
  signal sink_real    : std_logic_vector (32 - 1 downto 0);
  signal sink_imag    : std_logic_vector (32 - 1 downto 0);
  signal source_ready : std_logic;
  signal sink_ready   : std_logic;
  signal source_sop   : std_logic;
  signal source_eop   : std_logic;
  signal source_valid : std_logic;
  signal source_error : std_logic_vector(1 downto 0);
  signal source_real : std_logic_vector (32 - 1 downto 0);
  signal source_imag : std_logic_vector (32 - 1 downto 0);

----------OFDM alt taşıyıcı sayısını belirleme----------------------------------------------------------
-------------------------------------------------------------------------------------------------------------
constant alt_tasiyici : natural := 10;
-------------------------------------------------------------------------------------------------------------

-----Tüm alt taşıyıcıların uzunluğunun 64 olarak ayarlanması-----------------------------------------
-------------------------------------------------------------------------------------------------------------
  constant fft_uzunlugu : integer :=64;

-------------------------------------------------------------------------------------------------------------
  signal start           : std_logic;

  -------- giriş ve çıkış frame lerinin indexleri----------------------------------------------------------
-------------------------------------------------------------------------------------------------------------
```

```
signal frames_in_index   : natural range 0 to alt_tasiyici := 0;
signal frames_out_index : natural range 0 to alt_tasiyici := 0;

-----------------------------------------------------------------------------------------------------------------

  signal sayac_giris        : natural range 0 to 64:=0;
  signal sayac_cikis        : natural range 0 to 64:=0;



  -----------------Tüm veriler ifft işlemine tabi tutulduktan sonra tetiklenecek olan ve
simülasyonu bitirecek sinyal----------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------

  signal end_output : std_logic;

-----------------------------------------------------------------------------------------------------------------

signal data_giris : std_logic_vector (3 downto 0);
signal reset_negatif : std_logic;


---------------------------------------------------------------------------- ---------------------------
Component Tanımlamaları
------------------------------------------------------------- ----------------------------------------------
component dogm
       port (data_giris   : in std_logic_vector (3 downto 0);
             cikis_sanal  : out std_logic_vector (31 downto 0);
             cikis_gercel : out std_logic_vector (31 downto 0);
    cikis_var : out std_logic;
             clk : in std_logic;
             reset_negatif : in std_logic);
    end component;

  component ifft_64 is
    port (
      clk         : in  std_logic;
      reset_n     : in  std_logic;
      fftpts_in   : in  std_logic_vector (6 downto 0);
      fftpts_out  : out  std_logic_vector (6 downto 0);
      sink_valid  : in  std_logic;
      sink_sop    : in  std_logic;
      sink_eop    : in  std_logic;
      sink_real   : in  std_logic_vector (32 - 1 downto 0);
      sink_imag   : in  std_logic_vector (32 - 1 downto 0);
      source_ready : in  std_logic;
      sink_ready  : out std_logic;
      sink_error  : in  std_logic_vector(1 downto 0);
      source_error : out std_logic_vector(1 downto 0);
      source_sop  : out std_logic;
```

```
    source_eop   : out std_logic;
    source_valid : out std_logic;
    source_real  : out std_logic_vector (32 - 1 downto 0);
    source_imag  : out std_logic_vector (32 - 1 downto 0)
    );
  end component ifft_64;

begin
  ----------------------------------Reset Sinyalini Üret-----------------------------------------------
-------------------------------------------------------------------------------------------------------------
  reset_negatif <= '0',
           '1' after 92*saat_araligi;
-----------------------------------------------Saat Sinyalini üret-------------------------------------
-------------------------------------------------------------------------------------------------------------
  clkgen : process
  begin
    if  end_output = '1' then
      clk <= '0';
      wait;
    else
      clk <= '0';
      wait for 5*saat_araligi;
      clk <= '1';
      wait for 5*saat_araligi;
    end if;
  end process clkgen;
-------------------------------------------------------------------------------------------------------------

  ----------- source_ready her zaman tetiklenmiş-----------------------------------------------------

  source_ready <= '1';
-------------------------------------------------------------------------------------------------------------

 ------------------------------------------- giriş hatası yok-----------------------------------------------
  sink_error <= (others => '0');
-------------------------------------------------------------------------------------------------------------
  ---------- modülasyondan ilk veri gelene kadar ifft start sinyali tetiklenmiş durumda----------
-------------------------------------------------------------------------------------------------------------
  start_p : process (clk, reset_n)
  begin
    if reset_n = '0' then
      start <= '1';
    elsif rising_edge(clk) then
      if sink_valid = '1' and sink_ready = '1' then
        start <= '0';
      end if;
    end if;
  end process start_p;

-------------------------------------------------------------------------------------------------------------
```

```
----------paket başlangıcı ve sonucunu belirlemek için ifft girişinde sayaç belirlenmesi---------
-------------------------------------------------------------------------------------------------------------
  sayac_giris_p : process (clk, reset_n)
  begin
    if reset_n = '0' then
      sayac_giris <= 0;
    elsif rising_edge(clk) then
      if sink_valid = '1' and sink_ready = '1' then
        if sayac_giris = fft_uzunlugu - 1 then
          sayac_giris <= 0;
        else
          sayac_giris <= sayac_giris + 1;
        end if;
      end if;
    end if;
  end process sayac_giris_p;

-------------------------------------------------------------------------------------------------------------

   ---------------------------------------- ifft çıkışında bulunduğu framedeki kaçıncı verinin
olduğunu belirten sayacın oluşturulması-----------------------------------------------------------
-------------------------------------------------------------------------------------------------------------


  sayac_cikis_p : process (clk, reset_n)
  begin
    if reset_n = '0' then
      sayac_cikis <= 0;
    elsif rising_edge(clk) then
      if source_valid = '1' and source_ready = '1' then
        if sayac_cikis = fft_uzunlugu - 1 then
          sayac_cikis <= 0;
        else
          sayac_cikis <= sayac_cikis + 1;
        end if;
      end if;
    end if;
  end process sayac_cikis_p;

-------------------------------------------------------------------------------------------------------------


  -------- giriş framelerinin sayılması ve gerekli sinyaller tetiklendiğinde arttırılması-------------
-------------------------------------------------------------------------------------------------------------
  increment_index_p : process (clk, reset_n)
  begin
    if reset_n = '0' then
      frames_in_index <= 0;
    elsif rising_edge(clk) then
```

```
      if sink_eop = '1' and sink_valid = '1' and sink_ready = '1' then
        if frames_in_index < alt_tasiyici - 1 then
          frames_in_index <= frames_in_index + 1;
        end if;
      end if;
    end if;
  end process increment_index_p;
-------------------------------------------------------------------------------------------------------------

  ----- giriş framelerinin sayılması ve gerekli sinyaller tetiklendiğinde arttırılması----------------
-------------------------------------------------------------------------------------------------------------
  increment_out_index_p : process (clk, reset_n)
  begin
    if reset_n = '0' then
      frames_out_index <= 0;
    elsif rising_edge(clk) then
      if source_eop = '1' and source_valid = '1' and source_ready = '1' then
        if frames_out_index < alt_tasiyici - 1 then
          frames_out_index <= frames_out_index + 1;
        end if;
      end if;
    end if;
  end process increment_out_index_p;

-------------------------------------------------------------------------------------------------------------

 ----------------------simulasyonu bitiren end_output sinyalinin üretilmesi----------------------------
-------------------------------------------------------------
  end_output <= '1' when source_eop = '1' and source_valid = '1' and source_ready = '1' and
(frames_out_index = alt_tasiyici - 1) else
                '0';
-------------------------------------------------------------------------------------------------------------

-----------------------------paket başlangıcı ve paket sonu sinyallerini üret----------------------------
-------------------------------------------------------------------------------------------------------------
  sink_sop <= '1' when sayac_giris = 0 else
              '0';

  sink_eop <= '1' when sayac_giris = fftpts_array(frames_in_index) - 1 else
              '0';
-------------------------------------------------------------------------------------------------------------



---------------------- Modülasyon için giriş verilerin text dosyasından okunması--------------------
-------------------------------------------------------------------------------------------------------------
  testbench_giris : process(saat) is
    file dosya    : text open read_mode is "giris_verileri.txt";
    variable giris_dosya : std_logic_vector(3 downto 0);
    variable satir_dosya  : line;
```

```
 begin

   if (reset_negatif ='1') then

   if rising_edge(saat) then


     if not endfile(dosya) then
        readline(dosya, satir_dosya);
        read(satir_dosya, giris_dosya);

        data_giris  <= giris_dosya;
end if;

end if;
end if;

 end process testbench_giris;

-------------------------------------------------------------------------------------------------------

 ---------------IFFT çıkışlarının gerçel ve sanal iki farklı text dosyasına yazılması----------------
-------------------------------------------------------------------------------------------------------

 testbench_o : process(clk, reset_n) is
   file ro_file   : text open write_mode is "mod_ifft_reel.txt";
   variable rdata : line;
   variable data_r : std_logic_vector(32 - 1 downto 0);
   file io_file   : text open write_mode is "mod_ifft_sanal.txt";
   variable idata : line;
   variable data_i : std_logic_vector(32 - 1 downto 0);
 begin
   if rising_edge(clk) and reset_n = '1' then
     if(source_valid = '1' and source_ready = '1') then
       data_r := source_real;
       data_i := source_imag;
       hwrite(rdata, data_r);
       writeline(ro_file, rdata);
       hwrite(idata, data_i);
       writeline(io_file, idata);
     end if;
   end if;
 end process testbench_o;
-------------------------------------------------------------------------------------------------------

 ---------------------------------Test edilen tasarımların port aktarımlarının yapılması--------------
-------------------------------------------------------------------------------------------------------
dut1 : dogm
   port map (data_giris   => data_giris,
```

```
            cikis_sanal  => sink_imag,
            cikis_gercel => sink_reel,
cikis_var=> reset_n,
            clk => clk,

            reset_negatif=> reset_negatif);


  dut2 : ifft_64
    port map (
      clk          => clk,
      reset_n      => reset_n,
      fftpts_in    => fftpts_in,
      fftpts_out   => fftpts_out,
      sink_valid   => sink_valid,
      sink_sop     => sink_sop,
      sink_eop     => sink_eop,
      sink_real    => sink_real,
      sink_imag    => sink_imag,
      sink_error   => sink_error,
      source_error => source_error,
      source_ready => source_ready,
      sink_ready   => sink_ready,
      source_sop   => source_sop,
      source_eop   => source_eop,
      source_valid => source_valid,
      source_real  => source_real,
      source_imag  => source_imag);


end tb;
```

ÖZGEÇMİŞ

Ferdi KARA 1989 yılında Zonguldak'ta doğdu. İlköğretim ve Lise öğrenimini Çaycuma' da tamamladıktan sonra 2007 yılında Süleyman Demirel Üniversitesi Elektronik Haberleşme Mühendisliği Bölümü'nde lisans eğitimine başladı. 2011 Ağustos ayından itibaren Bülent Ecevit Üniversitesi Elektrik-Elektronik Mühendisliği Bölümü'nde Araştırma Görevlisi olarak çalışmaktadır.

ADRES BİLGİLERİ:

Adres: B.E.Ü. Elektrik-Elektronik Mühendisliği Bölümü
Farabi Kampüsü
İncivez-ZONGULDAK

Tel: (372) 257 40 10-1641
E-posta: f.kara@beun.edu.tr