EĞİTİM AMAÇLI SDR TEKNİKLERİNE DAYALI FPGA TABANLI GENLİK MODÜLELİ RADYO VERİCİSİ TASARIMI VE UYGULAMASI

Caner KİREMİTCİ

2021
YÜKSEK LİSANS TEZİ
ELEKTRİK ELEKTRONİK MÜHENDİSLİĞİ

Tez Danışmanı
Dr. Öğr. Üyesi Bilgehan ERKAL

EĞİTİM AMAÇLI SDR TEKNİKLERİNE DAYALI FPGA TABANLI GENLİK MODÜLELİ RADYO VERİCİSİ TASARIMI VE UYGULAMASI

Caner KİREMİTCİ

T.C.
Karabük Üniversitesi
Lisansüstü Eğitim Enstitüsü
Elektrik Elektronik Mühendisliği Anabilim Dalında
Yüksek Lisans Tezi
Olarak Hazırlanmıştır

Tez Danışmanı
Dr. Öğr. Üyesi Bilgehan ERKAL

KARABÜK

Ocak 2021

Caner KİREMİTCİ tarafından hazırlanan "EĞİTİM AMAÇLI SDR TEKNİKLERİNE DAYALI FPGA TABANLI GENLİK MODÜLELİ RADYO VERİCİSİ TASARIMI VE UYGULAMASI" başlıklı bu tezin Yüksek Lisans Tezi olarak uygun olduğunu onaylarım.

Dr. Öğr. Üyesi Bilgehan ERKAL .....
Tez Danışmanı, Elektrik Elektronik Mühendisliği Anabilim Dalı

Bu çalışma, jürimiz tarafından oy birliği ile Elektrik Elektronik Mühendisliği Anabilim Dalında Yüksek Lisans tezi olarak kabul edilmiştir. 19/01/2021

Ünvanı, Adı SOYADI (Kurumu)	İmzası
Başkan : Doç. Dr. Hüseyin DEMİREL (KBÜ)	.....
Üye : Doç. Dr. Salih GÖRGÜNOĞLU (KÜ)	.....
Üye : Dr. Öğr. Üyesi Bilgehan ERKAL (KBÜ)	….....

KBÜ Lisansüstü Eğitim Enstitüsü Yönetim Kurulu, bu tez ile, Yüksek Lisans derecesini onamıştır.

Prof. Dr. Hasan SOLMAZ .....
Lisansüstü Eğitim 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."

Caner KİREMİTCİ

ÖZET

Yüksek Lisans Tezi

EĞİTİM AMAÇLI SDR TEKNİKLERİNE DAYALI FPGA TABANLI GENLİK MODÜLELİ RADYO VERİCİSİ TASARIMI VE UYGULAMASI

Caner KİREMİTCİ

Karabük Üniversitesi
Lisansüstü Eğitim Enstitüsü
Elektrik Elektronik Mühendisliği Anabilim Dalı

Tez Danışmanı:
Dr. Öğr. Üyesi Bilgehan ERKAL

Ocak 2021, 102 sayfa

Yazılım tanımlı radyo sistemlerinde temel amaç, radyo işaretlerini bir sayısal işaret işleyiciyle tamamen sayısal olarak işlemektir. Bu sistemler üzerinde modülasyon, demodülasyon, işaret üretimi ve hat kodlaması gibi işlemlerin alanda programlanabilir kapı dizileri gibi bir işlemci ile yapılması analog devre temelli donanıma duyulan ihtiyacı büyük ölçüde azaltmaktadır. Alanda Programlanabilir Kapı Dizileri ise, programlanabilir mantık blokları arası bağlantılarla meydana gelen ve fazla sayıda uygulama alanları olan sayısal bütünleşik devrelerdir. Tasarımcı ihtiyacına yönelik mantık işlevlerini uygulayabilme amacıyla üretilmiştir. Bundan dolayı her mantık bloğunun işlevselliği tasarımcı tarafından düzenlenebilir. FPGA'in programlanması aşamasında ise genellikle VHDL kullanılır. Bu çalışma üzerinde ilk olarak FPGA'de modellenecek genlik modüleli radyo vericisinin simülasyonu matlab kodlarıyla oluşturulmuştur. Ardından VHDL kodu, ISE Design Suite 14.7 üzerinde yazılarak,

genlik modüleli verici, FPGA kartı (Mimas Spartan 6) üzerinde gerçekleştirilmiştir. Audacity programı ile modülasyonda kullanılacak örnek ses kaydı, ses kartı aracılığı ile FPGA kartına gönderilmiştir. FPGA kartı, ADC (LM4550) kartı üzerinden analog sinyali alarak HDSDR programı ile verici sinyali alınıp, demodüle edilip, kaydedilmiştir. FPGA kartı, DAC (LM4550) kartı aracılığı ile verici sinyalini analog formda üretip, laptopun ses kartı mikrofon girişine göndermiştir. Ve son olarak kaydedilmiş verici sinyali ayrıca matlab koduyla da offline olarak demodüle edilip sonuç harddiske kaydedilmiştir. Verilerin incelenmesi neticesinde, aynı test sinyali için simülasyon sonuçları ile reel test sonuçları arasında çok küçük miktar fark olduğu görülmektedir. Bu durum, reel testlerde sinyalin harici gürültüye maruz kaldığı göz önünde bulundurulursa gayet makul bir durum olarak kabul edilebilir. Aynı şekilde, elde edilen SNR değerleri incelendiğinde ise ortalama 20dB civarında bir değer görülmektedir ki bu AM modülasyonu için kabul edilebilir bir değerdir. Buradan yola çıkarak tasarlanan FPGA AM verici sisteminin AM yayınlarını başarıyla gerçekleştirebilecek kapasitede olduğu görülmektedir. Sonuç olarak, SDR sistemlerinin FPGA üzerinde gerçekleştirilmesine ve eğitimine yönelik güzel bir platform elde edilmiştir.

Anahtar Sözcükler : SDR, FPGA, genlik modülasyonu, radyo vericisi, VHDL
Bilim Kodu **:** 90523

ABSTRACT

M. Sc. Thesis

DESIGN AND IMPLEMENTATION OF AN EDUCATIONAL AM TRANSMITTER WITH FPGA USING SDR TECHNIQUES

Caner KİREMİTCİ

Karabük University

Institute of Graduate Programs

Department of Electrical and Electronics Engineering

Thesis Advisor:

Assist. Prof. Dr. Bilgehan ERKAL

January 2021, 102 pages

In software defined radio systems, the main purpose is to process the radio signals completely numerically with a digital signal processor. Performing operations such as modulation, demodulation, signal generation and line coding on these systems with a processor such as field programmable gate arrays greatly reduces the need for analog circuit-based hardware. Field Programmable Gate Arrays are digital integrated circuits consisting of connections between programmable logic blocks and have wide application areas. It is produced for the purpose of realizing logic functions for the needs of the designer. Therefore, the function of each logic block can be edited by the user. In the programming phase of FPGA, VHDL is generally used. In this study, firstly, the simulation of the amplitude modulated radio transmitter to be modeled in FPGA was created with matlab codes. Then the VHDL code was written on ISE Design Suite 14.7 and the amplitude modulated transmitter was implemented on the FPGA board (Mimas Spartan 6). The sample sound recording to be used in modulation with the Audacity program was sent to the FPGA card via the sound card. FPGA card

received analog signal from ADC (LM4550) card and transmitter signal was received, demodulated and recorded with HDSDR program. FPGA card generated the transmitter signal in analog form via DAC (LM4550) card and sent it to the microphone input of the laptop's sound card. And finally, the recorded transmitter signal was demodulated offline with matlab code and the result was saved to the hard disk. As a result of the analysis of the data, it is seen that there is very little difference between the simulation results and the real test results for the same test signal. This can be regarded as a perfectly reasonable situation considering that the signal is exposed to external noise in real tests. Likewise, when the obtained SNR values are examined, an average value of around 20dB is seen, which is an acceptable value for AM modulation. Based on this, it is seen that the FPGA AM transmitter system, which is designed, is capable of successfully realizing AM broadcasts. As a result, a good platform for the implementation and training of SDR systems on FPGA has been obtained.

Key Word **:** SDR, FPGA, amplitude modulation, radio transmitter, VHDL
Science Code : 90523

TEŞEKKÜR

Bu tez çalışmasının planlanmasında, araştırılmasında, ortaya koyulmasında ilgi ve desteğini esirgemeyen, bilgi ve tecrübelerinden yararlandığım, yönlendirme ve bilgilendirmeleriyle çalışmama bilimsel temeller ışığında yön veren saygıdeğer hocam Dr. Öğr. Ü. Bilgehan ERKAL'a teşekkürlerimi sunarım.

İÇİNDEKİLER

Sayfa

KABUL ii
ÖZET iv
ABSTRACT ..... **Hata! Yer işareti tanımlanmamış.**
TEŞEKKÜR viii
İÇİNDEKİLER ix
ŞEKİLLER DİZİNİ xiiii
ÇİZELGELER DİZİNİ xiiii
SİMGELER VE KISALTMALAR DİZİNİ xvi

BÖLÜM 1 1
GİRİŞ 1

BÖLÜM 2 6
YAZILIM TANIMLI RADYO 6

BÖLÜM 3 8
ALANDA PROGRAMLANABİLİR KAPI DİZİLERİ 8
3.1. FPGA İÇ YAPISI 8
3.1.1. Mantık Hücresi 9
3.1.2. FPGA Pinleri 9
3.1.3. RAM Blokları 10
3.2. FPGA PROGRAMLAMA 10
3.3. FPGA AKIŞ DİYAGRAMI 11
3.4. FPGA ÜRETİCİLERİ 11
3.4.1. Xilinx 11
3.4.2. Altera 12

Sayfa

BÖLÜM 4 13
GENLİK MODÜLASYONU 13
4.1. ÇİFT YAN BANT GENLİK MODÜLASYONU 14
4.1.1. Bilgi Sinyali 15
4.1.2. Taşıyıcı Sinyal 15
4.1.3. Modüleli Sinyal 16
4.1.4. Genlik Modülasyonun Matematiksel İfadesi 16
4.1.5. Bant Genişliği 17
4.2. TEK YAN BANT GENLİK MODÜLASYONU 18
4.2.1. Frekans Ayrım Yöntemi 18
4.2.2. Faz Kaydırma Yöntemi 18

BÖLÜM 5 20
GENLİK MODÜLELİ RADYO VERİCİSİ 20

BÖLÜM 6 22
VHDL – DONANIM TANIMLAMA DİLİ 22
6.1. VHDL TERMİNOLOJİSİ 22
6.1.1. Davranışsal Modelleme 22
6.1.2. Yapısal Modelleme 23
6.1.3. RTL (Register Transfer Level) 23
6.2. VHDL TASARIMI 24
6.3. VHDL TASARIM BÖLÜMLERİ 25
6.3.1. Entity 25
6.3.2. Mimari (Archtitecture) 25
6.3.3. Paket (Package) 25
6.3.4. Bileşen (Component) 25
6.3.5. İşlem (Process) 25
6.4. VHDL MODELLEME TEMELLERİ 26
6.4.1. Sabit (Constant) 26
6.4.2. Sinyal 26
6.4.3. VHDL Operatörleri 27

Sayfa

6.4.4. Eşzamanlı Sinyal Atamaları 27
6.4.5. Sıralı Komutlar 28

BÖLÜM 7 29
MATERYAL VE METOTLAR 29

7.1. FPGA TABANLI AM VERİCİ TASARIMI VE UYGULAMASI 29
7.1.1. Donanım Bileşenleri 29
7.1.1.1. Mimas Spartan 6 FPGA Katı 31
7.1.1.2. LM4550 Ses Kartı 32
7.1.1.3. IO Genişletme Kartı 33
7.1.2. Programlar 35
7.1.2.1. Audacity 35
7.1.2.2. HDSDR 36
7.1.2.3. Xilinx ISE Webpack 36
7.1.2.4. Matlab 37
7.1.3. Matlab Kodları 37
7.1.4. VHDL Kodu ve Sistemin Blok Şeması 39
7.1.5. Sistemin RTL Diyagramları 43

BÖLÜM 8 45
SONUÇLAR VE TARTIŞMA 45

BÖLÜM 9 49
SONUÇ 49
KAYNAKLAR 51

EK AÇIKLAMALAR A. VERİ SAYFALARI. 51

EK AÇIKLAMALAR B. MATLAB KOD LİSTELERİ. 58
Ek B.1. AM Modülasyon Simülasyon Kodu 59
Ek B.2. Demodülasyon Kod Listesi 60
Ek B.3. Analiz Kodu 62

Sayfa

Ek B.4. Filtre Tasarım Kodu 63
EK AÇIKLAMALAR C. VHDL KOD LİSTELERİ. 64
Ek C.1. AM Verici Üst Modülü 65
Ek C.2. AM_TX AM Verici Alt Modülü 74
Ek C.3. frecalc Faz Artım Hesaplama Alt Modülü 79
Ek C.4. sercomrx Seri Veri Alıcı Alt Modülü 81
Ek C.5. sercomtx Seri Veri Vericisi Alt Modülü 89
Ek C.6. AC'97_ADAC LM4550 Ses Kartı Yönetim Alt Modülü 97

ÖZGEÇMİŞ. 102

ŞEKİLLER DİZİNİ

Sayfa

Şekil 2.1. Yazılım tanımlı radyo mimarisi; a (verici), b (alıcı) 7
Şekil 3.1. FPGA iç yapısı 9
Şekil 3.2. Mantık hücresi 9
Şekil 3.3. FPGA akış şeması. 11
Şekil 4.1. Bilgi sinyali, taşıyıcı sinyal ve modüleli sinyal. 14
Şekil 4.2. Çift yan bant genlik modüleli verici blok şeması 14
Şekil 4.3. Bilgi sinyali. 15
Şekil 4.4. Taşıyıcı sinyali. 15
Şekil 4.5. Modüleli işarette bilgi işaretine ait büyüklükler. 16
Şekil 4.6. Çift yan bant genlik modülasyonu frekans spektrumu 17
Şekil 4.7. Tek yan bant genlik modüleli verici blok şeması 18
Şekil 4.8. TYB faz kaydırma yöntemi blok şeması 19
Şekil 5.1. Yüksek ve alçak seviyeli modülatör vericileri 20
Şekil 5.2. Genlik modüleli stereo vericinin blok diyagramı. 21
Şekil 6.1. Davranışsal modelleme. 22
Şekil 6.2. Yapısal modelleme. 23
Şekil 6.3. Örnek RTL modelleme. 24
Şekil 6.4. VHDL tasarım akışı. 24
Şekil 7.1. Sistemin donanımsal yapısının blok şeması 29
Şekil 7.2. Sistemin genel görünüşü : FPGA kartı (sol), IO genişletme kartı (orta), LM4550 ses kartı (sağ). 31
Şekil 7.3. Mimas Spartan 6 FPGA geliştirme kartı. 32
Şekil 7.4. FPGA kart bağlantı şeması. 32
Şekil 7.5. LM4550 ses kartı. 33
Şekil 7.6. LM4550 bağlantı şeması. 33
Şekil 7.7. LM4550 pin bağlantı şeması. 33
Şekil 7.8. IO genişletme kartı. 34
Şekil 7.9. IO bağlantı diyagramı. 34
Şekil 7.10. IO pin bağlantı şeması 35
Şekil 7.11. FPGA AM verici sistemi blok şeması 40

Sayfa

Şekil 7.12. Üst modül AMTX'in RTL diyagramı. 43

Şekil 7.13. Alt modül am_tx'in RTL diyagramı. 44

Şekil 8.1. FPGA AM verici sisteminin A1 modüle edici sinyaliyle yapılan test sonuçları: üstte A1 test sinyali, ortada modülasyonu takiben demodülasyonla elde edilen dalga şekli, altta iki sinyalin farkı. 46

Şekil 8.2. FPGA AM verici sisteminin A1 modüle edici sinyaliyle yapılan simülasyon sonuçları: üstte A1 test sinyali, ortada modülasyonu takiben demodülasyonla elde edilen dalga şekli, altta iki sinyalin farkı. 46

Şekil 8.3. FPGA AM verici sisteminin A2 modüle edici sinyaliyle yapılan test sonuçları: üstte A2 test sinyali, ortada modülasyonu takiben demodülasyonla elde edilen dalga şekli, altta iki sinyalin farkı. 47

Şekil 8.4. FPGA AM verici sisteminin A2 modüle edici sinyaliyle yapılan simülasyon sonuçları: üstte A2 test sinyali, ortada modülasyonu takiben demodülasyonla elde edilen dalga şekli, altta iki sinyalin farkı. 47

Şekil Ek A.1. Mimas Spartan 6 (XC6SLX9-3TQG144). 54

Şekil Ek A.2. LM4550 Audio Codec. 56

ÇİZELGELER DİZİNİ

Sayfa

Çizelge 8.1. Deney ve simülasyon sonuçları 47

SİMGELER VE KISALTMALAR DİZİNİ

SİMGELER

v_m : mesaj sinyali anlık ac değeri

V_m : mesaj sinyali maksimum değeri

f_m : mesaj sinyali frekansı

v_c : taşıyıcı sinyalin anlık ac değeri

V_c : taşıyıcı sinyalin maksimum değeri

f_c : taşıyıcı sinyalin frekansı

V : modüleli sinyal

m : modülasyon indisi

KISALTMALAR

SDR : Software Defined Radio (Yazılım Tanımlı Radyo)

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

VHDL : Very High-Speed Integrated Circuit Hardware Description Language (Yüksek Hızlı Tümleşik Devreler için Donanım Tanımlama Dili)

ICNIA : Integrated Communication Navigation Identification and Avionics System (Entegre İletişim Navigasyon Tanımlama ve Aviyonik Sistem)

DSP : Digital Signal Processing (Sayısal Sinyal İşleme)

MMITS : Modular Multifunction Information Transfer System (Modüler Çok Fonksiyonlu Bilgi Aktarım Sistemi)

DUC : Digital Up Converter (Sayısal Yukarı Çevirici)

IF : Intermediate Frequency (Orta Frekans)

DAC : Digital Analog Converter (Sayısal Analog Çevirici)

ADC : Analog Digital Converter (Analog Sayısal Çevirici)

DDC : Digital Down Converter (Sayısal Aşağı Çevirici)

FIR : Finite Impulse Response (Sonlu Darbe Tepkisi)

CPLD : Complex Programmable Logic Device (Karmaşık Programlanabilir Mantık Cihazı)

PROM : Programmable Read Only Memory (Programlanabilir Salt Okunur Bellek)

AM : Amplitude Modulation (Genlik Modülasyonu)

DSB-AM : Double Side Band-Amplitude Modülation (Çift Yan Bant Genlik Modülasyonu)

SSB-AM : Single Side Band-Amplitude Modülation (Tek Yan Bant Genlik Modülasyonu)

RTL : Register Transfer Level (Kayıt Aktarım Seviyesi)

HDSDR : High Definition Software Defined Radio (Yüksek Çözünürlüklü Yazılım Tanımlı Radyo)

BÖLÜM 1

GİRİŞ

Yazılım tanımlı radyo (SDR), kablosuz haberleşme yapmak üzere düşünülmüş ve konfigürasyonu yeniden düzenlenebilen bir donanım ve yazılım teknolojileri koleksiyonudur. Yazılım tanımlı radyoya yönelik ilk fikirler, 1991 yılında John Mittola tarafından radyoların yazılımsal olarak düzenlenebileceği ve yeniden programlanabileceği fikriyle ortaya konulmuştur. Klasik bir donanım tabanlı radyo sisteminde sinyal indirgeme/yükseltme, modülasyon, filtreleme ve diğer sinyal şekillendirme işlemleri donanım elemanları üzerinden gerçekleştirilir. İdeal bir yazılımsal radyoda bu elemanların yerini kullanıcının istediği anda değiştirebileceği yani programlanabilir bir sistem yer alır. SDR sistemlerinin geliştirilmesiyle birlikte donanım tabanlı radyolara göre maliyetin düşürülmesi, işlevselliğin değiştirilebilmesi gibi avantajlar sağlamıştır.

Alanda Programlanabilir Mantık Dizileri (FPGA), programlanabilir mantık blokları arasında bulunan ara bağlantılar ile meydana gelen ve fazla sayıda uygulama alanları olan sayısal devrelerdir. Kullanıcının ihtiyacına yönelik mantık işlevlerini gerçekleştirme amacı ile üretilmiştir. Bundan dolayı mantık bloklarının işlevleri tasarımcı tarafından düzenlenebilir. FPGA kullanarak temel mantık kapılarının ve karmaşık devre elemanlarının işlevselliği gözle görülür biçimde artmaktadır.

FPGA programlamak için; grafiksel tasarım ve VHDL yöntemleri kullanılır. Grafiksel tasarım, derleyici program kütüphanesinde bulunan araç ve mantık kapılarından faydalanarak yapılır. VHDL ise en yaygın olarak kullanılan programlama türüdür. VHDL 1980'lerden beri sürekli gelişmektedir ve IEEE'de standart olarak almaktadır.

FPGA devrelerinin SDR sistemlerinde kullanılma nedenleri ise; ilk olarak, aynı anda birden fazla parallel işlem gerçekleştirebilme kapasitesine sahiptir ve yüksek sayıda

giriş-çıkış ünitesi vardır.Ayrıca FPGA istenildiği gibi yeniden programlanabilir. FPGA ve SDR sistemleri birlikte kullanılarak bu özellikleriyle ile birlikte ayrıca çeşitli modülasyon yöntemlerini daha iyi alma ve iletme imkanı sağlarlar. Bu bilgiler doğrultusunda, genlik modüleli radyo vericisi, SDR teknikleri ve FPGA devresi ile birlikte VHDL kodları kullanılarak tasarlanıp uygulamaya alınacaktır.

FPGA'lerin kullanım alanlarına bakıldığında; uzay, havacılık ve savunma sektöründen iletişim sektörüne, ses ve görüntü işlemeden, veri depolamaya kadar birden çok alanda son zamanlarda birçok kez çalışmalar yapıldığı görülmektedir. Yazılım tanımlı radyo mimarisinde ise özellikle haberleşme alanında, telsiz, radyo gibi cihazların maliyetini düşürebilmek amacıyla çalışmalar yapılmaktadır. Bu çalışmadaki başlıklar kapsamında bir literatür araştırması yapıldığında, SDR teknikleri ile FPGA tabanlı çalışmaların 2000'li yılların ortalarından itibaren başlamasıyla birlikte özellikle son 10 yıldır çalışmaların ve araştırmaların sıklaştığı görülmektedir.

2016 yılında A. Gareane'nin gerçekleştirdiği 'Transmit and Receive of FM Signals Using Softrock SDR and MATLAB' adlı çalışmada Softrock Ensemble RXTX SDR alıcı-vericileri kullanılarak FM sinyallerinin alımı ve iletimi araştırılmıştır. FM modülasyon/demodülasyonu Matlab üzerinden gerçekleştirilen ve Audacity üzerinden görüntülenen çalışmada olumlu sonuçlar alındığı ve diğer sinyal tipleri içinde çalışmalar yapılabileceği belirtilmiştir [1].

2016 yılında Hervas, Alsina-Pages ve Salvador gerçekleştirdikleri 'An FPGA Scalable SDR Platform Design for Educational and Research Purposes' isimli çalışmada ise, analog dönüştürücülerin, çekirdek işlemciler ile entegrasyonunda yaşadıkları sıkıntılardan dolayı, eğitim amaçlı Spartan 6 tabanlı bir FPGA ölçeklenebilir SDR platformu oluşturmuşlardır. IRIS adını verdikleri bu platform yüksek düzey ölçeklenebilirlik ve bağlanabilirlik sağlayan kompakt bir SDR platformudur. Bu platform, ADC performans değerleri sinyal bütünlüğü ve emc değerleri bakımından doğru bir tasarım olarak görülmektedir. Burada sonuç olarak ADC ve DAC entegrasyonu sıkıntılarını yok eden ve daha az maliyetli bir çalışma gerçekleştirmişlerdir [2].

2017 yılında Cai, Zhou ve Huang ortaya koydukları ‘Model-Based Design for SDR on an FPGA’ isimli çalışma ile FPGA donanım hedefli bir SDR için model tabanlı tasarım kullanma prosedürü oluşturmuşlardır. Evrişimsel kodlayıcı içeren bir verici ve Viterbi kod çözücülü bir alıcı ile oluşturulan bir dijital iletişim sistemini, FPGA tabanlı bir SDR platformunda uygulamışlardır. Burada verici için QPSK modülasyonu kullanılarak, hem verici hem de alıcı da HDL tabanlı bir kodlama yapılmıştır. Alıcıdaki Viterbi kod çözücü ile vericideki evrişimsel kodlayıcı uygun hale getirilerek, birbirine yakın reel ve simülasyon test sonuçları elde etmişlerdir [3].

Yine 2017 yılında Tsoeunyane, Winberg ve Inggs tarafından gerçekleştirilen ‘SDR FPGA Cores: Building Towards a Domain-Spesific Language’ isimli çalışmada, FPGA tabanlı bilgi işlem platformlarında, SDR uygulamalarının geliştirilmesi için tasarlanmış, parametrelendirilebilir ve yeniden kullanılabilir HDL ve IP çekirdeklerinden oluşan açık kaynaklı bir kütüphane tasarımı üzerinde çalışmışlardır. Burada parametreleri ve arayüzü belirlenen bir dizi SDR çekirdeği önce doğrulanıp ardından buna benzer 2 farklı çekirdek işlemci ile oluşturulmuş kütüphane ile karşılaştırılmıştır ve burada elde edilen sonuçlarda SDR işlemcisinin Xilinx işlemcisine göre daha hızlı performans verdiği ve daha az maliyetli bir kütüphane tasarımı yapıldığı görülmüştür [4].

Son olarak 2019 yılında Haggui, Affes ve Bellili tarafından yapılan ‘FPGA-SDR Integration and Experimental Validation of a Joint DA ML SNR and Doppler Spread Estimator for 5G Cognitive Transceivers’ isimli çalışma ele alındığında, 5G çalışmalarının başlamasıyla birlikte, bilişsel radyo (CR) olarak bilinen yapının FPGA ve SDR entegrasyonu üzerinde çalıştıkları görülmektedir. CR alıcı-vericileri için en önemli iki parametre sinyal-gürültü oranı (SNR) ve doppler yayılması ele alınarak, bu ikisi üzerinden gerçekleştirilen donanım tasarımı ve entegrasyonunu ele almışlardır. Bu tasarım sonucu elde edilen prototip ile son derece ölçeklenebilir bir kanal emülatörü tarafından yeniden üretilen gerçekçi yayılma koşulları altında gerçek zamanlı test etmişlerdir. Sonuçlar neticesinde, performans kayıpları yaşamalarına ragmen gelecekte 5G CR’lerine entegrasyonu için çok güçlü potansiyel ortaya koymuştur [5].

Yukarıda belirtilen çalışmalara bakıldığında, SDR entegreli FPGA tabanlı farklı çalışmaların ortaya koyulduğu; softrock ensemble katmanlı yapılardan açık kaynak kütüphanelerine, FPGA üzerinde SDR tabanlı bir model oluşturmaktan, SDR teknikleri FPGA tabanlı ile 5G teknolojisi entegrasyonuna kadar, FPGA-SDR birlikte kullanımını birden fazla alanda kullanabilmeyi hedefleyen çalışmalar yapılmıştır. Bu çalışmanın amacı da, yukarıda verilen bilgiler doğrultusunda, daha önce çalışılmayan bir kapsam olarak, genlik modüleli radyo vericisini, SDR teknikleri ve FPGA devresi ile birlikte tasarlayıp; daha hızlı çalışan ve aynı zamanda tekrar programlanabilir olduğu için daha düşük maliyetli bir tasarım ortaya koymaktır.

Ortaya koyulan çalışma, genel itibari ile teorik ve tasarım kısmı olarak iki kısımdan oluşmaktadır. Teorik ve tasarım kısmı da yine kendi içinde başlıklara ayrılmıştır. İlk bölüm olarak "Giriş" bölümünde, görüldüğü üzere çalışmanın kısa bir özeti ile birlikte literatür taramasına yer verilmiştir. İkinci bölümde, çalışmada kullanılan yazılım tanımlı radyo sistemleri ile ilgili teorik kapsamlardan bahsedilmiştir. Üçüncü bölümde alanda programlanabilir mantık dizilerinin iç yapısı, programlama modelleri, akış diyagramları ve üreticileri ile ilgili bilgiler verilmiştir.

Dördüncü bölümde, genlik modülasyonu başlığı altında, çift yan bant ve tek yan bant genlik modülasyonu hakkında teknik bilgiler anlatılırken, beşinci bölümde ise genlik modüleli radyo vericisinin oluşturulmasına dair bilgiler verilmiştir.. Altıncı bölümde, VHDL programlama dilinin terminolojisinden, modelleme çeşitlerinden, tasarımından ve tasarımı modelleyebilmek için gerekli temellerden bahsedilmiştir.

Çalışmanın yedinci bölümünde ise, FPGA tabanlı AM verici tasarımının oluşturulma aşamalarından bahsedilmiştir. Burada ilk olarak kullanılan donanım bileşenleri ve bu bileşenler arasında kurulan bağlantılardan bahsedilmiştir. Devamında, kullanılan programlar ile birlikte, tasarım simülasyonu için oluşturulan Matlab kodları anlatılmıştır. Üçüncü olarak ise, kurulan FPGA tabanlı sistemin blok diyagramı ile birlikte detaylarından bahsedilerek, aynı zamanda kullanılan VHDL kodları anlatılmıştır.

Çalışmanın sekizinci bölümünde elde edilen reel sonuçlar ile simülasyon üzerinden elde edilen sonuçlar karşılaştırılarak, oluşturulan tablo ile birlikte genel bir çıkarım sağlanmıştır. Çalışmanın son ve dokuzuncu bölümünde ise, elde edilen test sonuçlarıyla birlikte diğer sonuçlar yorumlanarak, bu çalışmada nasıl bir fayda sağlanıldığından bahsedilmiştir.

BÖLÜM 2

YAZILIM TANIMLI RADYO

Yazılım tanımlı radyo (SDR) sistemlerinde temel amaç radyo işaretlerini bir sayısal işaret işleyiciyle tamamen sayısal olarak işlemektir. Bu sistemlerde modülasyon, demodülasyon, işaret üretimi ve hat kodlaması gibi işlemleri sayısal işaret işleyebilen sayısal işaret işleyici (DSP) ve alanda programlanabilir kapı dizileri (FPGA) gibi bir işlemci ile yapılması analog devre temelli donanıma duyulan ihtiyacı azaltmaktadır [6].

İlk olarak 1970'lerin başında haberleşme mühendisleri tarafından programlanabilen yazılım bazlı cihazlar üzerinde çalışılmaya başlandı. Bu yıllarda Birleşik Devletler ICNIA adında bir sistem üretti. Bu sistem DSP bazlı programlanabilen ve modem kullanan ilk sistemlerden biriydi.

1980'lerde ise programlanabilen sayısal baz bant radyoları prototiplenmeye başlandı. 'Yazılımsal radyo' ifadesi ilk olarak 1984 yılında E-Systems şirketinde çalışan bir ekip tarafından ortaya atıldı. Bu ifade, ekibin ürettiği bir sayısal baz bant alıcısı için kullanılmıştı. 1992 yılına gelindiğinde Joseph Mittola, GSM baz istasyonu projesini ifade etmek için 'yazılımsal radyo' terimini ortaya çıkardı. 1984 yılındaki ortaya koyulan ifadeden farklı olarak burada Joseph Mittola'nın yazılımsal radio terimini alıcı ve vericiyi de içeren bütün bir sistem için kullanmış olmasıydı [7].

1996 yılına gelindiğinde SDR'ye adına ilk dernek kuruldu. İlk başta ismi MMITS Forum olsa da ismi 1998 yılında ismi SDR Forum olarak değiştirlmiştir.

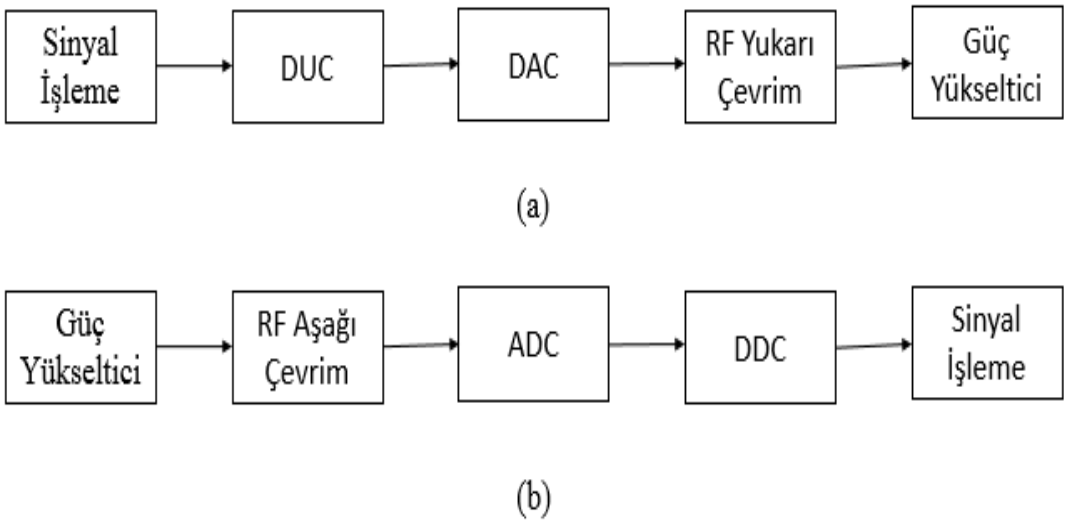
SDR yapısal olarak analog ve sayısal iki alt sistemden oluşur. Analog sistem, RF band geçirici filtre, anteni alıcı ve verici olarak pozisyonlayan mikrodalga anahtar, düşük gürültülü kuvvetlendirici, RF güç kuvvetlendirici ve referans frekans üretecinden

oluşmaktadır. Analog sistem sayısal olarak gerçekleştirilemeyen modülleri içermektedir [8].

Sayısal sistem ise donanım üzerinde çalışan bir yazılımdan meydana gelmektedir. Yazılım, donanımın yazılımdan ayrılabilmesi için katmanlı bir yapıdadır. Katmanlı yapıyı oluşturmak için ise özel yazılımlar oluşturulur. Yazılım işletim sistemi ise donanım sürücüleri, kaynak yönetimleri gibi yazılımları içerir. Şekil 2.1'de yazılım tanımlı radyo mimarisi gösterilmiştir [9].

Sistemin çalışma sürecine bakıldığında ise ilk olarak vericide sayısal veri kodlanır ve module edilir. Veri sonrasında sayısal yukarı çeviriciye (DUC) sokulur ve sayısal baz bant örnekleri orta frekans (IF) örneklerine çevrilir. Bu örnekler, sayısal analog çeviriciye (DAC) aktarılır ve analog bir IF sinyali elde edilir. Daha sonra bu sinyal RF yukarı çeviriciye sokularak RF sinyali elde edilir.

Alıcıda ise ilk olarak gelen RF sinyali yükseltilir ve devamında analog IF sinyaline çevrilir. Analog sayısal çevirici (ADC) ise IF sinyalini sayısal örneklere dönüştürür. Sonrasında sinyal, sayısal aşağı çeviriciye (DDC) sokulur ve gelen sinyal, baz bant sinyale dönüşür. Devamında son olarak, sonlu darbe tepkisi (FIR) filtresi ile sinyalin bant genişliği sınırlandırılır [10].



Şekil 2.1. Yazılım tanımlı radyo mimarisi; (a) verici, (b) alıcı

BÖLÜM 3

ALANDA PROGRAMLANABİLİR KAPI DİZİLERİ

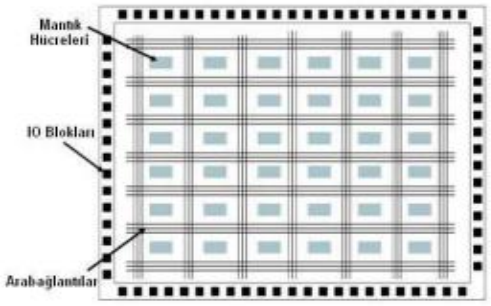
Alanda Programlanabilir Mantık Dizileri (FPGA), programlanabilir mantık blokları arasında bulunan ara bağlantılar ile meydana gelen ve fazla sayıda uygulama alanları olan sayısal devrelerdir.

Kullanıcının ihtiyacına yönelik mantık işlevlerini gerçekleştirme amacı ile üretilmiştir. Bundan dolayı mantık bloklarının işlevleri tasarımcı tarafından düzenlenebilir. FPGA kullanarak temel mantık kapılarının ve karmaşık devre elemanlarının işlevselliği gözle görülür biçimde artmaktadır [11].

Tarihsel sürecine bakıldığında ilk olarak 1980'li yıllarda FPGA'lerden genellikle ara yapıştırıcı mantık ve kısıtlı veri işleme görevlerinde faydalanıldı. 1990'lı yıllara gelindiğinde ise artan kapasiteleri sayesinde geniş very işlemleri gerektiren ağ ve haberleşme ortamlarında kullanılmaya başlandı. 90'ların sonunda ise otomotiv ve endüstriyel sektörlerdeki kullanımı büyük bir büyüme gösterdi. 2000'lerin başında ise milyonlarca kapı içeren yüksek performanslı FPGA'ler piyasaya girdi ve günümüzde de birden fazla pazar kolunda oldukça geniş yer bulmaktadırlar [12], [13].

3.1. FPGA İÇ YAPISI

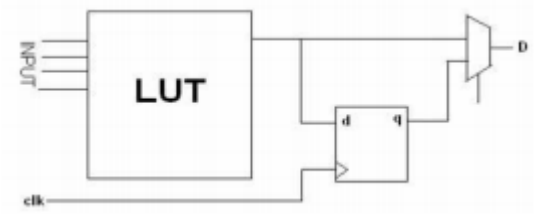
FPGA'in iç yapısı, giriş-çıkış blokları, ara bağlantılar ve mantık hücresi olmak üzere üç kısımdan oluşur. Şekil 3.1'de FPGA'in iç yapısı gösterilmiştir.



Şekil 3.1. FPGA'in iç yapısı

3.1.1. Mantık Hücresi

Mantık hücresi FPGA'in temel yapısıdır. Birer adet D tipi FF ve LUT ile bir adet 2x1 Mux'dan oluşur. Şekil 3.2'de mantık hücresi gösterilmiştir.



Şekil 3.2. Mantık hücresi

LUT'lar mantık işlemlerini gerçekleştiren mini belleklerdir. N girişli bir LUT 2^N'li bir bellek oluşturur. Mantık hücrelerindeki ara bağlantılar matris şeklindeki veri yolları ve programlanabilir anahtarlar ile gerçekleşmektedir.

3.1.2. FPGA Pinleri

FPGA pinleri genellikle ayrılmış pinler ve kullanıcı pinleri olmak üzere ikiye ayrılır. Ayrılmış pinler ise fonksiyonlarına göre; güç pinleri, konfigürasyon pinleri ve clock pinleri olmak üzere üç kategoriye ayrılır. Güç pinleri, FPGA'in enerji gereksinimini sağlar. Konfigürasyon pinleri, programın FPGA'e yüklenmesini sağlar. Clock pinleri ise saat sinyalleri için ayrılmıştır. Kullanıcı pinleri de tasarımcının ayarladığı standart olan giriş/çıkış pinleridir [14].

3.1.3. RAM Blokları

FPGA'lerde RAM ayrı bir biçimde bulunur. Bu RAM'ler mantık devresinin çalışması esnasında duyulan geçici hafıza gereksinimi için hazırdır. Tekli ve çoklu erişimi desteklerler. Multi erişimde birçok işlem, bu bellek üniteleri üzerinde okuma ve yazma yapabilir.

3.2. FPGA PROGRAMLAMA

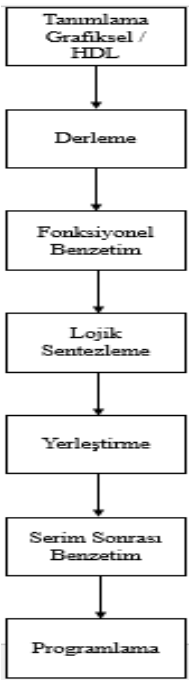
FPGA'in programlanması aşamasında kullanılan yöntemler Grafiksel Tasarım ve HDL'dir. Grafiksel tasarım, derleyici program kütüphanesindeki element ve mantık kapılarak kullanılarak programlanır. HDL'de ise tasarım; Verilog veya VHDL kullanılarak programlanır [15], [16].

VHDL, aynı zamanda çok yüksek hızlı tümleşik devre donanım tanımlama dili olarak da ifade edilir. 1980'lardan beri sürekli gelişmekte ve IEEE tarafından da bir standart olarak alınmaktadır [17].

VHDL kullanımının başlıca iki amacı vardır;
Sentezleme : FPGA'e yüklenecek kodları oluşturmak için kullanılır.
Simülasyon : FPGA'e yüklenecek kodların simülasyonunu gerçekleştirmek için kullanılır.

3.3. FPGA AKIŞ DİYAGRAMI

Tasarım oluşturulurken izlenilmesi gereken adımlar Şekil 3.3'te verilmiştir.



Şekil 3.3. FPGA akış şeması

3.4. FPGA ÜRETİCİLERİ

Altera ve Xilinx piyasada çok talep gören FPGA üreticileridir. Diğer önemli üreticiler olarak ise Actel, Latice ve Quicklogic firmaları sayılabilir.

3.4.1. Xilinx

FPGA'i ilk olarak üreten ve dünya piyasasındaki en büyük firmadır. Derleyici olarak ISE Design Suite programını sunmaktadır. İletişim, askeri, otomotiv, tüketim gibi alanlarda pek çok uygulaması ve ürünü olan FPGA ve CPLD aygıtlarının üreticisidir. Xilinx, arabirim devreleri (CoolRunner), düşük maliyet gösteren devreler (Spartan), ve yüksek başarı getiren FPGA yongalarının (Virtex) yanı sıra PROM aygıtları da üretmektedir. Ross Freeman, Bernard Vonderschmitt, ve James Barnett tarafından 1984 yılında ABD'de kurulmuştur.

3.4.2. Altera

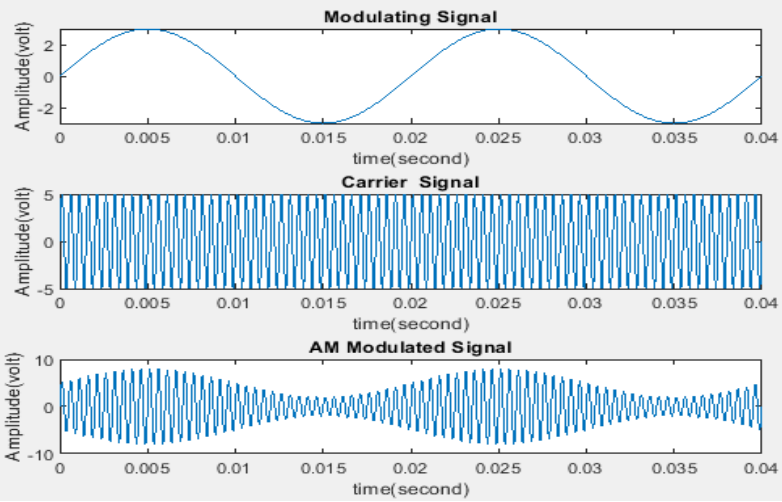
Altera, FPGA'in mucidi Xilinx'in en büyük rakibidir. İlk olarak 1984 yılında piyasaya çıktılar. Altera, 2015 yılında Intel firması tarafından satın alındı. Derleyici olarak Quartus II programını tasarımcılara sundular. Firmanın sunduğu seriler arasında Stratix, Cyclone ve Aria bulunmaktadır.

BÖLÜM 4

GENLİK MODÜLASYONU

Genlik modülasyonu (AM) ilk olarak 1906 yılında araştırmacı Reginald Fessenden tarafından ortaya çıkarılmıştır. Genlik modülasyonunda taşıyıcı olan sinyal sinüs sinyalidir. Verici üzerinde, sinus sinyalinin genliği ise bilgi sinyali ile bağlantılı olarak değişir. Belirtileni gerçekleştiren devrenin ismi modülatördür. Alıcıda üzerinde bu işlemin tam tersi olarak genlik değişiklik bilgi sinyaline dönüşür. Alıcıda üzerinde gerçekleştirilen bu işlem genlik modülasyonu, gerçekleştiren devreye de demodulator denir [18].

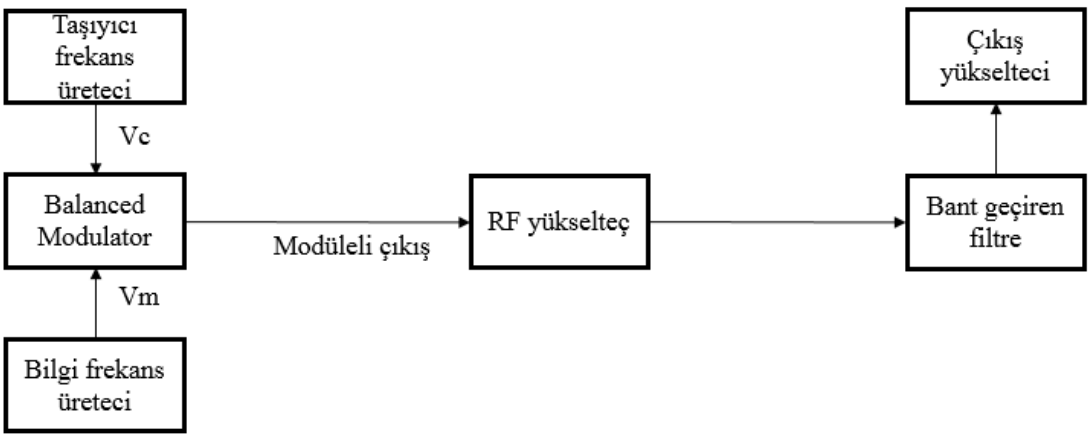
Genlik modülasyonunda bilgi sinyalinin genlik ve frekansına göre taşıyıcı sinyalin genliği değişir. Uzak mesafeye iletilmek istenen alçak frekanslı bilgiler ilk olarak elektrik enerjisine çevrilerek devamında taşıyıcı sinyal üzerinden elektromanyetik dalgalar olarak uzak mesafelere gönderilir. Alçak frekans bilgi sinyalinin pozitif alternansında taşıyıcı genliği de artar. En büyük genlik ise bilgi sinyalinin tepe noktasındadır. Alçak frekans bilgi sinyalinin negative alternansında ise taşıyıcı genliği azalmaktadır. En küçük genlikte bilgi sinyalinin eksi tepe noktasında görülmektedir. Burada module edilen sinyal bilgi sinyali, module eden sinyal ise taşıyıcı sinyalidir. Elde edilen sinyalde modüleli sinyaldir. Şekil 4.1'de tüm sinyaller sırasıyla gösterişmiştir [19].



Şekil 4.1. Bilgi sinyali, taşıyıcı sinyal ve modüleli sinyal

4.1. ÇİFT YAN BANT GENLİK MODÜLASYONU

Genlik modülasyonunda modülasyon işleminde bilgi sinyalindeki tüm frekanslar, üst ve alt yan bantlar olarak elde edilmektedir. Veri iletimi sırasında bu yan bantların ikisininde kullanıldığı genlik modülasyonu, çift yan bant genlik modülasyonu (DSB-AM) olarak adlandırılır. Şekil 4.2'de çift yan bant genlik modüleli verici blok şeması yer almaktadır.



Şekil 4.2. Çift yan bant genlik modüleli verici blok şeması

4.1.1. Bilgi Sinyali

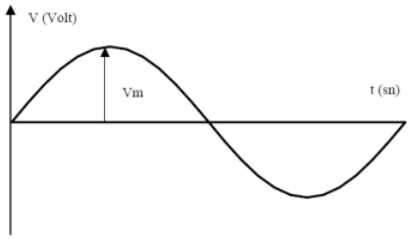
Bilgi (mesaj) sinyali düşük frekanslıdır. Şekil 4.3'te bilgi sinyali gösterilmiştir.

Matematiksel olarak : $\mathbf{v_m} = \mathbf{V_m}\sin 2\pi \mathbf{f_m} t$

$\mathbf{v_m}$ = Mesaj sinyali anlık ac değeri

$\mathbf{V_m}$ = Mesaj sinyali maksimum değeri

$\mathbf{f_m}$ = Mesaj sinyali frekansı



Şekil 4.3. Bilgi sinyali

4.1.2. Taşıyıcı Sinyal

Taşıyıcı sinyali yüksek frekanslı sin/cos sinyalidir. Şekil 4.4'te gösterilmiştir.

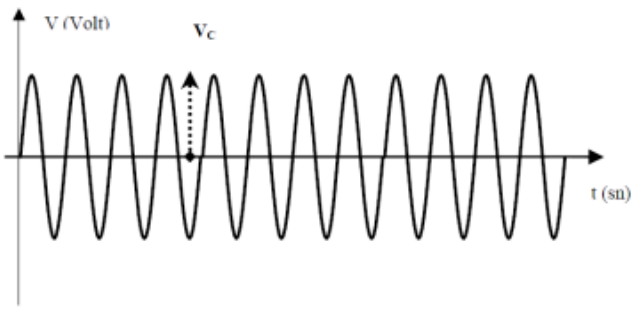
Matematiksel olarak : $\mathbf{v_c} = \mathbf{V_c}\sin 2\pi \mathbf{f_c} t$

$\mathbf{v_c}$ = Taşıyıcı sinyalin anlık ac değeri

$\mathbf{V_c}$ = Taşıyıcı sinyalin maksimum değeri

$\mathbf{f_c}$ = Taşıyıcı sinyalin frekansı

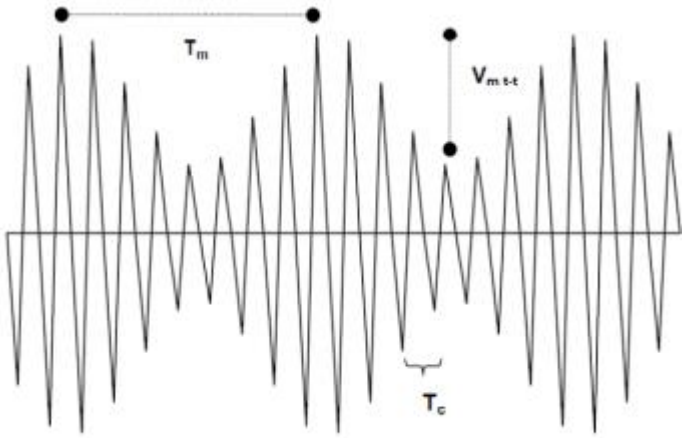
Şekil 4.4'te taşıyıcı sinyali gösterilmiştir.



Şekil 4.4. Taşıyıcı sinyali

4.1.3. Modüleli Sinyal

Bilgi sinyali ile taşıyıcı sinyalin birleştirilmiş halidir.



Şekil 4.5. Modüleli işarette bilgi işaretine ait büyüklükler

4.1.4. Genlik Modülasyonunun Matematiksel İfadesi

$\mathbf{v_m} = \mathbf{V_m}\sin 2\pi \mathbf{f_m} t$

$\mathbf{v_c} = \mathbf{V_c}\sin 2\pi \mathbf{f_c} t$

$\mathbf{V} = (\mathbf{V_c} + \mathbf{V_m}\sin 2\pi \mathbf{f_m} t).\ \sin 2\pi \mathbf{f_c} t$ (Modüleli sinyal)

Çarpma işlemi gerçekleştirildiğinde ;

$\mathbf{V} = \mathbf{V_c}\sin 2\pi \mathbf{f_c} t + \mathbf{V_m}\sin 2\pi \mathbf{f_m} t.\ \sin 2\pi \mathbf{f_c} t$

İki sinus ifadesi çarpımı açılırsa;

$\mathbf{V_m}.\sin a.\mathbf{V_c}.\sin b = -\frac{1}{2}.\ \mathbf{V_m}.\ \mathbf{V_c}\,[\cos(a+b)-\cos(a-b)]$

$\mathbf{V_m}\sin 2\pi \mathbf{f_m} t.\ \sin 2\pi \mathbf{f_c} t = \frac{Vm}{2}.\ \cos 2\pi t(\mathbf{f_c}-\mathbf{f_m}) - \frac{Vm}{2}.\ \cos 2\pi t(\mathbf{f_c}+\mathbf{f_m})$

Sonuç olarak;

$\mathbf{V} = \mathbf{V_c}\sin 2\pi \mathbf{f_c} t + \frac{Vm}{2}.\ \cos 2\pi t(\mathbf{f_c}-\mathbf{f_m}) - \frac{Vm}{2}.\ \cos 2\pi t(\mathbf{f_c}+\mathbf{f_m})$

Yukarıdaki formülde,

V = modüleli sinyal

$\mathbf{V_c}\sin 2\pi \mathbf{f_c} t$ = taşıyıcı sinyal

$\frac{\mathrm{Vm}}{2} . \cos 2\pi t(\mathbf{f_c}-\mathbf{f_m})$ = alt yan bant sinyal

$\frac{\mathrm{Vm}}{2} . \cos 2\pi t(\mathbf{f_c}+\mathbf{f_m})$ = üst yan bant sinyal

$m = \frac{Vm}{Vc}$ (modülasyon indisi)

Buradan da çift yan bant genlik modülasyonu için matematiksel ifade;

$\mathbf{V} = \mathbf{V_c}\sin 2\pi \mathbf{f_c} t + \frac{\mathrm{Vm}}{2} . \cos 2\pi t(\mathbf{f_c}-\mathbf{f_m}) - \frac{\mathrm{Vm}}{2} . \cos 2\pi t(\mathbf{f_c}+\mathbf{f_m})$

V = modüleli sinyal

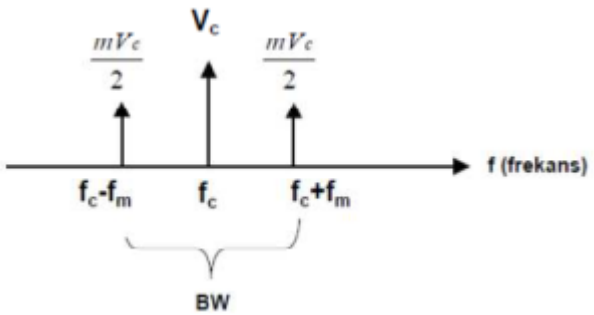
$\mathbf{V_c}\sin 2\pi \mathbf{f_c} t$ = taşıyıcı sinyal

$\frac{\mathrm{mVc}}{2} . \cos 2\pi t(\mathbf{f_c}-\mathbf{f_m})$ = alt yan bant sinyal

$\frac{\mathrm{mVc}}{2} . \cos 2\pi t(\mathbf{f_c}+\mathbf{f_m})$ = üst yan bant sinyal

4.1.5. Bant Genişliği

Sinyalin frekans tayfında işgal ettiği alan bant genişliğidir. Her iki genlik modülasyonu için, RF bant genişliği, bilgi bant genişliğinin iki katıdır. Taşıyıcı frekansın her iki tarafında da bilgi bandına özdeş simetrik olarak birer bant oluşur. Şekil 4.6'da çift yan bant genlik modülasyonu için frekans spektrumu gösterilmiştir.



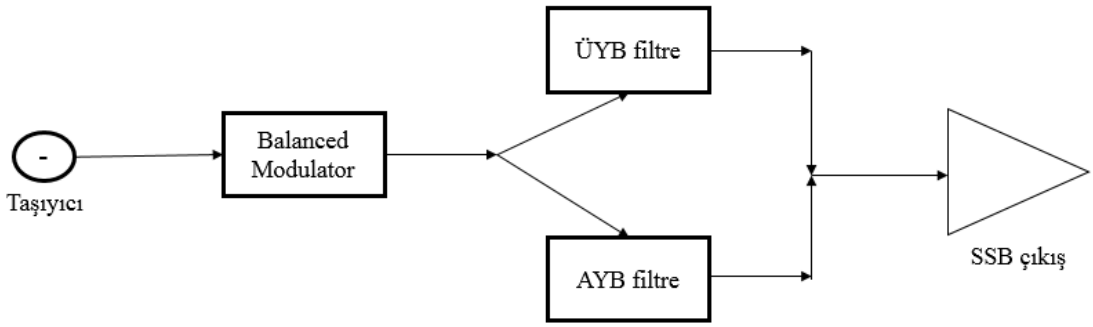
Şekil 4.6. Çift yan bant genlik modülasyonu frekans spektrumu

4.2. TEK YAN BANT GENLİK MODÜLASYONU

Çift yan bant modülasyonunda alt ve üst yan bantların her ikisi de iletilir. Tek yan bant modülasyonunda (SSB-AM) ise sadece bir yan bantın iletildiği modülasyon tipidir. Genelde iki şekilde elde edilmektedir.

4.2.1. Frekans Ayrım Yöntemi

Tek yan bant işaret elde edebilmek için önce bir çift yan bant işareti oluşturulur. Devamında bant geçiren bir filtre yardımıyla ile de istenilen yan bant süzülür ve tek yan bant işareti elde edilir. Bu yöntemin ismi frekans ayrım yöntemidir. Şekil 4.7'de tek yan bant genlik modüleli verici blok şeması gösterilmiştir.



Şekil 4.7. Tek yan bant genlik modüleli verici blok şeması

4.2.2. Faz Kaydırma Yöntemi

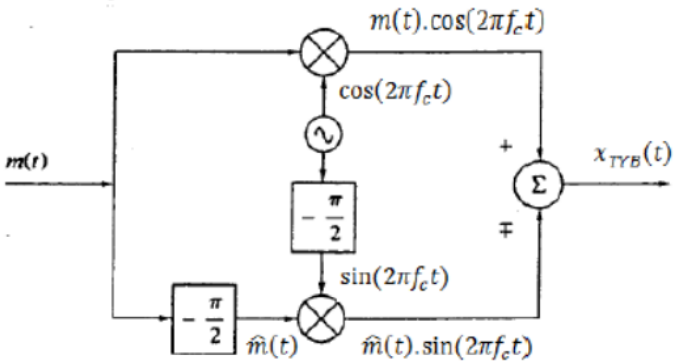
Bu yöntemde ise hem mesaj hem de taşıyıcı iaretinin fazı 90 derece kaydırılır ve çarpılır. Bu işaret ise çift yan bant modülasyon işareti ile toplanarak tek yan bant modülasyonu elde edilir.

Tek yan bant genlik modülasyonunun matmatiksel ifadesi;

$\mathbf{X_{ÜTYB}} = m(t)\cos(2\pi t\mathbf{f_c}t) - \hat{m}(t)\sin(2\pi t\mathbf{f_c}t)$

$\mathbf{X_{ATYB}} = m(t)\cos(2\pi t\mathbf{f_c}t) + \hat{m}(t)\sin(2\pi t\mathbf{f_c}t)$

Şekil 4.8'de bu yöntemin blok şeması gösterilmiştir.

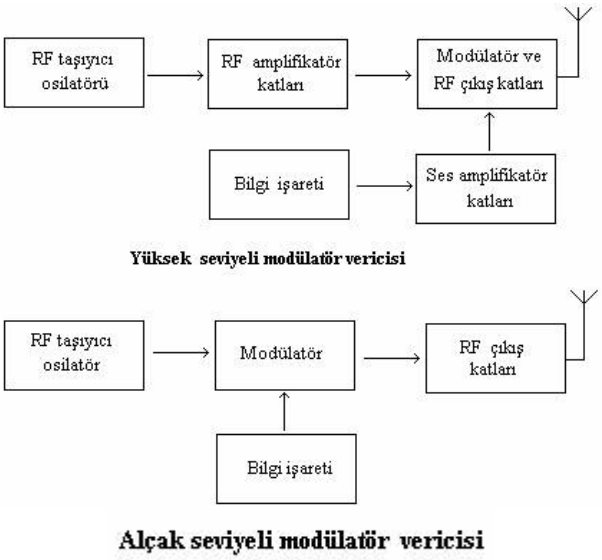


Şekil 4.8. TYB faz kaydırma yöntemi blok şeması

BÖLÜM 5

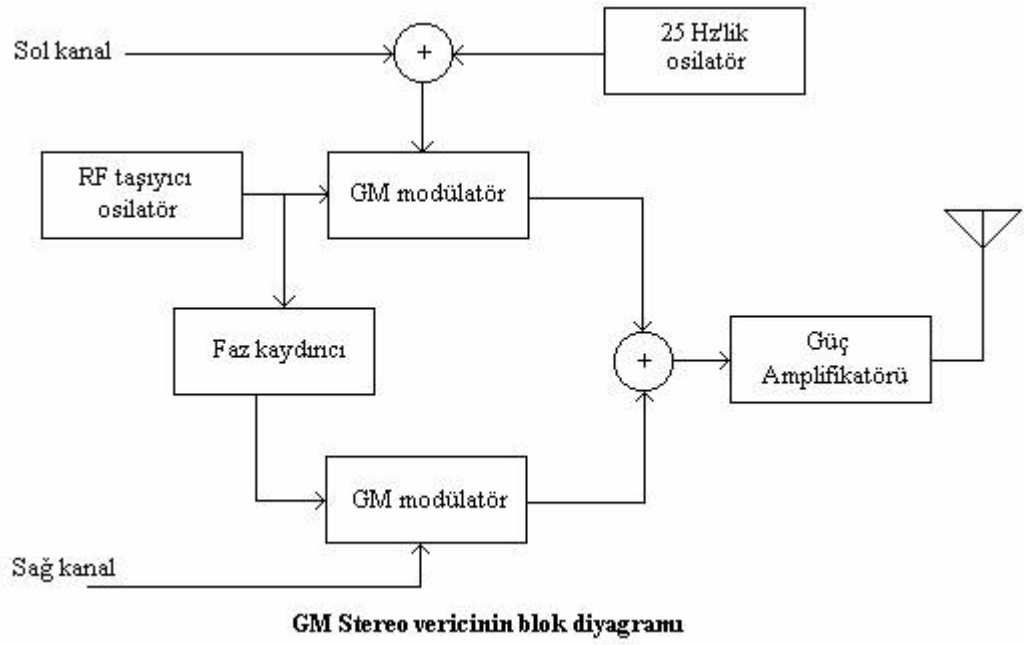
GENLİK MODÜLELİ RADYO VERİCİSİ

Genlik modülasyonu bilindiği üzere, lineer olmayan devre üzerinde taşıyıcı ile mesaj işaretlerinin karıştırılması ile oluşur. Genlik modülasyonunda ki ayrımlardan bir diğeri de bilgi işaretinin mümkün olabilecek en son noktaya uygulanıp uygulanmamasına göre fark edilir. Yüksek seviye bir modülasyon yapıldığında bilgi işareti çoğu zaman son kat çıkışı ile antenin hemen öncesine uygulanır. Bilgi işareti bundan daha önceki başka bir noktaya uygulanmışsa bu modülasyon alçak seviye modülasyondur. Alçak ve yüksek seviyelerin ise gerekli olan çıkış gücüne göre seçimi yapılır. Normal radyo vericilerinde çıkış gücü kW düzeyindedir. Bundan dolayı yüksek seviye modülasyonu kullanılır. Bu devreler üzerinde verimi yüksek tutmak amacı ile transistörler C sınıfı çalıştırılır. C sınıfı amplifikatörler lineer değildir. Dolayısıyla genlik modülasyonu işaretinde az da olsa bir distorsiyon meydana gelir. Alçak seviye modülasyonunda da çıkış güç katları için lineer amplifikatörler kullanılır. Lineer amplifikatörlerde distorsiyon düşüktür fakat verimleri de düşüktür. Dolayısıyla düşük güçlü vericilerde tercih edilir.



Şekil 5.1. Yüksek ve alçak seviyeli modülatör vericileri

Bunun haricinde, bir müzik sesinin iki kanal üzerinden dinlenmesi daha kaliteli ve gerçeğe daha yakındır. GM stereo sistemlerde taşıyıcı sinyali, faz kaydırıcı bir devre üstünden geçirilerek iki farklı taşıyıcı olarak dönüştürülür. Devamında sol kanaldaki sesle taşıyıcının kendisi ve sağ taraftaki sesle fazı kaydırılmış taşıyıcı modüle edilir. Taşıyıcılardan biri aynı zamanda 25 Hz'lik bir sinyal ile modüle edilir. Bu kullanılan sinyal ise stereo alıcılarda yayının stereo olduğunu gösteren gösterge lambasını çalıştırmak amacı iledır. Vericideki bu modüleli sol ve sağ kanallar da bir toplayıcı devrede toplanarak güç amplifikatörüne verilir. Normal alıcılarda ise sağ ve sol kanal tek kanal olarak algılanır. Fakat bu durumda stereo yayınlar için tasarlanmış olan bu alıcılarda faz farkı olan iki taşıyıcı alınarak da iki farklı ses işareti elde edilir.



Şekil 5.2. Genlik modüleli stereo vericinin blok diyagramı

BÖLÜM 6

VHDL – DONANIM TANIMLAMA DİLİ

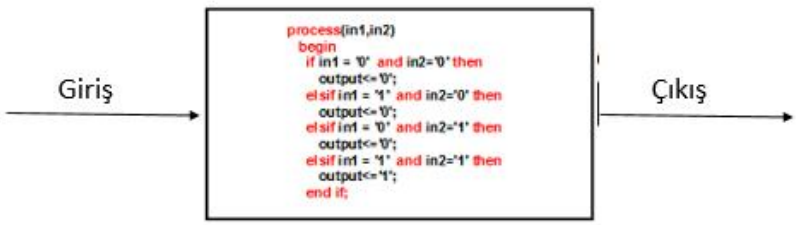
VHDL anlam olarak "Very High-Speed Integrated Circuit Hardware Description Language" den gelir. Yüksek hızlı tümleşik devreler için donanım tanımlama dili olarak çevrilebilir. 1980'lerden bu yana sürekli gelişmektedir ve aynı zamanda IEEE tarafından da kabul görmektedir.

6.1. VHDL TERMİNOLOJİSİ

HDL bir donanım parçasını modellemek için kullanılan programlama dilidir. HDL dili, yazılım kullanarak donanımları yapılandırmak ve donanım davranışlarını belirlemek için imkan sağlar. VHDL, FPGA programlamada en çok kullanılan HDL dilidir.

6.1.1. Davranışsal Modelleme

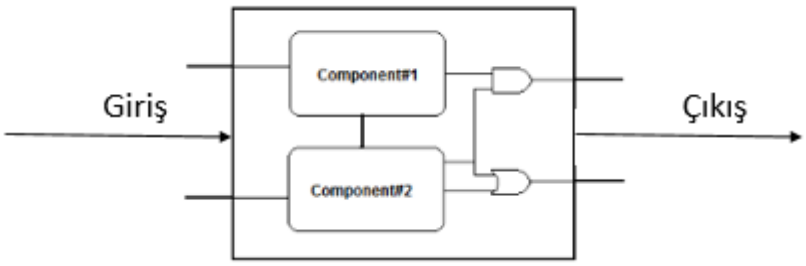
Modeldeki giriş-çıkış tepkileri davranışsal olarak tanımlanır. İç yapısı ile ilgilenilmez. Devrenin işlevi ve fonksiyonu önemlidir. Şekil 6.1 'de davranışsal modelleme gösterilmiştir.



Şekil 6.1. Davranışsal modelleme

6.1.2. Yapısal Modelleme

Bir bileşenin, alt seviyesindeki bileşenler ile arasındaki ilişkileri gösterir. Yapısal modelleme, modelin yapısının tasarımcı tarafından yapılandırılması temeline dayanır. VHDL tasarımlarında çoğu zaman tercih edilen teknik, birbirinden farklı olarak modüllenen alt kısımdaki modüllerin yapısal modellemeyle oluşturulup, yine üst kısımdaki modüllere yapısal modelleme ile bağlanması üzerinedir. Şekil 6.2 'de yapısal modelleme gösterilmiştir.

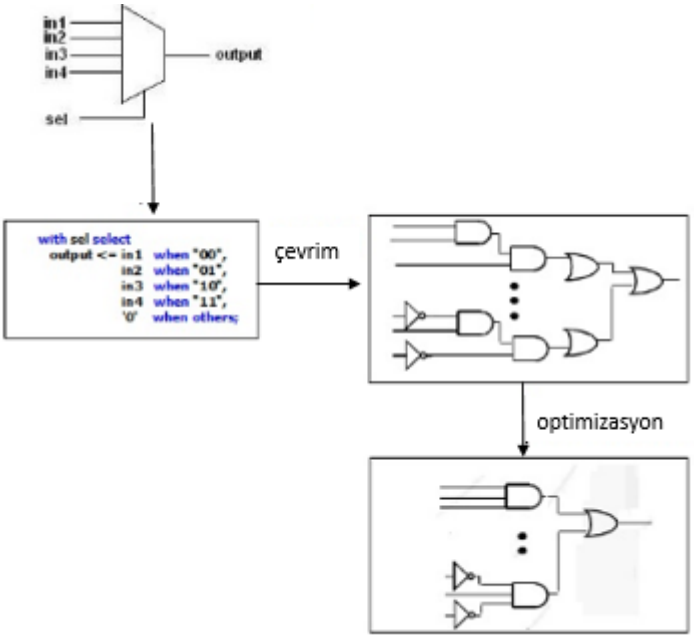


Şekil 6.2. Yapısal modelleme

6.1.3. Register Transfer Level (RTL)

RTL bir soyutlama yöntemidir ve sentezleme amaçlı kullanılır. RTL, oluşturulan bir kodun register cinsinden tasarımının gösterilmesidir. Basitçe ifade edilecek olunursa, VHDL kodumuza karşılık gelen ve mantık kapılarından oluşan devrelerdir.

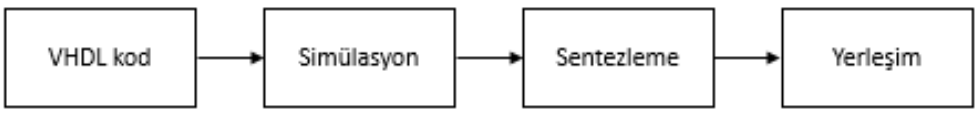
RTL sentezlemede ilk olarak VHDL kodu çevrim işlemi yapılarak sayısal bir devreye çevrilir. Devamında optimizasyon yapılarak VHDL kodunun karşılığı olarak gelen devre optimize edilerek FPGA elemanlarının etkin bir biçimde kullanılması sağlanır. Şekil 6.3'de örnek bir RTL modeli gösterilmiştir. Bu örnekte, 4 girişli tek çıkışlı bir MUX yapılmak isteniyor. İlk olarak VHDL kodu oluşturuluyor. Derleyici tarafından sentezlenen bu kod devamında karşılığı olan sayısal devreye dönüştürülüyor. Son olarak sayısal devreye dönüşen bu kod optimize edilerek RTL akışı tamamlanıyor .



Şekil 6.3. Örnek RTL modelleme

6.2. VHDL TASARIMI

VHDL tasarım olarak, kodlama, simülasyon ve sentezleme olmak üzere üç kısımdan oluşur. Kodlama kısmı, programın VHDL kodunun oluşturulduğu kısımdır. Simülasyon kısmında, VHDL kodunun simlasyonu yapılıp programın doğru olup olmadığı gözlemlenir. Sentezleme kısmında ise, yazılan VHDL kodu donanım diline çevrilip RTL şeması çıkarılır. Daha sonra kod, derleyici tarafından FPGA'e yüklenecek olan konfigürasyon dosyasına dönüştürülür. Şekil 6.4'de VHDL tasarım akışı gösterilmiştir.



Şekil 6.4. VHDL tasarım akışı

6.3. VHDL TASARIM BÖLÜMLERİ

Bir VHDL tasarımı; entity, mimari (architecture), paket (package), bileşen (component) ve işlem (process) olmak üzere 5 bölümden oluşur.

6.3.1. Entity

Tasarımın en temel bloğudur. Tasarım ile tasarımın dış çevresi arasında bulunan arayüzü tanımlamaktadır. Bu bölümde giriş-çıkış portları tanımlanır.

6.3.2. Mimari (Architecture)

Modelin fonksiyonunu tanımlamak için kullanılır. Bir entity birden fazla mimariye sahip olabilir. Bir mimari, davranışsal modelleme, yapısal modelleme ve very akışı olmak üzere üç farklı şekilde kullanılabilir.

6.3.3. Paket (Package)

Paket, entity tarafından kullanılan tanımlamaları bir grup haline getirir ve aynı zamanda farklı tasarımlarda kullanmak üzere de gruplamaya yarar.

6.3.4. Bileşen (Component)

Bileşen yapısal olarak, devre tanımlamasında bir alt devre gibi kullanılan bileşenin adını ve arayüzünü tanımlar.

6.3.5. İşlem (Process)

İşlem bloğu sıralı şekilde gerçekleşecek durumları içerir. Bir mimaride birden fazla işlem bloğu anlık gerçekleştirilir. İşlem blokları aynı anda başlar ve her bir işlem bloğu kendi içinde satır satır sıralı olarak gerçekleştirilir.

6.4. VHDL MODELLEME TEMELLERİ

6.4.1. Sabit (Constant)

İlk başta değeri belirlendikten sonra değiştirilemeyen nesnelerdir. Kodun anlaşılabilirliğini artırmak çoğu zaman kullanılır.

Gösterim şekli; **Constant isim:data tipi:=değer;**

6.4.2. Sinyal

Sinyaller, mimari (architecture) içerisinde yer alan işlemler arasında iletişimi sağlarlar.

Sinyal tanımlama; **Signal isim: tip:=ilk değer;**

Sinyal değer atarken ise <= sembolü atanır. C/C++, yazılım dillerindeki eşittir (=) ifadesi ile aynı görevi görür.

Bütün değerler için;

Reg<=”1100”;

Reg<=x”C”;(hexadecimal)

Tek bit atama için;

Reg (2)<=’1’;

Bit slicing için;

Reg (1 to 2)<=”20”;

6.4.3. VHDL Operatörleri

VHDL aritmetik ve boolen fonksiyonları, sadece standard package olarak tanımlı olan data tiplerinde tanımlıdır.

Aritmetik operatörler (+, -, <, >, <= ,>=) integer tipler de uygulanır. Boolean operatörler (And, Or, Not) ise BIT tipler için uygulanır.

Diğer data tiplerinde aritmetik ve boolen işlemleri için ise IEEE kütüphanesinde bulunan özel fonksiyonlar kullanılır.

IEEE kütüphanesinde özel fonksiyon bulunduran paketler;

std_logic_arith (aritmetik fonksiyonlar)

std_logic_signed (signed aritmetik fonksiyonlar)

std_logic_unsigned (unsigned aritmetik fonksiyonlar)

Yukarıda belirtilen fonksiyonlarda ise operatör ile operatörün ismi aynı olduğundan dolayı fonksiyon ismi tırnak işareti ile belirtilir.

6.4.4. Eşzamanlı Sinyal Atamaları

Eşzamanlı sinyal atamaları için 3 farklı yol vardır:

- Basit sinyal atamaları
- Koşullu sinyal atamaları
- Seçilmiş sinyal atamaları

Basit sinyal atamaları;

Gösterim şekli: **sinyalin ismi <= ifade**

Koşullu sinyal atamaları;

Gösterim şekli: **sinyalin ismi <= ifade ‘when’ koşul ‘else’,**

ifade ‘when’ koşul ‘else’,

ifade;

Seçilmiş sinyal atamaları;

Gösterim şekli: **değer <= ifade ‘when’ seçim,**

ifade ‘when’ seçim,

ifade ‘when’ others;

6.4.5. Sıralı Komutlar

Süreç, fonksiyon ve prosedür işlemlerinde kullanılan komutlardır. Basit sinyal atamaları için de sıralı komutlar kullanılır. Komutlar aşağıdaki gibidir:

- if-then komutu
- case komutu
- döngü komutu
- wait komutu

BÖLÜM 7

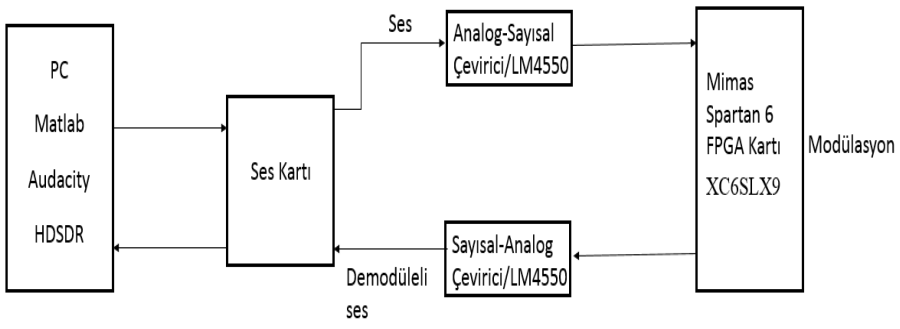
MATERYAL VE METOTLAR

7.1. FPGA TABANLI AM VERİCİ TASARIMI VE UYGULAMASI

FPGA tabanlı AM verici tasarımı ve uygulaması 3 temel aşamadan oluşmaktadır: donanım bileşenlerinin seçilmesi, Matlab'da simülasyon ve tasarım kodlarının geliştirilmesi ve geliştirilen kodun VHDL koduna aktarılması, testlerin sistem üzerinde gerçekleştirilmesi ve test ve simülasyon sonuçlarının karşılaştırılması. Bu aşamalar takip eden kısımlarda ayrıntılı bir şekilde ele alınmıştır.

7.1.1. Donanım Bileşenleri

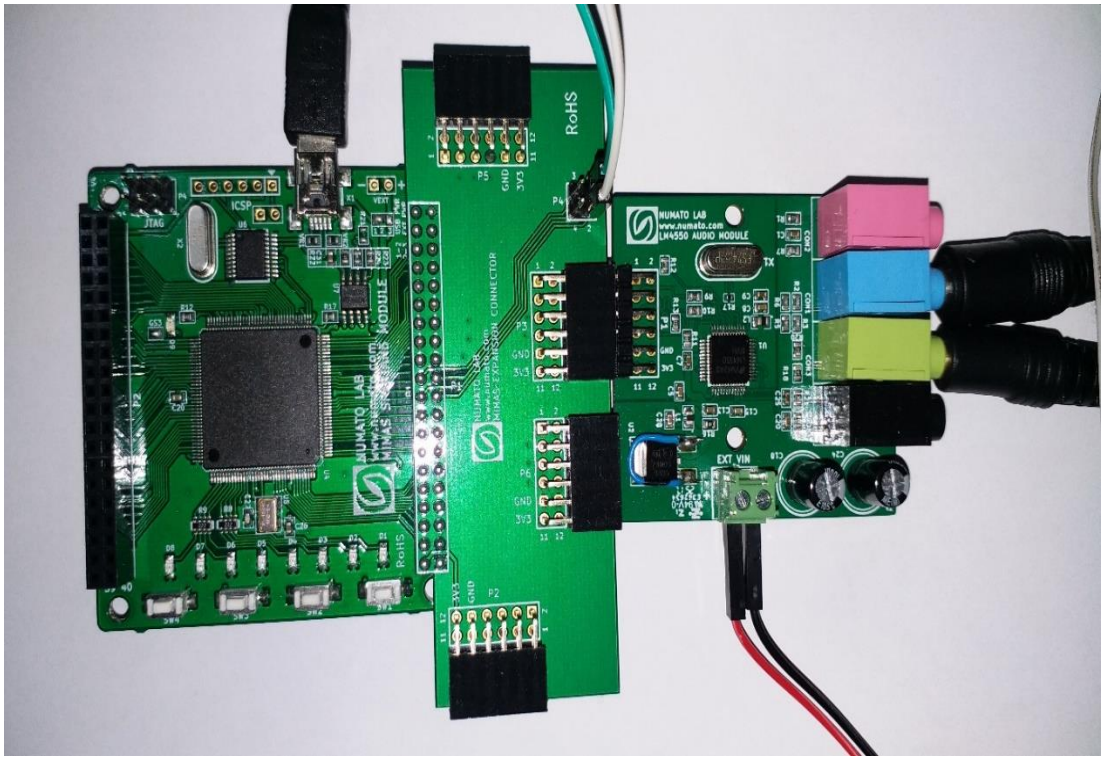
Sistem temel olarak 3 donanım bileşeni üzerine kurulmuştur: Mimas Spartan6 FPGA kartı, LM4550 ses kartı ve IO genişletme kartı. IO genişletme kartı FPGA kart üzerindeki konnektörleri ses kartı üzerindeki PMOD konnektörlere uydurmak için kullanılmaktadır. Bunların dışında sistemin hizmetçi bilgisayar (PC) ile olan bağlantısını sağlamak üzere 2 adet ses uzatma kablosu, programlama için bir adet USB kablosu ve seri iletişimi sağlamak amacıyla bir adet USB'den TTL'ye seri dönüşüm kablosu (PL2303) kullanılmıştır. Sistemin donanımsal kısmının blok şeması şekil 7.1'de görülmektedir.



Şekil 7.1 Sistemin donanımsal yapısının blok şeması

Şekil 7.1'i kısaca açıklayacak olursak: PC'de Audacity programını kullanarak test amacıyla hazırlanmış ses dosyası ses kartından sürekli bir şekilde çalınmaktadır. LM4550 ses kartının ADC kısmında 48KSPs hızında örneklenen bu test sinyali FPGA kartındaki AM verici modülü tarafından modülasyona uğratılır. FPGA içindeki VHDL kodlarıyla tanımlanmış sistemin ayrıntıları ileride verilecektir. Modülasyonun merkez frekansı PC üzerindeki bir seri terminal programı ve seri veri kablosu aracılığı ile gönderilen frekans komutları tarafından belirlenmektedir. Üretilen modüleli işaret daha sonra LM4550 ses kartının DAC kısmı tarafından analog forma dönüştürülür ve PC ses kartının mikrofon girişine gönderilir. Burada alınan modüleli işaret PC yürütülen HDSDR programında hem görüntülenir ve hem de daha sonra analiz edilmek üzere bilgisayara kaydedilir. HDSDR programı aynı zamanda modüleli sinyali demodüle edip ikinci bir ses kartı aracılığı ile dinlemeye de imkân verir.

Bütün bunlara ek olarak FPGA kartı üzerindeki butonlardan komut alan ve bu komutlar vasıtasıyla FPGA içinde sinyalin geçirdiği farklı işlem aşamalarını çıkışa aktarmaya yarayan bir kaynak seçimi modülü de vardır. Kart üzerinde yer alan ledler ise yine bu farklı aşamalar esnasında sinyalin kırpılmaya uğrayıp uğramadığını göstermek için kullanılmıştır. Bir ledin yanmasıyla o ledin ait olduğu blok girişindeki sinyalin sağlıklı çalışma için belirlenen sınırların üstüne çıkıldığı anlaşılmaktadır. Sistemin işletilmesi esnasında PC ses kartının volüme ayarı kullanılarak tüm ledlerin sönük kaldığı en yüksek seviyeye sinyalin getirilmesi gerekmektedir. PC ses kartı volüme ayarı, aynı zamanda modülasyon derinliğini ayarlayan bir ayar olarak görev görmektedir. Ortalama modülasyon derinliğinin tüm test sinyalleri için aynı kalmasını sağlamak amacıyla tüm test kayıtları standart bir normalizasyon seviyesi uygulanarak kaydedilmelidir ve her bir test için PC ses kartı volüme ayarı aynı değerde tutulmalıdır.



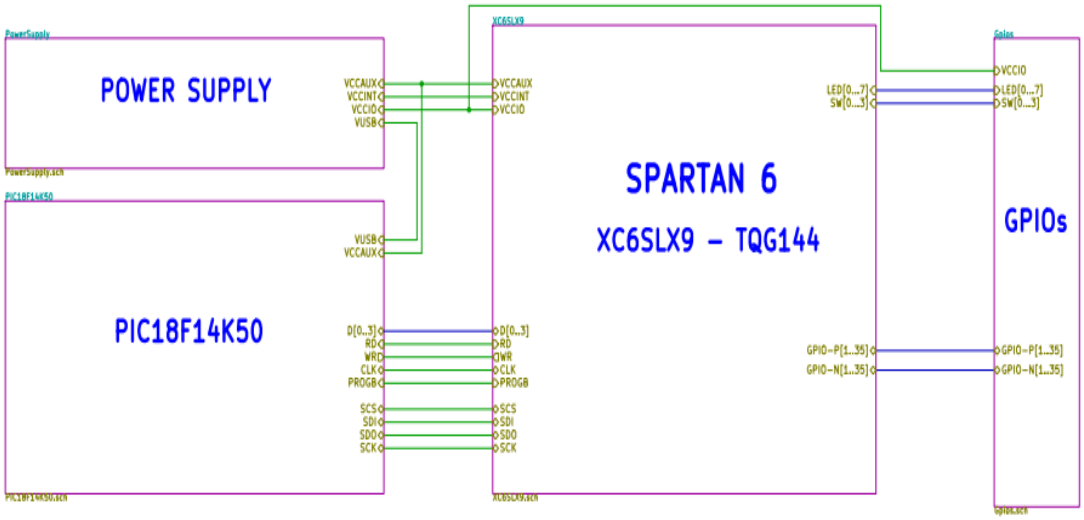
Şekil 7.2. Sistemin genel görünüşü: FPGA kartı (sol), IO genişletme kartı (orta), LM4550 ses kartı (sağ)

Sistemin genel görünümünü veren bir fotoğraf Şekil 7.2'de görülmektedir. Burada 3 donanım bileşeni ve bağlantı kabloları net bir şekilde görülmektedir. Seri veri iletişimini sağlayan kablo aynı zamanda LM4550 ses kartının ihtiyaç duyduğu harici +5V beslemeyi de sağlamaktadır. Bu gerilimin FPGA kartından alınması mümkün değildir, çünkü kart PMOD konnektörler üzerinden sadece 3.3V besleme sağlayabilmektedir. Donanım bileşenleri takip eden kısımlarda ayrıntılı bir şekilde ele alınmıştır.

7.1.1.1. Mimas Spartan 6 FPGA Kartı

Mimas Spartan 6 FPGA kartı kullanımı kolay bir FPGA geliştirme kartıdır. Bu geliştirme kartı; Sistem tasarımını FPGA'larla denemek ve öğrenmek için uygun olacak şekilde tasarlanmıştır. Bu geliştirme kartında maksimum 70 kullanıcı IO'su ile Xilinx XC6SLX9/TQG144 FPGA'ya sahiptir.Yüksek hızlı USB 2.0 arabirimi, yerleşik SPI flaşına hızlı ve kolay yapılandırma indirmesini sağlar. Bit akışını panoya indirmek için herhangi bir programcı veya özel indirici kablosu gerekmez. Şekil 7.3 ve 7.4'te Mimas Spartan 6 FPGA geliştirme kartı ve bağlantı diyagramı gösterilmiştir [20].

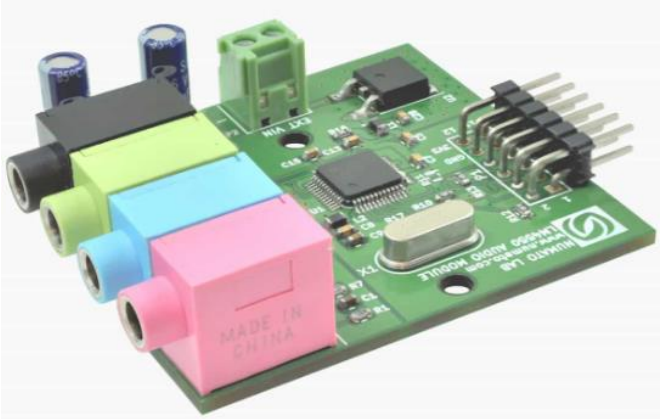
Şekil 7.3. Mimas Spartan 6 FPGA geliştirme kartı



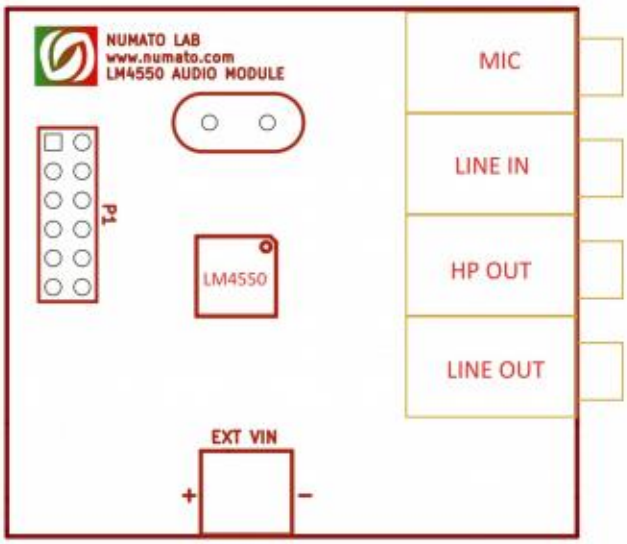
Şekil 7.4. FPGA kart bağlantı şeması

7.1.1.2. LM4550 Ses Kartı

LM4550 genişletme modülü, yüksek kaliteli stereo ses üretip kaydetmeyi sağlayan AC’97 Rev 2.1 uyumlu bir ses codec'i olan LM4550'ye sahiptir. Bu modül, 2×6 pin genişletme konnektörüne sahip FPGA/Mikrodenetleyici kartlarıyla kullanılmak üzere tasarlanmıştır. Manuel kablolama kullanılarak diğer kartlarla ve konektör tipleriyle de kullanılabilir. Şekil 7.5, 7.6 ve 7.7’de LM4550 genişletme modülü ve bağlantı şemaları gösterilmiştir [21].



Şekil 7.5. LM4550 ses kartı



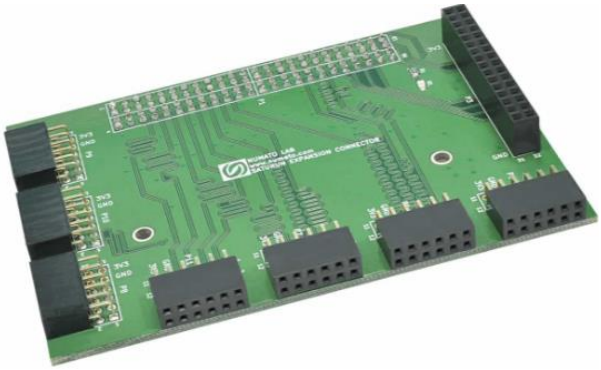
Şekil 7.6. LM4550 bağlantı şeması

Header Pin No.	Pin Details
1	SDO
2	-
3	BIT-CLK
4	SDI
5	SYNC
6	RESET
7	-
8	-
9	GND
10	GND
11	VCC3V3
12	VCC3V3

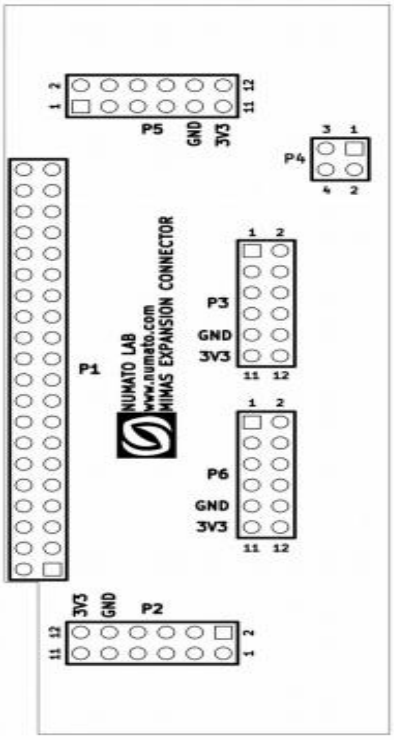
Şekil 7.7. LM4550 pin bağlantı şeması

7.1.1.3. IO Genişletme Kartı

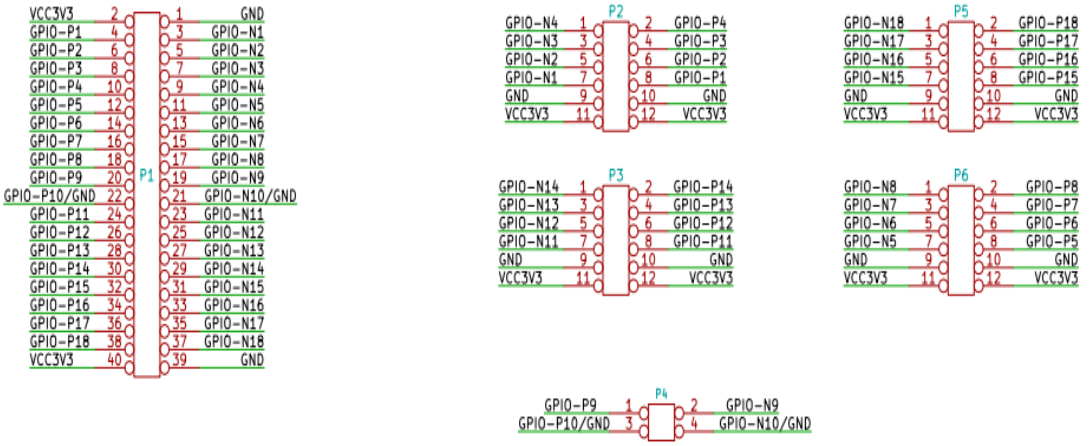
IO çoğaltıcı breakout kartı, Saturn Spartan 6 geliştirme kartı için bir IO koparma çözümüdür. Bu kart, Mimas IO'larının diğer çevresel genişletme modüllerinin kolayca takılmasını kolaylaştıracak daha küçük 2×6 başlıklara ayrılmasını sağlar. Kartta dört adet 2×6 genişletme konektörü bulunur. Şekil 7.8, 7.9 ve 7.10'da IO breakout kartı ve bağlantı şemaları gösterilmiştir [22].



Şekil 7.8. IO genişletme kartı



Şekil 7.9. IO bağlantı diyagramı



Şekil 7.10. IO pin bağlantı şeması

7.1.2. Programlar

Bu çalışmada hizmetçi PC bilgisayarda çeşitli üçüncü parti yazılımlar kullanılmıştır. Takip eden kısımlarda bunlarla alakalı kısa bilgiler verilmiştir.

7.1.2.1. Audacity

Audacity, Windows, Mac OS ve Linux gibi birçok platformda çalışabilen ücretsiz bir sayısal ses düzenleme ve ses kaydetme yazılımıdır. 1999 yılında, Dominic Mazzoni ve Roger Dannenberg tarafından geliştirildi. Bazı özelliklerine değinilecek olursa, ses dosyaları üzerinde kes, yapıştır, birleştir gibi düzenleme işlemleri yapılabilir. Ogg Vorbis, WAV ,MP3 dosya biçimlerini destekler. Bazı ses kartları ve Windows Vista, 7, 8 işletim sistemi kullanarak, bilgisayarda oynatılmakta olan sesleri de kaydedebilir. Yapılan işlemleri sınırsız sayıda geriye ve ileriye alabilir. Bu çalışmada audacity programı ile test amacıyla hazırlanmış ses dosyası ses kartı üzerinden çalınmıştır. Ayrıca FPGA AM verici sisteminin A1 modüle edici sinyali ile elde edilen test ve simülasyon sonuçları yine audacity üzerinden görüntülenmiştir [23].

7.1.2.2. HDSDR

HDSDR, Microsoft Windows için ücretsiz bir yazılım tanımlı radyo (SDR) programıdır. Alberto Di Bene tarafından geliştirilmiştir.
Genel bazı özellikleri ise;

- AM, ECSS, FM, SSB ve CW modülasyonu
- Tx giriş sinyali için I/Q modülasyonlu sinyal çifti Tx çıkışında üretilir.
- Susturucu, gürültü azaltma, gürültü azaltıcı, ayarlanabilir bant geçiren filter ve kenar yumuşatma filtresi uygulanabilmesi
- Kayıt zamanlayıcı ile RF, IF ve AF WAV dosyalarını kaydetip oynatabilme

Bu çalışmada alınan modüleli işaret HDSDR programında görüntülenmiş ve ayrıca daha sonra analiz edilmek üzere bilgisayara kaydedilmiştir. HDSDR programı aynı zamanda modüleli sinyali demodüle edip ikinci bir ses kartı aracılığı ile dinlemeye de imkân vermiştir [24].

7.1.2.3. Xilinx ISE Webpack

Xilinx ISE (Integrated Synthesis Environment), HDL tasarımlarının sentezi ve ve analizi için Xilinx tarafından üretilen, geliştiricinin tasarımlarını derleyebilmesini, zamanlama analizini gerçekleştirmeyi, RTL diyagramlarını incelemeyi, simüle etmeyi sağlayan ücretsiz bir yazılım programıdır.
Xilinx ISE aynı zamanda FPGA ürünleri için bir tasarım ortamıdır. Xilinx ISE öncelikle devre sentezi ve tasarımı için kullanılırken, ayrıca ISIM veya Modelsim mantık simülatörü sistem seviyesi testi için de kullanılır.
Bu çalışmada FPGA AM vericiyi oluşturmak için gerekli tüm kodlamalar VHDL ile ISE Design Suite 14.7 üzerinden yapılmıştır. Tüm bu aşamalar takip eden kısımlarda ayrıntılı bir şekilde ele alınmıştır [25].

7.1.2.4. Matlab

Matlab, MathWorks tarafından geliştirilen tescilli bir çok paradigma programlama dili ve sayısal hesaplama ortamıdır. Matlab, matris manipülasyonlarına, fonksiyonların ve verilerin çizilmesine, algoritmaların uygulanmasına, kullanıcı arayüzlerinin oluşturulmasına ve diğer dillerde yazılmış programlarla arayüz oluşturmaya izin verir. Matlab öncelikle sayısal hesaplama için tasarlanmış olsa da, isteğe bağlı bir araç kutusu, sembolik hesaplama yeteneklerine erişim sağlayan MuPAD sembolik motorunu kullanır. Ek bir paket olan Simulink, dinamik ve gömülü sistemler için grafiksel çok alanlı simülasyon ve model tabanlı tasarım ekler.
Bu çalışmada, FPGA'de modellenecek AM vericinin simülasyonu matlab kodları ile yapılmıştır ve ayrıca kaydedilmiş verici sinyali matlab koduyla offline olarak demodüle edilmiştir. Bu aşamaların detayları takip eden kısımlarda ayrıntılı bir şekilde ele alınmıştır [26].

7.1.3. Matlab Kodları

Öncelikle testlerde kullanılacak ses dosyaları oluşturuldu ve kaydedildi. Bunlar her biri 16-bitte 8KSps hızında örneklenmiş 10 saniye uzunluğunda müzik içeren wav formatında mono ses kayıt dosyalarıdır. Bu dosyalar sonra modülasyon işlemiyle uyumlu olması için 48KSps örnekleme hızına yükseltilmiş, Fc=4KHz'de kesilecek şekilde filtrelenmiş ve normalize edilip test dosyası olarak kaydedilmişlerdir. Bu son işlem Ek B.1'de listesi verilen ve FPGA üzerindeki AM verici tasarımına da temel teşkil eden AM modülasyonu simülasyonu kodu tarafından gerçekleştirilmektedir. Koddaki açıklama satırları ne yapıldığını net göstermektedir. Dolayısıyla fazla bir açıklamaya gerek yoktur. Modüleli sinyal merkez frekansı koddaki bir parametre ile belirlenmektedir ve varsayılan değeri fc=12KHz'dir. Bu değer simülasyon ve testler için kullanılan frekans değeridir. Modülasyon sonucu 2-kanallı stereo formatta kaydedilmektedir. Çünkü HDSDR ile yapılan kayıtlar da 2-kanallı stereo kayıtlardır. HDSDR kayıtları karmaşık formatta olmakla beraber FPGA AM verici modülü gerçek bir sinyali her iki kanalda da tekrarlayarak gönderdiğinden dolayı aslında gerçek

sinyallerdir. Dolayısıyla simülasyon kodunda da benzer şekilde aynı gerçek sinyal her iki kanalda da tekrarlayacak şekilde kayıt yapılmıştır.

Listesi Ek.B.2'de verilen demodülasyon kodu simülasyon veya test sonucunda kaydedilmiş 16-bit 48KSps stereo wav formatlı dosyaları alarak işlemektedir. Yine kodun akışı düz bir mantık izlemekte ve açıklama satırları ne yapıldığını açıkça ortaya koymaktadır. Alınan işlem dosyası stereo formatta olmakla birlikte aslında gerçek bir sinyaldir ve her iki kanalın toplamı alınarak gerçek hale getirilir. Daha sonra normalize edilerek işleme devam edilir. Kayıt boyu farklı durumlar için farklı olacağından işlem esnasında kullanılacak sinyallerin boyu da buna göre uydurulur. Daha sonra demodülasyonu gerçekleştirilecek istasyonu seçecek şekilde karmaşık bir aşağıya kaydırma işlemi yapılır. Bu kaymanın miktarı bir parametreyle seçilebilir ve daha evvel bahsedildiği üzere fc=12KHz varsayılan değerdir. Kaydırmadan sonra baseband işaretin band genişliğine uygun olarak Fc=4KHz' alçak geçiren bir filtre ile sinyal bant genişliği sınırlandırılır. Sonra ikinci bir karmaşık kaydırma işlemi ile sinyal 12KHz ara frekans noktasına doğru yukarı kaydırılır. Bu ikinci kaydırma sabit bir değerdedir ve değiştirilemez. İkinci yukarı kaydırma işlemi demodülasyon için gereklidir. Simülasyon ve testlerde 12KHz lik varsayılan değer kullanılmışsa bu aşağı sonra tekrar yukarı kaydırmalara aslında gerek yoktur. Fakat demodülasyonun temiz bir şekilde gerçekleşmesi için sinyalin filtrelenmesi gerekmekte ve ayrıca merkez frekansı 12KHz den farklı sinyallerle de işlem yapabilmek için demodülasyon kodunun esnek bir şekilde yazılması icap etmektedir.

Demodülasyon için gerekli ön-işlemlerin tamamlanmasının ardından, 12KHz merkezli $\pm$4KHz band genişlikli karmaşık sinyal gerçek hale getirilir. Bunun için basitçe I ve Q kanalları (karmaşık sinyalin gerçek ve hayali kısımları) toplanır. Daha sonra kare alma yöntemi ile asıl demodülasyon gerçekleştirilir. İşlemin tamamlanması için doğrusal olmayan kare alma işlemini takiben oluşan yüksek frekanslı bileşenleri atan bir alçak geçiren filtreleme işlemi fc=4KHz'de gerçekleştirilir. Nihai sinyal bilgisayara wav formatlı olarak kaydedilir. Ara işlem sonuçları da gözlem amaçlı olarak bilgisayara kaydedilmektedir.

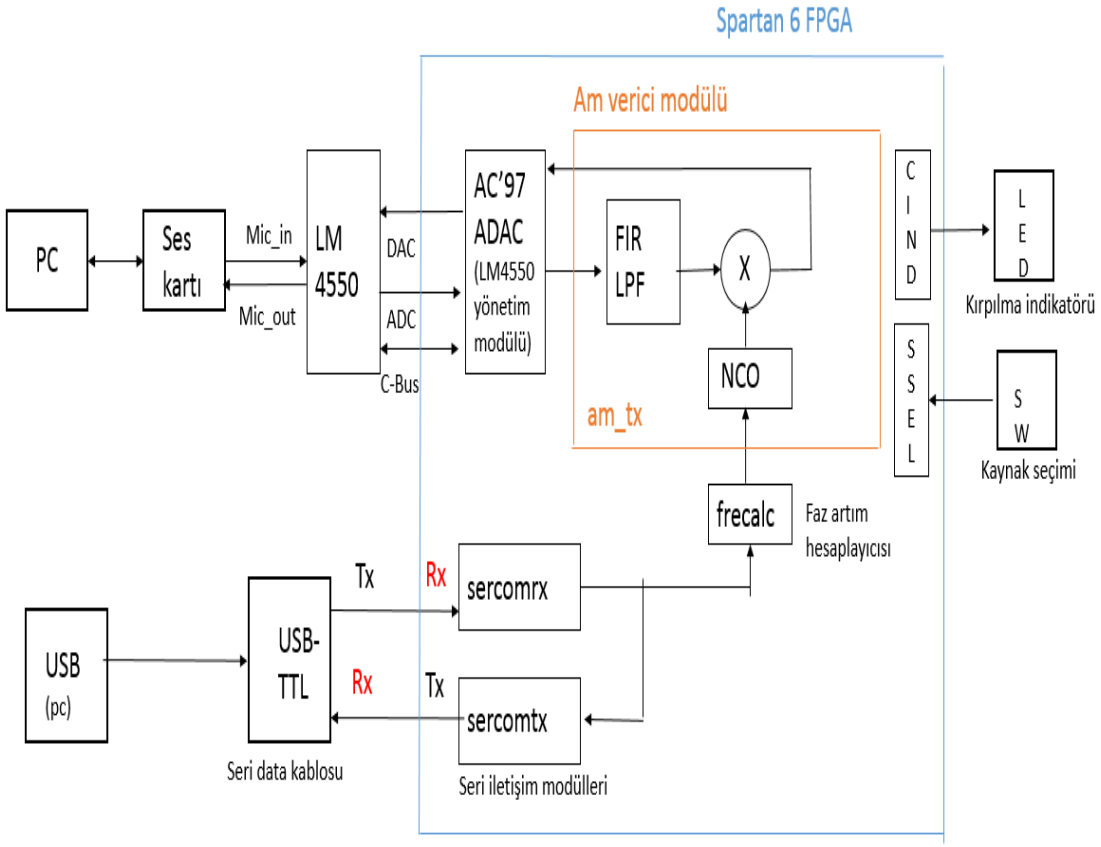
Demodülasyon kodundan elde edilen ve bilgisayara kaydedilen nihai sonuç dosyası ayrıca bir başka analiz kodunda orijinal modüle edici sinyalle karşılaştırılmaktadır. Bu analiz kodunun listesi Ek. B.3'te verilmiştir. Karşılaştırma işlemini kolayca gerçekleştirebilmek ve senkronizasyonu sağlamak için sonuç dosyası ile en başta kullanılan orijinal test dosyası tek bir çift kanallı wav dosyasında birleştirilir. Bu ön-işlem Audacity programı kullanılarak elle gerçekleştirilmektedir. Bu şekildeki ön-işlem dosyalarında sol kanal demodülasyonla elde dilmiş sinyale ayrılmışken sağ kanala orijinal test kaydı konulur. Karşılaştırma işlemi sonucunda bir fark sinyali elde edilir ve ayrıca wav formatında bilgisayara kaydedilir. Bu fark sinyalinin 10 saniyelik bir çerçevede rms değeri hesaplanır ve sunulur. Ayrıca orijinal sinyalin aynı çerçeve için rms değeri hesaplanarak rms fark değerine oranı dB ölçeğinde verilir. Bu bulunan değer o karşılaştırma işlemi için bize sinyal-gürültü oranını (SNR) verir. SNR ne kadar yüksek ise modülasyon o derecede kusursuz gerçekleştirilmiş demektir. Bu sonuçların dökümü ve bir tartışması sonuçlar ve tartışma bölümünde ayrıntılı bir şekilde irdelenmektedir.

Son olarak FPGA AM vericinin AM_TX modülünde ihtiyaç duyulan FIR alçak geçiren filtrenin katsayılarını hesaplayan bir tasarım kodu Ek. B.4'te verilmiştir. Tasarlanacak filtrenin parametreleri (tipi, katsayı sayısı, kesim frekansı, örnekleme frekansı, katsayı bit çözünürlüğü) girildikten sonra tasarlanan filtrenin Bode diyagramı ekrana getirilerek performansının değerlendirilmesi sağlanmaktadır. Elde edilen katsayılar bilgisayarda bir dosyaya kaydedilmektedir. Katsayılar bu dosyadan alınarak, Xilinx ISE ortamındaki FIR filtre IP sihirbazında kullanılmak üzere bir coe dosyasına kolaylıkla aktarılabilir. Yüksek performanslı olması ve tasarımı kolaylaştırması ve hızlandırması bakımından AM_TX modülündeki filtre, NCO ve çarpıcılar ilgili IP sihirbazları kullanılarak tasarlanmış ve uygulanmıştır.

7.1.4. VHDL Kodu ve Sistemin Blok Şeması

Sistemin VHDL kodlarının listesi Ek.C'de verilmiştir. MATLAB kodlarında olduğu açıklama satırları neyin nasıl yapıldığını açıkça göstermektedir ve fazlaca açıklamaya gerek yoktur. Şekil 7.11'deki blok diyagram FPGA AM verici sisteminin iç yapısını göstermektedir. Üst modül amtx (Ek.C.1) diğer bütün alt modüller için bir santral

görevi görmektedir. Alt modüller arasındaki bağlantılar ve çipin dış dünyayla olan bağlantısı bu modülde sağlanmaktadır. Bu modülde ayrıca ihtiyaç duyulan tüm saat ve reset sinyalleri de üretilmektedir.



Şekil 7.11. FPGA AM verici sistemi blok şeması

AM vericiye ait tüm temel fonksiyonlar am_tx (ek.c.2) alt modülünde toplanmıştır. Bu modül giriş ve çıkışları itibariyle I/Q şeklinde çift kanal giden karmaşık sinyallerle çalışıyor gibi gözükse de aslında iç yapı olarak tamamen tek kanal üzerinden giden ve sinyalleri gerçek formda işleyip sunan bir düzene sahiptir. Buradaki maksat ileride karmaşık sinyallerle çalışmaya müsait bir altyapı hazırlamak veya iki kanaldan birisini seçmek suretiyle yayını değiştirebilmektir. Bu şekil aynı anda iki farklı yayını iki farklı istasyon üzerinden yayınlamaya da imkan vermektedir. Girişin çift kanallı olması iki ayrı kanalın karışımı ile yayın yapma gibi alternatifler de sunmaktadır. Burada hali hazırda giriş ve çıkışın çift kanal üzerinden gidip geliyor olmasının getireceği imkanlar neredeyse sınırsızdır. Modülün iç yapısında basit bir düzenleme yaparak sistemin çalışma şekli tamamen değiştirilebilmektedir. Bu da SDR sistemlerinin en önemli avantajlarından birisidir.

Modülün dahili yapısına bir göz atacak olursak: girişteki modüle edici sinyal alınarak kesim frekansı fc=4Khz olan bir alçak geçiren filtre ile sınırlandırılmaktadır. Bu işlem yayının ancak müsaade edilen kanal bandgenişliği içinde kaldığından emin olmak için gereklidir. Aksi takdirde komşu istasyonların yayınını etkileyecek şekilde bir sarkma meydana gelebilir. Filtre FIR tipinde olup tasarımı daha evvel bahsedilen filtre tasarım kodu ile gerçekleştirilmektedir. Tasarım sonucunda elde edilen katsayılar FIR filtre IP sihirbazında kullanılır.

Filtreleme işleminden sonra sinyale belli bir ofset değer verdirilir. Bu modülasyonun taşıyıcılı genlik modülasyonu olması için gereklidir. Sinyale yeterli bir seviyede ofset verilmezse modülasyon taşıyıcısı bastırılmış genlik modülasyonu olur. AM radyo yayınları ucuz zarf dedektörleri içeren radyo alıcıları ile kolaylıkla dinlenebilmesi için taşıyıcılı tipte olmak zorundadırlar. Bu nedenle bu ofset eklemesi gereklidir. Ofset basitçe bir toplayıcı tarafından sinyale eklenir.

Filtrelenen ve ofset verdirilen sinyal daha sonra bir çarpıcı ile frekans kaymasına uğratılır. Frekans kaymasının miktarı diğer girişten uygulanan NCO (nümerik kontrollü osilatör) tarafından belirlenir. Filtre gibi NCO ve çarpıcı da ilgili IP sihirbazları kullanılarak gerçekleştirilmektedir. NCO'nun frekans kontrolü frecalc (Ek.c.3) faz artım hesaplayıcısı üzerinden sağlanmaktadır. Frecalc modülü faz artımını sercomrx (ek.c.4) modülünden gelen frekans bilgisi ile hesaplar. Sercomrx modülü de hizmetçi PC üzerindeki bir terminal programından gönderilen frekans komutlarını aradaki seri bağlantı üzerinden temin eder. Alınan komutlar sercomtx (Ek.c.5) modülü tarafından yankılanır. Böylece bilgisayar başındaki operatör komutun doğru bir şekilde alındığını görebilir. Frekans kontrolünde kullanılan komutların formatı şu şekildedir: f +<nnnnn>. Burada f, komutun frekans değiştirme komutu olduğunu göstermektedir. Sistemde başka komut yoktur, fakat ileride başka kontrol komutlarına ihtiyaç olması durumunda diğer komutlardan ayırt etmek için gereklidir. Yine sadece pozitif frekanslar kullanılmış olduğu halde, ileride negatif frekans tanımlama gerekebileceği düşünülerek artı işaretinin kullanılma zorunluluğu getirilmiştir. nnnnn ise 5 haneli frekansı gösteren rakam olup 00000-23999 arasındaki değerler geçerlidir. Frekans çözünürlüğü bu durumda 1Hz olmaktadır.

am_tx modülünde her bir sinyal bloğu girişine iliştirilmiş kırpılma indikatörleri vardır. Bu detektörler ait oldukları bloğun girişindeki sinyalin belli limitler dahilinde olup olmadığını gözlemlerler. Limit aşımı gerçekleştiğinde modülasyonun sağlıklı biçimde gerçekleştirilmesi mümkün olmaz. Her bir indikatör FPGA kart üzerinde yerleşik bulunan LEDlere ikaz verir. Bu LEDlerden herhangi birinin yanması limitin aşıldığı dolayısıyla modülasyonun hatalı gerçekleşme ihtimali olduğunu gösterir.

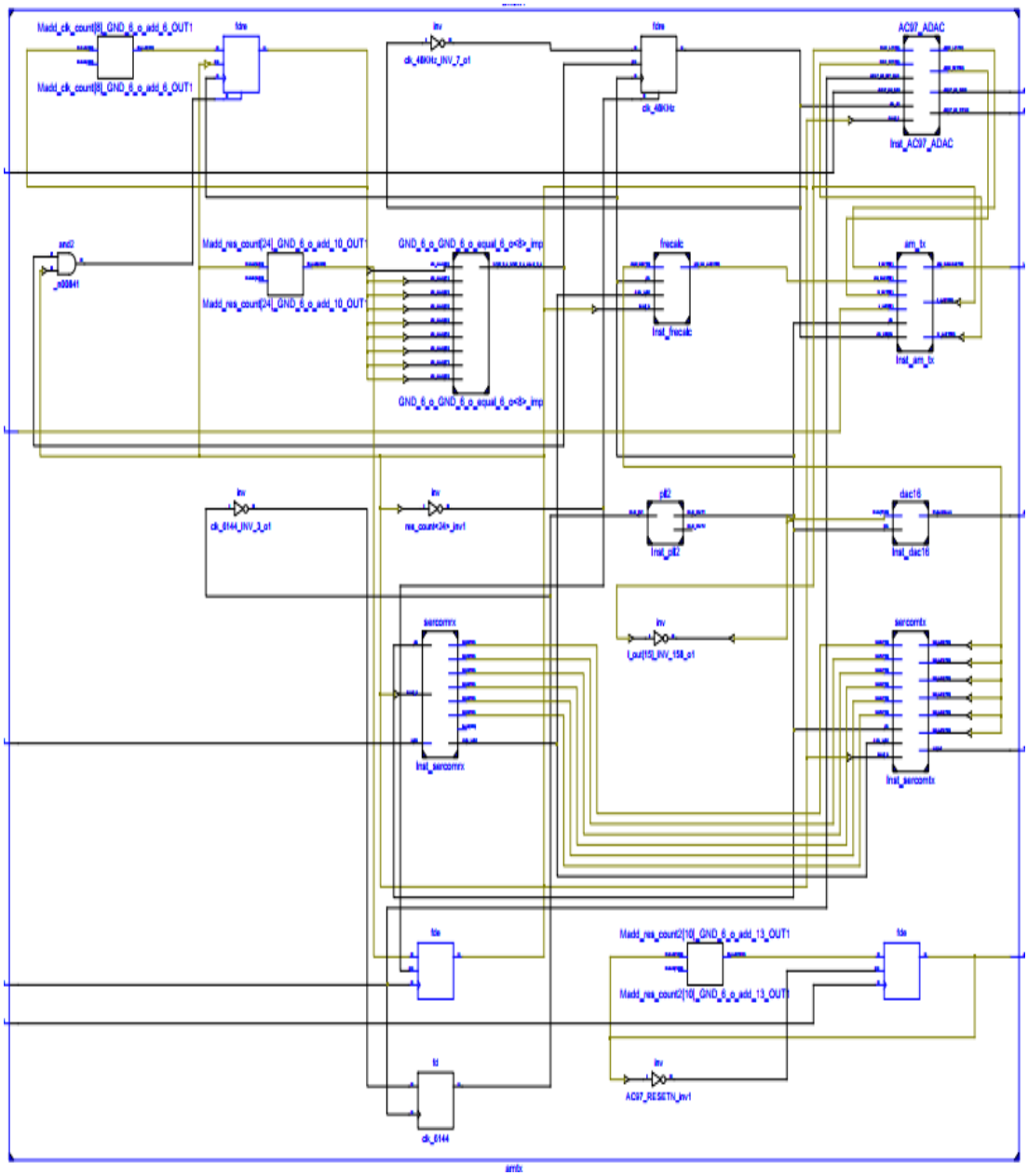
Yine FPGA kart üzerindeki yerleşik butonlardan emir alan bir kaynak seçimi prosesi (s_sel_proc) mevcuttur. Bu prosesin görevi butonlardan gelen talimata bağlı olarak am_tx modülü içindeki bloklardan birisini seçerek çıkışa vermektir. Böylece sinyalin modülasyon yolunda geçirdiği farklı evreler çıkışa yönlendirilmek suretiyle gözlenebilir. Bu özellik sistemin eğitim amaçlı kullanımı için özellikle düşünülmüştür. Prosesin seçimlikleri, modül girişi (baypas modu), filtre çıkışı, ofset çıkışı, nco çıkışı, çarpıcı çıkışı (varsayılan) şeklindedir.

Giriş ve çıkıştaki sinyallerin sayısallaştırılması ve analog forma dönüştürülmesi LM4550 ses kartı üzerindeki ADC ve DAC'lar tarafından gerçekleştirilir. LM4550 ses kartının kontrolü amacıyla FPGA'de bir AC97_ADAC (ek.c.6) kontrol modülü geliştirilmiştir. Bu modül FPGA çipinin resetten çıkmasıyla beraber LM4550 çipine gerekli konfigürasyon komutlarını gönderir ve bu konfigürasyon işlemi tamamlanır tamamlanmaz sayısal verileri almaya ve göndermeye başlar. AC97_ADAC yönetim modülünün kodlaması LM4550'ye ait veri sayfalarının incelenmesi neticesinde yapılmıştır (ek.a.2).

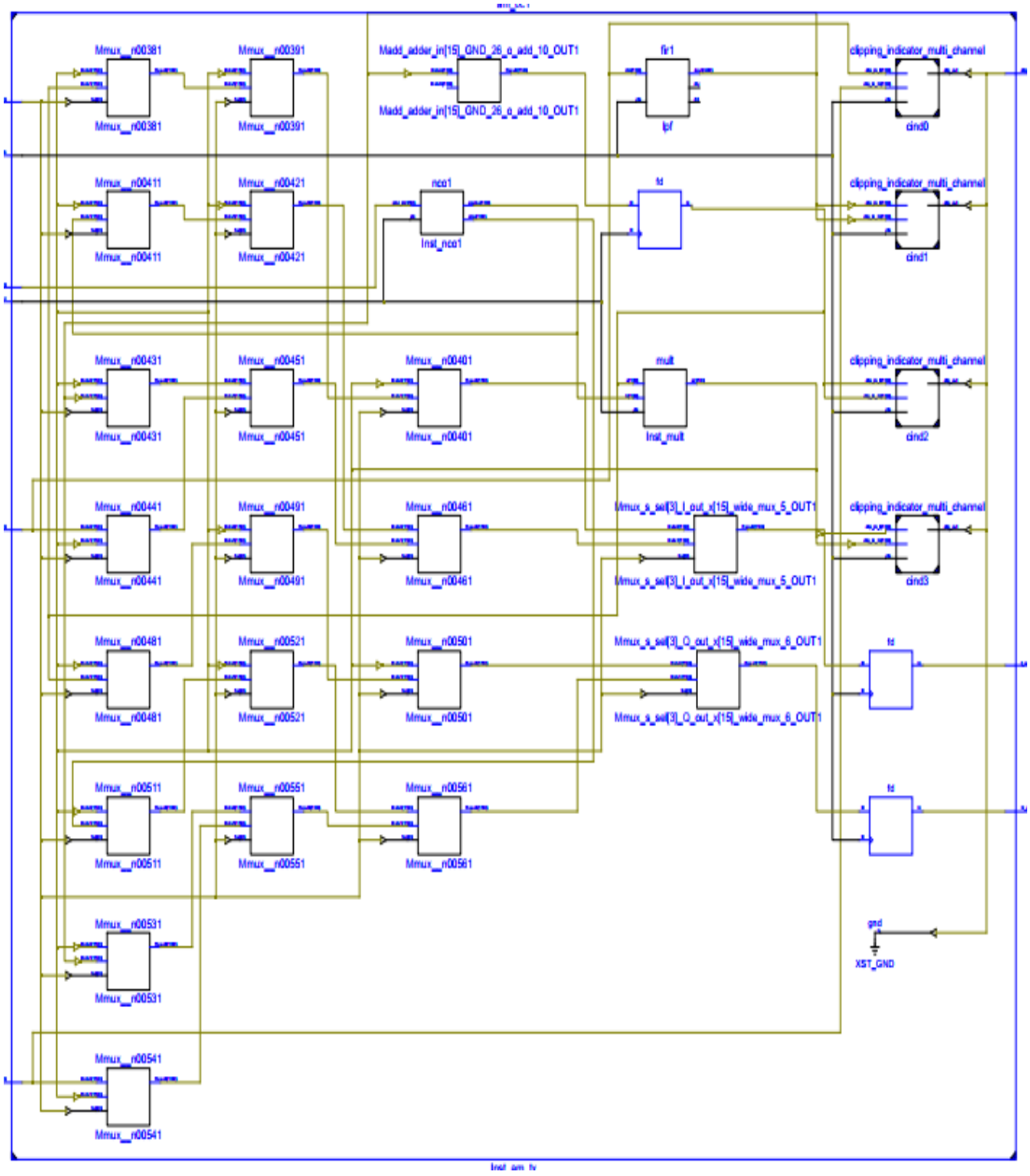
Tasarım neticesinde elde edilen sentez raporuna göre: 1430 adet lojik dilimden 430 adedi (%30), 32 adet RAMB16WER ram bloğundan 15 adedi (%46), 16 adet DSP48A1 dsp bloğundan 7 adedi (%43) kullanılmıştır. Tasarımda kullanılan işlemlerin çoğunun sinyal işlemeye yönelik aritmetik hesaplamalar olduğu göz önünde bulundurularak ram ve dsp kullanımının yüksek çıkması normal karşılanmalıdır. Öte yandan FPGA üzerinde ikinci bir vericiye daha yetecek kadar kaynak olduğu görülmektedir. Giriş ve çıkışların çift kanal üzerinden olduğu da göz önünde bulundurulursa bunun kolaylıkla mümkün olabileceği çok açıktır.

7.1.5. Sistemin RTL Diyagramları

RTL diyagramları FPGA tasarımının iç yapısını göstermek açısından faydalı araçlardır. Tasarımda neyin nasıl yapıldığını hızlıca görme imkanı sağlarlar. Tasarımın debug ve test aşamasında problemlerin takibinde kullanılırlar. Bu kısımda FPGA AM verici tasarımının önemli kısımlarının RTL diyagramlarına yer verilecektir. Bunlar üst modül AMTX ve am_tx alt modülüdür.



Şekil 7.12. Üst modül AMTX'in RTL diyagramı



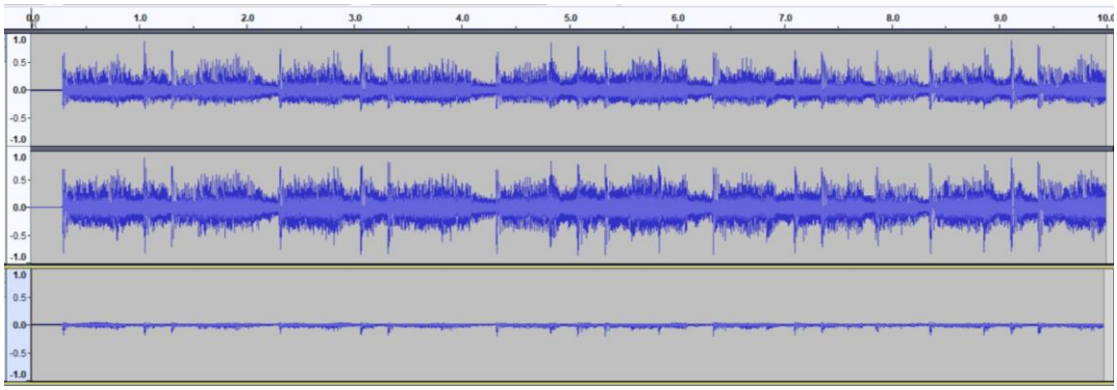
Şekil 7.13. Alt modül am_tx'in RTL diyagramı

BÖLÜM 8

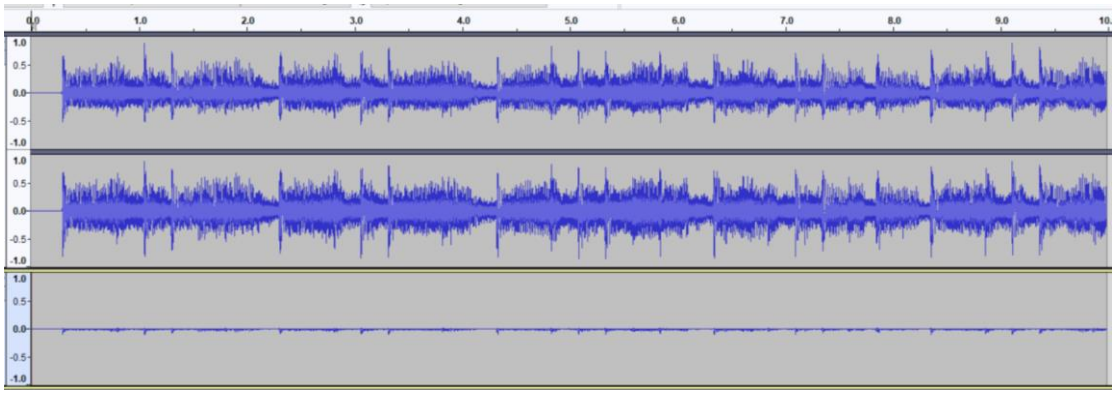
SONUÇLAR VE TARTIŞMA

Sistem, wav formatında 8KSps hızında örneklenmiş iki adet 10saniye uzunluğunda müzik içeren ses kaydı dosyası A1 ve A2 kullanılarak test ve simüle edilmiştir. Test ve simülasyonda kullanılan kayıtlar modülasyon ve demodülasyonun gerçekleştirildiği 48KSps örnekleme hızına uydurmak maksadıyla örnekleme hızları uygun bir interpolasyon işlemi ile 6 kat artırılmıştır. Bu şekilde elde edilen kayıtların frekans içeriğinin 4KHz'de sınırlandırıldığından emin olmak için keskin bir alçak geçiren filtreden geçirilmiştir. Her bir örnek kayıt farklı bir müzik içermektedir. Müzikler seçilirken test ve simülasyonun adil ve geçekçi şartlarda gerçekleşebilmesi için spektral içeriğinin 4KHz'lik bandgenişliği içerisinde normal bir dağılım göstermesine özen gösterilmiştir. Bu şekilde yapılan deney sonuçları aynı band genişliğine sahip beyaz gürültü ile yapılan testlere benzer olacaktır. Bu test sinyallerinin her birisi için bir kez EkA.1'de listesi verilen modülasyon koduyla bir kez de sistem çalıştırılarak (test kaydı Audacity'de sürekli çalma modunda çalınarak, HDSDR ile izlenerek ve kaydedilerek) modüleli sinyal kayıtları alınmıştır. Bu kayıtlar Ek.A.2'de listesi verilen demodülasyon koduyla demodüle edilip sonuçlar ayrı ayrı kaydedilmiştir. Bu şekilde elde edilen kayıtlar Audacity'de orijinal kayıtla senkronize hale gelecek ve 10 saniyelik kayıt uzunluğu verecek şekilde baştan ve sondan fazlalıkları kırpılarak kaydedilmiştir. Bu son kayıtlar üstte (sol kanal) test kaydı, altta (sağ kanal) orijinal kaydolacak şekilde çift kanallı (stereo) formatta 48KSps örnekleme hızında kaydedilmiştir. Bu yaklaşımın amacı iki dalga şekli arasındaki senkronizasyonu kaybetmeden tüm dalga şekillerini tek bir dosyada toplayarak karşılaştırma ve analiz işlemlerini kolaylaştırmaktır. Ayrıca karşılaştırmanın sağlıklı yapılabilmesi için her iki dalga şekli de aynı standart normalizasyon işlemine tabi tutulmuşlardır. Demodülasyon kodunda kare alma yöntemi kullanıldığından sonuçlar ciddi oranda DC ofset içermektedir. Karşılaştırmayı zorlaştıracak fakat yokluğunun herhangibir sorun oluşturmayacak olan bu DC ofset normalizasyon işlemi esnasında Audacity'de atılmıştır. Daha sonra bu kayıtlar üzerinde Ek.A.3'de listesi verilen analiz koduyla

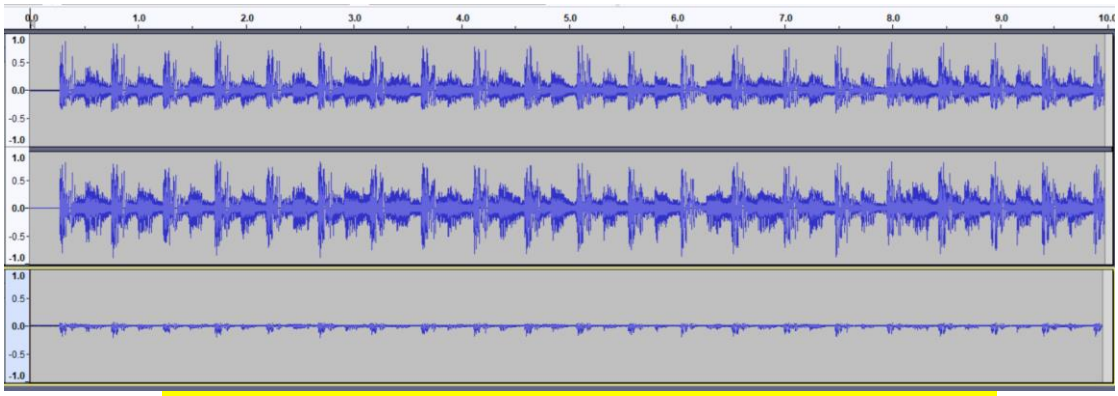
değerlendirme yapılmıştır. Değerlendirme işlemi sonucunda 3 veri elde edilmektedir: iki dalga arasındaki farkı gösteren wav formatındaki fark dalga şekli dosyası, iki dalga arasındaki farkı hata kabul ederek bu dalga şeklinin 10saniyelik bir çerçevede elde edilen rms değeri, benzer şekilde orijinal dalgaşeklinden elde edilen rms değeri kullanarak, rms hata değerine oranının dB cinsinden hesaplandığı sinyal/gürültü oranı (SNR) değeri. Her bir kayıt için: test veya simülasyon dalgaşekli üstte, orijinal dalgaşekli ortada ve analzi işlemi neticesinde elde edilen fark dalgaşekli altta olacak biçimde Şekil 8.1-4 arası grafiklerde bu sinyaller gösterilmiştir. Ayrıca analizden elde edilen diğer iki veri de Tablo 8.1'de kaydedilmiştir.



Şekil 8.1 FPGA AM verici sisteminin A1 modüle edici sinyaliyle yapılan test sonuçları: üstte A1 test sinyali, ortada modülasyonu takiben demodülasyonla elde edilen dalga şekli, altta iki sinyalin farkı



Şekil 8.2 FPGA AM verici sisteminin A1 modüle edici sinyaliyle yapılan simülasyon sonuçları: üstte A1 test sinyali, ortada modülasyonu takiben demodülasyonla elde edilen dalga şekli, altta iki sinyalin farkı



Şekil 8.3 FPGA AM verici sisteminin A2 modüle edici sinyaliyle yapılan test sonuçları: üstte A2 test sinyali, ortada modülasyonu takiben demodülasyonla elde edilen dalga şekli, altta iki sinyalin farkı

Şekil 8.4 FPGA AM verici sisteminin A2 modüle edici sinyaliyle yapılan simülasyon sonuçları: üstte A2 test sinyali, ortada modülasyonu takiben demodülasyonla elde edilen dalgaşekli, altta iki sinyalin farkı

Çizelge 8.1 Deney ve simülasyon sonuçları

Test Sinyali	**hata (rms)**	**SNR (dB)**
A1_test	15.23024e-03	20.94
A1_sim	6.916849e-03	27.80
A2_test	19.75155e-03	18.65
A2_sim	8.887607e-03	25.58

Şekil 8.1-4 arası şekillerin ve Tablo 8.1'deki verilerin incelenmesi neticesinde, aynı test sinyali için simülasyon sonuçlarının gerçek hayat test sonuçlarından biraz daha iyi olduğu görülmektedir. Bu durum, gerçek hayattaki testlerde sinyalin harici gürültüye maruz kaldığı göz önünde bulundurulursa, normal kabul edilebilir. Bu gürültü sinyalin PC ve LM4550 ses kartı arasındaki analog bağlantı kabloları üzerindeki seyahati

esnasında sinyale sızabileceği gibi alıcı ve verici terminaller arasındaki ufak örnekleme hızı farklarından da kaynaklanabilir. Bir diğer hatadaki farkı artıran husus, Audacity'de yapılan analiz öncesi işlemlerde (senkronizasyon, normalizasyon ve filtreleme işlemleri) yapılan kusurlardır.

Tablo 8.1'de SNR değerleri üzerinden bir kıyas yapacak olursak; yüksek SNR değeri düşük hata değeri dolayısıyla daha iyi sonuç demektir. En iyi sonuç A1 test sinyaliyle yapılan simülasyonlarda elde edilmiştir. A2 test sinyaliyle yapılan simülasyon A1 ile yapılana kıyasla daha düşük SNR vermiştir. Benzer şekilde A1 ile yapılan test sonucu A2'ye kıyasla daha iyidir. Bu durum A2 test sinyalinin yapısı dolayısıyla ortalama modülasyon derinliğinin daha düşük kalmasından kaynaklanmaktadır. A2 yüksek tempolu müzik içermekte ve ani genlik yükselmelerini nispeten daha uzun süren sakin ve daha düşük genlik dalgalanmaları takip etmektedir. Test sinyallerinin her ikisi de aynı standart normalizasyon işlemine tabi tutulduğundan A1 sinyali A2'ye kıyasla daha yüksek bir ortalama modülasyon derinliği verecektir. Simülasyon ve testler ortalama modülasyon derinliği optimal değere yakın olacak şekilde gerçekleştirildiğinden, ortalama modülasyon derinliği düşük olan A2 sinyalinde SNR değerinin düşük çıkması kaçınılmaz olacaktır. SNR en yüksek değerini ideal şartlarda optimal modülasyon derinliğinde alır. A1 test sinyaliyle yapılan test ve simülasyon işlemleri neticesinde elde edilen SNR'lerin arasındaki fark 6.86dB olurken, A2 sinyalinde 6.93dB olmaktadır. Farkların birbirine çok yakın çıkıyor olması ortalama modülasyon derinliğinin optimal noktaya yakın olmasıyla SNR'nin yükseldiği tezini doğrulamaktadır. Çünkü test sinyalinin değişmesiyle hem simülasyon ve hem de test için ortalama modülasyon derinliği değişmektedir. Dolayısıyla az bir rastgele ölçüm gürültüsünün getirdiği fark haricinde iki sinyalle yapılan işlemler arasında ciddi bir fark oluşmamaktadır. Genel olarak SNR'leri incelediğimizde ise ortalama 20dB civarında bir değerle karşılaşıyoruz ki bu AM modülasyonu için kabul edilebilir bir değerdir. Buradan yola çıkarak tasarlanan FPGA AM verici sisteminin AM yayınlarını başarıyla gerçekleştirebilecek kapasitede olduğunu söyleyebiliriz.

BÖLÜM 9

SONUÇ

Bu çalışmada, SDR tekniklerine dayalı olarak FPGA platformu üzerinde AM verici görevini yerine getirecek bir sistemin tasarımı ve uygulaması gerçekleştirilmiştir. Çalışmanın ana amacı, SDR temellerini öğretmeye ve öğrenmeye yarayacak basit ve ucuz FPGA tabanlı bir platform geliştirmektir.

Test ve simülasyon çalışmalarından elde edilen sonuçlar incelendiğinde, tartışma kısmında bahsedildiği gibi aynı test sinyali için elde simülasyon sonuçlarının, reel sonuçlardan biraz daha iyi olduğu gözlemlenmiştir. Fakat bu durum, reel testlerde sinyalin harici gürültüye maruz kaldığı göz önünde bulundurulduğunda normal olarak kabul edilebilir. Çünkü bu gürültü sinyalin pc ile ses kartı arasındaki analog bağlantı kabloları üzerindeki iletimi sırasında sızabileceği gibi yine aynı şekilde terminaller arasındaki ufak örnekleme hızı farklarından da kaynaklanabilir.

SNR değerleri üzerinden yapılan çıkarımlarda ise A1 test sinyalinden elde edilen reel ve simülasyon test sonuçları A2 test sinyaline göre biraz daha iyidir. Bu durum A2 test sinyalinin yapısı dolayısıyla ortalama modülasyon derinliğinin daha düşük kalmasından kaynaklanmaktadır. Burada test sinyallerinin her ikisine de aynı standart normalizasyon işlemi uygulandığından A1 sinyali A2'ye kıyasla daha yüksek bir ortalama modülasyon derinliği verecektir. Görüldüğü üzere farkların birbirine çok yakın çıkıyor olması ortalama modülasyon derinliğinin optimal noktaya yakın olmasıyla SNR'nin yükseldiği tezini doğrulamaktadır. Çünkü test sinyalinin değişmesiyle hem simülasyon ve hem de test için ortalama modülasyon derinliği değişmektedir. Bu sebeple ki az bir rastgele ölçüm gürültüsünün getirdiği fark haricinde iki sinyalle yapılan işlemler arasında ciddi bir fark oluşmamaktadır. SNR değerlerine bakıldığında ise ortalama 20dB civarında bir değer çıkmaktadır ve bu AM modülasyonu için kabul edilebilir bir değerdir.

Sonuç olarak tasarımı ve uygulaması yapılan FPGA AM verici sisteminin AM yayınlarının yapılmasına uygun bir aday olduğunu göstermektedir. Geliştirilen platform, bu çalışmanın amacına paralel olarak AM vericilerin SDR teknikleri ile gerçekleştirilmesine yönelik eğitim faaliyetlerine uygundur. Radyo sinyallerinin FPGA'ler üzerinde işlenmesini gösteren iyi bir örnektir.

Sistem, ses kartlarıyla çalışmaya uygun bir son katla (örneğin Softrock Ensemble Transceiver) beraber kullanıldığı takdirde HF ve kısadalga bantlarında amatör telsiz yayınlarının yapılmasına müsaittir. Uygun bir mikro denetleyici üzerinden veya arta kalan lojik elemanlar kullanılarak tasarlanacak bir kullanıcı arabirimi sayesinde platform tek başına çalışır bir hale getirilebilir.

İleri bir çalışma olarak, sistem farklı modülasyon metotlarını içerecek bir hale getirilebilir. Sistem, aynı anda iki istasyondan yayın yapacak şekilde yeniden tasarlanabilir. Uygun bir son kat ve kullanıcı arabirimi tasarımı ile tek başına çalışan bir sistem haline getirilebilir. Böylece sadece eğitim maksadıyla değil de amatör radyoculuk faaliyetleri açısından da faydalı bir düzen elde edilmiş olur.

Aynı şekilde buradaki çalışmada genlik modüleli bir radyo vericisi üzerinde çalışıldığı gibi, farklı modülasyon çeşitleriyle birlikte veya kablosuz haberleşme üzerine yeni çalışmalar yapılarak, SDR teknikleriyle FPGA tabanlı sistemler üzerinde VHDL ile birlikte farklı yazılım ve programlar kullanılarak daha geniş çalışma alanları oluşturulabilir.

KAYNAKLAR

1. A.G.A. Gareane, "Transmit and Receive of FM Signals Using Softrock SDR and Matlab" Yüksek Lisans Tezi, ***Karabük Üniversitesi Fen Bilimleri Enstitüsü***, Karabük (2016).

2. Hervas M., Alsina-Pages R.M. and Salvador M. "An FPGA Scalable Software Defined Radio Platform Design for Educational and Research Purposes", ***IEEE Access*** (2016).

3. Cai X., Zhou M. and Huang X. "Model Based Design for Software Defined Radio on an FPGA", ***IEEE Access*** (2017).

4. Tsoeunyane L., Winberg S. and Inggs M. "Software Defined Radio FPGA Cores: Building Towards a Domain-Specific Language", ***Hindawi*** (2017).

5. Haggui H., Affes S. and Bellili F. "FPGA-SDR Integration and Experimental Validation of a Joint DA ML SNR and Doppler Spread Estimator for 5G Cognitive Transceivers", ***IEEE Access*** (2019).

6. Collins T.F., Getz R., Pu D. and Wyglinski A.M. "Software Defined Radio for Engineers", Çeviri Editörü/Editörleri, **John Gomes,** US (2018).

7. Mady, Z.G.A., "Transmit and Receive of Quadrature Phase-Shift Keying (QPSK) Signal Using Softrock SDR and Matlab", Yüksek Lisans Tezi, ***Karabük Üniversitesi Fen Bilimleri Enstitüsü***, Karabük (2016).

8. Eame M.A.M . "Transmit and Receive of FSK Signals Using Softrock SDR and Matlab", Yüksek Lisans Tezi, ***Karabük Üniversitesi Fen Bilimleri Enstitüsü***, Karabük (2016).

9. Feng Z. "A Software Defined Radio Implementation Using Matlab", ***Vaasan Ammattikorkeakoulu University of Applied Sciences***, Finland (2013).

10. Akpolat A.N. "FPGA Tabanlı Nesne Algılama", Yüksek Lisans Tezi, ***Fırat Üniversitesi Fen Bilimleri Enstitüsü***, Şanlıurfa (2015).

11. Yılmaz N. "Alan Programlamalı Kapı Dizileri (FPGA) Üzerinde Bir YSA'nın Tasarlanması ve Donanım Olarak Gerçekleştirilmesi", Yüksek Lisans Tezi, ***Selçuk Üniversitesi Fen Bilimleri Enstitüsü***, Şanlıurfa (2008).

12. Utrilla R., Rodriguez-Zurrunero R., Martin J., Rozas A. and Araujo A. "A Low-Power Experimental Platform with Programmable Logic Resources and Software-Defined Radio Capabilities ", ***MDPI*** (2019).

13. Kibar A.E. “Genlik Modülasyonu Kullanarak Güç Sinyallerinde Harmonik Çözümleme”, Yüksek Lisans Tezi, ***Gazi Üniversitesi Fen Bilimleri Enstitüsü***, Ankara (2019).
14. Alghiryani S.G.S . “Transmit and Receive of SSB and DSB-AM Signals Using Softrock SDR and Matlab”, Yüksek Lisans Tezi, ***Karabük Üniversitesi Fen Bilimleri Enstitüsü***, Karabük (2016).

15. Dikmen O., “Genlik Modülasyonun İncelenmesini-2”, ***Düzce Üniversitesi Elektrik Elektronik Mühendisliği Bölümü***, Düzce (2016).

16. İnternet: Michigan Technological University, “Am Transmitter-Prelab”, **http://www.ece.mtu.edu/labs/EElabs/EE3305/AM_transmitter.pdf**

17. İnternet: Amateur Radio Station N0GSG, “Chapter 4: AM Transmitters”, **http://n0gsg.com/ecfp/ch4_sample.pdf**

18. Kara F. “VHDL Kullanarak OFDM Gerçeklenmesi”, Yüksek Lisans Tezi, ***Bülent Üniversitesi Fen Bilimleri Enstitüsü***, Zonguldak (2015).

19. Navabi Z. “VHDL: Analysis and Modeling of Digital Systems”, Çeviri Editörü/Editörleri, ***Mc-Craw-Hill***, US (1997).

20. İnternet: Numato Lab, “Mimas Spartan 6 FPGA Development Board”, **https://numato.com/product**

21. İnternet: Numato Lab, “Lm4550 AC’97 Stereo Audio Codec Modüle”, **https://numato.com/product**

22. İnternet: Numato Lab, “IO Breakout Module for Mimas”, **https://numato.com/product**

23. İnternet: Audacity, **https://www.audacityteam.org**

24. İnternet: HDSDR, **http://www.hdsdr.de**

25. İnternet: Xilinx ISE Webpack, **https://www.xilinx.com/products**

26. İnternet: Matlab, **https://www.mathworks.com/products/matlab.html**

27. Kiremitci C., Erkal B. “Eğitim Amaçlı SDR Tekniklerine Dayalı FPGA Tabanlı Genlik Modüleli Radyo Vericisi Tasarımı ve Uygulaması”, **EJOSAT Dergipark Eylül 2020 s.184-189, Özel Sayı**

28. Kiremitci C., Erkal B. “Eğitim Amaçlı SDR Tekniklerine Dayalı FPGA Tabanlı Genlik Modüleli Radyo Vericisi Tasarımı ve Uygulaması”, **1st International Conference on Computer, Electrical and Electronic Science ICCESS 2020, 8-10 Ekim 2020**

EK AÇIKLAMALAR A.

VERİ SAYFALARI

Spartan-6 Family Overview

DS160 (v2.0) October 25, 2011 **Product Specification**

General Description

The Spartan®-6 family provides leading system integration capabilities with the lowest total cost for high-volume applications. The thirteen-member family delivers expanded densities ranging from 3,840 to 147,443 logic cells, with half the power consumption of previous Spartan families, and faster, more comprehensive connectivity. Built on a mature 45 nm low-power copper process technology that delivers the optimal balance of cost, power, and performance, the Spartan-6 family offers a new, more efficient, dual-register 6-input look-up table (LUT) logic and a rich selection of built-in system-level blocks. These include 18 Kb (2 x 9 Kb) block RAMs, second generation DSP48A1 slices, SDRAM memory controllers, enhanced mixed-mode clock management blocks, SelectIO™ technology, power-optimized high-speed serial transceiver blocks, PCI Express® compatible Endpoint blocks, advanced system-level power management modes, auto-detect configuration options, and enhanced IP security with AES and Device DNA protection. These features provide a low-cost programmable alternative to custom ASIC products with unprecedented ease of use. Spartan-6 FPGAs offer the best solution for high-volume logic designs, consumer-oriented DSP designs, and cost-sensitive embedded applications. Spartan-6 FPGAs are the programmable silicon foundation for Targeted Design Platforms that deliver integrated software and hardware components that enable designers to focus on innovation as soon as their development cycle begins.

Summary of Spartan-6 FPGA Features

- Spartan-6 Family:
  - Spartan-6 LX FPGA: Logic optimized
  - Spartan-6 LXT FPGA: High-speed serial connectivity
- Designed for low cost
  - Multiple efficient integrated blocks
  - Optimized selection of I/O standards
  - Staggered pads
  - High-volume plastic wire-bonded packages
- Low static and dynamic power
  - 45 nm process optimized for cost and low power
  - Hibernate power-down mode for zero power
  - Suspend mode maintains state and configuration with multi-pin wake-up, control enhancement
  - Lower-power 1.0V core voltage (LX FPGAs, -1L only)
  - High performance 1.2V core voltage (LX and LXT FPGAs, -2, -3, and -3N speed grades)
- Multi-voltage, multi-standard SelectIO™ interface banks
  - Up to 1,080 Mb/s data transfer rate per differential I/O
  - Selectable output drive, up to 24 mA per pin
  - 3.3V to 1.2V I/O standards and protocols
  - Low-cost HSTL and SSTL memory interfaces
  - Hot swap compliance
  - Adjustable I/O slew rates to improve signal integrity
- High-speed GTP serial transceivers in the LXT FPGAs
  - Up to 3.2 Gb/s
  - High-speed interfaces including: Serial ATA, Aurora, 1G Ethernet, PCI Express, OBSAI, CPRI, EPON, GPON, DisplayPort, and XAUI
- Integrated Endpoint block for PCI Express designs (LXT)
- Low-cost PCI® technology support compatible with the 33 MHz, 32- and 64-bit specification.
- Efficient DSP48A1 slices
  - High-performance arithmetic and signal processing
  - Fast 18 x 18 multiplier and 48-bit accumulator
  - Pipelining and cascading capability
  - Pre-adder to assist filter applications
- Integrated Memory Controller blocks
  - DDR, DDR2, DDR3, and LPDDR support
  - Data rates up to 800 Mb/s (12.8 Gb/s peak bandwidth)
  - Multi-port bus structure with independent FIFO to reduce design timing issues
- Abundant logic resources with increased logic capacity
  - Optional shift register or distributed RAM support
  - Efficient 6-input LUTs improve performance and minimize power
  - LUT with dual flip-flops for pipeline centric applications
- Block RAM with a wide range of granularity
  - Fast block RAM with byte write enable
  - 18 Kb blocks that can be optionally programmed as two independent 9 Kb block RAMs
- Clock Management Tile (CMT) for enhanced performance
  - Low noise, flexible clocking
  - Digital Clock Managers (DCMs) eliminate clock skew and duty cycle distortion
  - Phase-Locked Loops (PLLs) for low-jitter clocking
  - Frequency synthesis with simultaneous multiplication, division, and phase shifting
  - Sixteen low-skew global clock networks
- Simplified configuration, supports low-cost standards
  - 2-pin auto-detect configuration
  - Broad third-party SPI (up to x4) and NOR flash support
  - Feature rich Xilinx Platform Flash with JTAG
  - MultiBoot support for remote upgrade with multiple bitstreams, using watchdog protection
- Enhanced security for design protection
  - Unique Device DNA identifier for design authentication
  - AES bitstream encryption in the larger devices
- Faster embedded processing with enhanced, low cost, MicroBlaze™ soft processor
- Industry-leading IP and reference designs

Spartan-6 FPGA Feature Summary

Table 1: **Spartan-6 FPGA Feature Summary by Device**

Device	Logic Cells[1]	Configurable Logic Blocks (CLBs)			DSP48A1 Slices[3]	Block RAM Blocks		CMTs[5]	Memory Controller Blocks (Max)[6]	Endpoint Blocks for PCI Express	Maximum GTP Transceivers	Total I/O Banks	Max User I/O
		Slices[2]	Flip-Flops	Max Distributed RAM (Kb)		18 Kb[4]	Max (Kb)						
XC6SLX4	3,840	600	4,800	75	8	12	216	2	0	0	0	4	132
XC6SLX9	9,152	1,430	11,440	90	16	32	576	2	2	0	0	4	200
XC6SLX16	14,579	2,278	18,224	136	32	32	576	2	2	0	0	4	232
XC6SLX25	24,051	3,758	30,064	229	38	52	936	2	2	0	0	4	266
XC6SLX45	43,661	6,822	54,576	401	58	116	2,088	4	2	0	0	4	358
XC6SLX75	74,637	11,662	93,296	692	132	172	3,096	6	4	0	0	6	408
XC6SLX100	101,261	15,822	126,576	976	180	268	4,824	6	4	0	0	6	480
XC6SLX150	147,443	23,038	184,304	1,355	180	268	4,824	6	4	0	0	6	576
XC6SLX25T	24,051	3,758	30,064	229	38	52	936	2	2	1	2	4	250
XC6SLX45T	43,661	6,822	54,576	401	58	116	2,088	4	2	1	4	4	296
XC6SLX75T	74,637	11,662	93,296	692	132	172	3,096	6	4	1	8	6	348
XC6SLX100T	101,261	15,822	126,576	976	180	268	4,824	6	4	1	8	6	498
XC6SLX150T	147,443	23,038	184,304	1,355	180	268	4,824	6	4	1	8	6	540

Notes:

1. Spartan-6 FPGA logic cell ratings reflect the increased logic cell capability offered by the new 6-input LUT architecture.
2. Each Spartan-6 FPGA slice contains four LUTs and eight flip-flops.
3. Each DSP48A1 slice contains an 18 x 18 multiplier, an adder, and an accumulator.
4. Block RAMs are fundamentally 18 Kb in size. Each block can also be used as two independent 9 Kb blocks.
5. Each CMT contains two DCMs and one PLL.
6. Memory Controller Blocks are not supported in the -3N speed grade.

Şekil Ek A.1. Mimas Spartan 6 (XC6SLX9-3TQG144)

LM4550B AC '97 Rev 2.1 Multi-Channel Audio Codec With Stereo Headphone Amplifier, Sample Rate Conversion and TI 3D Sound

1 Features

- AC '97 Rev 2.1 Compliant
- High Quality Sample Rate Conversion From 4 kHz to 48 kHz in 1 Hz Increments
- Supports up to 6 DAC Channel Systems With Multiple LM4550Bs or With Other TI LM45xx Codecs
- Unique TI Chaining Function Shares a Single Controller SDATA_IN Pin Among Multiple Codecs
- Stereo Headphone Amp With Separate Gain Control
- TI's 3D Sound Stereo Enhancement Circuitry
- Advanced Power Management Support
- External Amplifier Power-Down (EAPD) Control
- PC Beep Passthrough to Line Out During Initialization or Cold Reset
- Digital 3.3-V and 5-V Supply Options
- Extended Temperature: $-40°C \le T_A \le 85°C$
- Key specifications
  - Analog Mixer Dynamic Range, 97 dB (Typical)
  - DAC Dynamic Range, 89 dB (Typical)
  - ADC Dynamic Range, 90 dB (Typical)
  - Headphone Amp THD+N at 50 mW, 0.02% (Typical) into 32Ω

2 Applications

- Desktop PC Audio Systems on PCI Cards, AMR Cards, or With Motherboard Chips Sets Featuring AC Link
- Portable PC Systems as on MDC Cards, or with a Chipset or Accelerator Featuring AC Link
- General Audio Frequency Systems Requiring 2, 4 or 6 DAC Channels and/or up to 8 ADC Channels
- Automotive Telematics

3 Description

The LM4550B device is an audio codec for PC systems which is fully PC99 compliant and performs the analog intensive functions of the AC '97 Rev 2.1 architecture. Using 18-bit Sigma-Delta ADCs and DACs, the LM4550B provides 90 dB of Dynamic Range.

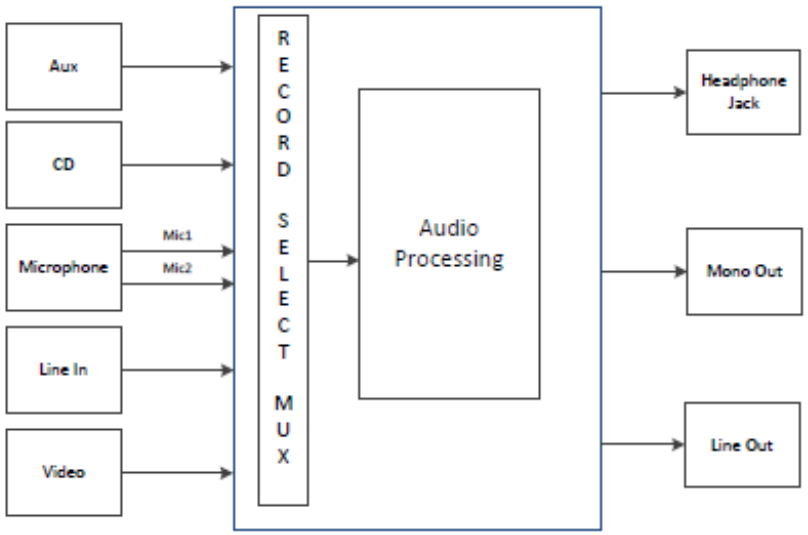
The LM4550B was designed specifically to provide a high quality audio path and provide all analog functionality in a PC audio system. It features full duplex stereo ADCs and DACs and analog mixers with access to 4 stereo and 4 mono inputs.

Device Information[1]

PART NUMBER	PACKAGE	BODY SIZE (NOM)
LM4550B	LQFP (48)	7.00 mm × 7.00 mm

(1) For all available packages, see the orderable addendum at the end of the data sheet.

Simplified Block Diagram



5 Description (continued)

Each mixer input has separate gain, attenuation and mute control and the mixers drive 1 mono and 2 stereo outputs, each with attenuation and mute control. The LM4550B provides a stereo headphone amplifier as one of its stereo outputs and also supports TI's 3D sound stereo enhancement and a comprehensive sample rate conversion capability. The sample rate for the ADCs and DACs can be programmed separately with a resolution of 1 Hz to convert any rate from 4 kHz to 48 kHz. Sample timing from the ADCs and sample request timing for the DACs are completely deterministic to ease task scheduling and application software development. These features together with an extended temperature range also make the LM4550B suitable for non-PC codec applications.

The LM4550B features the ability to connect several codecs together in a system to provide up to 6 simultaneous channels of streaming data on output frames (controller to codec) for surround sound applications. Such systems can also support up to 8 simultaneous channels of streaming data on input frames (codec to controller). Multiple codec systems can be built either using the standard AC Link configuration (that is, of one serial data signal to the controller per codec) or using a unique TI feature for chaining codecs together. This chain feature shares only a single data signal to the controller among multiple codecs.

The AC '97 architecture separates the analog and digital functions of the PC audio system allowing both for system design flexibility and increased performance.

Şekil Ek A.2. LM4550 Audio Codec

EK AÇIKLAMALAR B.

MATLAB KOD LİSTELERİ

Ek B.1. AM Modülasyon Simülasyon Kodu

```
% (DSB-WC) Mod. with MUSIC by B. ERKAL 2020
% AM transmitter code by Bilgehan ERKAL
% Karabuk 2020
clear all;

% sound file 1 loading (4Khz mono (8KSps))
[iff1 , afs]=audioread('a1.wav');
[y1,~]=size(iff1);
% upsample x6 (8x6=48Khz)
yu1=upsample(iff1,6);
% Baseband signal is filtered and normalized
yu1=filter(fir1(255,4e3/24e3),1,yu1);
yu1=yu1./(1.01*max(abs(yu1)));
audiowrite('a1_48k.wav', yu1, 48e3);

fs=48e+3;       % sampling frequency
ts=1/fs;        % sampling interval
t=0:ts:10-ts;   % time axis

% carrier parameters: amplitude, frequency and phase
C1=1;
fct1=12e+3;
tetac1=0*(pi/180);

% carrier signal
ct1=int16(32767*C1*cos(2*pi*fct1*t+tetac1));

% Real AM (DSB-WC) signal
yux=int16(32767*(0.25*yu1+0.5));
m=int16((int32(yux)'.*int32(ct1))/32768);

% IF signal is recorded in wav file
audiowrite('R_DSB_WC.wav', [m' m'], fs);
```

Ek B.2. Demodülasyon Kod Listesi

```
% (DSB-WC) Demod. with MUSIC by B. ERKAL 2020
% AM receiver code by Bilgehan ERKAL
% Karabuk 2020
clear all;

% real IF file loading (48Khz mono)
[yu , afs]=audioread('sim_a2.wav');
yu1=yu(1:end,1)' + yu(1:end,2)';
[~,y1]=size(yu1);

% IF signal is normalized
yu1=yu1./(1.01*max(abs(yu1)));

fs=afs;         % sampling frequency
ts=1/fs;        % sampling interval
t=0:ts:y1/48e3-ts;   % time axis

% carrier parameters: amplitude, frequency and phase
C1=0.1;
fct1=-12e+3;
tetac1=0*(pi/180);

% carrier signal
ct1=C1*exp(2*1i*pi*fct1*t+tetac1);
ct2=C1*exp(2*1i*pi*12e3*t+tetac1);

% demodulation
% complex frequency downshift operation
iffc=yu1.*ct1;

% zero IF cut at 4KHz
yu=filter(fir1(255,4e3/(24e3)),1,iffc);
% complex result is normalized and recorded
yu=yu./(1.01*max(abs(yu)));
audiowrite('a_res1.wav', [real(yu)',imag(yu)'], fs);

% AM detection
% complex upshifting for detector IF ofset
yu=yu.*ct2;
% complex result is normalized and recorded
yu=yu./(1.01*max(abs(yu)));
audiowrite('a_res2.wav', [real(yu)',imag(yu)'], fs);

% AM demodulation using squaring method
% first complex IF is realized
dem=real(yu)+imag(yu);
% real IF is normalized and recorded
```

```
dem=dem./(1.01*max(abs(dem)));
audiowrite('a_res3.wav', dem, fs);
% actual demodulation of real IF signal is accomplished here
dem=dem.*dem;
% Raw demodulation result is normalized and recorded
dem=dem./(1.01*max(abs(dem)));
audiowrite('a_res4.wav', dem, fs);

% Filtered demodulation result is normalized and recorded
dem=filter(fir1(255,4e3/(24e3)),1,dem);
dem=dem./(1.01*max(abs(dem)));
audiowrite('a_res5.wav', dem, fs);
```

Ek B.3. Analiz Kodu

```
% Demod. performance analysis
% AM receiver analysis by Bilgehan ERKAL
% Karabuk 2020
clear all;

% stereo comparison file loading (48Khz stereo)
[iff1 , afs]=audioread('a2_aligned_st_Lsim_Ra2.wav');
[y1,~]=size(iff1);
% channel seperation and gain error correction
rec=1.1634*iff1(1:y1,1)';
a1=1*iff1(1:y1,2)';

% Calculate rms error and rms signal
diff=(a1-rec)/2;
err=(mean(diff.^2))^0.5;
a1_rms=(mean(a1.^2))^0.5;
fprintf('rms error: %d \nSNR(dB): %d \n', err, 20*log10(a1_rms/err));

audiowrite('diff.wav', diff, afs);
```

Ek B.4. Filtre Tasarım Kodu

```
clear all;
% filter cut at 4KHz 48KSps (24KHz)
yu=fir1(254,4e3/24e3);
z=int16(32767*(yu./max(abs(yu))));
%z=z';
freqz(yu);
fid = fopen('exp.txt','w');
fprintf(fid,'%i ',z);
fclose(fid);
```

EK AÇIKLAMALAR C.

VHDL KOD LİSTELERİ

Ek C.1. AM Verici Üst Modülü

```
----------------------------------------------------------------------------------
-- Company:    KARABUK ÜNİVERSİTESİ
-- Engineer:   Bilgehan ERKAL - Caner KİREMİTÇİ
--
-- Create Date:    13:03:25 04/18/2020
-- Design Name:         AM verici üst modül
-- Module Name:    amtx - Behavioral
-- Project Name:   AM Verici
-- Target Devices: Spartan6LX9
-- Tool versions:
-- Description:    AM verici tasarımı
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity amtx is
        Port ( SW                       : in  STD_LOGIC_VECTOR(3  downto
0);
                        RX                      : in STD_LOGIC;--p4_1--green
                        TX                      : out STD_LOGIC;--p4_2--white
                        LED                     :  out  STD_LOGIC_VECTOR(7
downto 0);
                        AUDIO                   : out STD_LOGIC;--p4_3
                        AC97_SDO        :       out         STD_LOGIC;--p3_2--
mimas_p1_11
                        AC97_SDI        : in STD_LOGIC;--p3_3--mimas_p1_14
                        AC97_BIT_CLK    : in STD_LOGIC;--p3_4--mimas_p1_13
                        AC97_RESETN     :       out         STD_LOGIC;--p3_5--
mimas_p1_16
```

```
                        AC97_SYNC       :       out     STD_LOGIC;--p3_6--
mimas_p1_15
                        CLK_IN                  : in STD_LOGIC--100MHz on-
board clock
                       );
end amtx;

architecture Behavioral of amtx is

component pll2
port
 (-- Clock in ports
  CLK_IN1         : in     std_logic;
  -- Clock out ports
  CLK_OUT1        : out    std_logic;
  CLK_OUT2        : out    std_logic
 );
end component;

        COMPONENT AC97_ADAC
        PORT(
                AC97_int_SDI : IN std_logic;
                AC97_int_BIT_CLK : IN std_logic;
                DAC_L : IN std_logic_vector(17 downto 0);
                DAC_R : IN std_logic_vector(17 downto 0);
                clk_48 : IN std_logic;
                AC97_int_SDO : OUT std_logic;
                reset_n : IN std_logic;
                AC97_int_SYNC : OUT std_logic;
                ADC_L : OUT std_logic_vector(17 downto 0);
                ADC_R : OUT std_logic_vector(17 downto 0)
                );
        END COMPONENT;

COMPONENT sercomrx
        PORT(
                bin5            : OUT std_logic_vector(7 downto 0);
                bin4            : OUT std_logic_vector(7 downto 0);
                bin3            : OUT std_logic_vector(7 downto 0);
                bin2            : OUT std_logic_vector(7 downto 0);
                bin1            : OUT std_logic_vector(7 downto 0);
                bin0            : OUT std_logic_vector(7 downto 0);
                level    : OUT std_logic_vector(6 downto 0);
                data_valid: OUT std_logic;
                clk             : IN std_logic;
                serin    : IN std_logic;
                reset_n         : IN std_logic
                );
        END COMPONENT;
```

```
COMPONENT sercomtx
	PORT(
		bin5 : IN std_logic_vector(7 downto 0);
		bin4 : IN std_logic_vector(7 downto 0);
		bin3 : IN std_logic_vector(7 downto 0);
		bin2 : IN std_logic_vector(7 downto 0);
		bin1 : IN std_logic_vector(7 downto 0);
		bin0 : IN std_logic_vector(7 downto 0);
		data_valid : IN std_logic;
		clk : IN std_logic;
		reset_n : IN std_logic;
		binout5 : OUT std_logic_vector(7 downto 0);
		binout4 : OUT std_logic_vector(7 downto 0);
		binout3 : OUT std_logic_vector(7 downto 0);
		binout2 : OUT std_logic_vector(7 downto 0);
		binout1 : OUT std_logic_vector(7 downto 0);
		binout0 : OUT std_logic_vector(7 downto 0);
		serout : OUT std_logic
		);
END COMPONENT;

	COMPONENT frecalc
	PORT(
		digit_in : IN std_logic_vector(27 downto 0);
		reset_n : IN std_logic;
		data_valid : IN std_logic;
		clk : IN std_logic;
		phi_inc_out : OUT std_logic_vector(31 downto 0)
		);
	END COMPONENT;

	COMPONENT am_tx
	PORT(
		phi_inc		: IN std_logic_vector(31 downto 0);
		I_in		: IN std_logic_vector(15 downto 0);
		Q_in		: IN std_logic_vector(15 downto 0);
		s_sel		: IN std_logic_vector(3 downto 0);
		clk_48KHz : IN std_logic;
		clk		: IN std_logic;
		clip_indicator : out std_logic_vector(7 downto 0);
		I_out	: OUT std_logic_vector(15 downto 0);
		Q_out	: OUT std_logic_vector(15 downto 0)
		);
	END COMPONENT;

	COMPONENT dac16
	PORT(
		Clk : IN std_logic;
```

```
            Data : IN std_logic_vector(15 downto 0);
            PulseStream : OUT std_logic
            );
    END COMPONENT;

-- serial communication module signals
-- command completion stage indicator
signal stage : std_logic_vector(6 downto 0) := (others => '0');
signal sample  : std_logic := '0';
-- command data valid indicators
signal data_valid  : std_logic := '0';
-- completed command data
signal dat5 : std_logic_vector(7 downto 0) := (others => '0');
signal dat4 : std_logic_vector(7 downto 0) := (others => '0');
signal dat3 : std_logic_vector(7 downto 0) := (others => '0');
signal dat2 : std_logic_vector(7 downto 0) := (others => '0');
signal dat1 : std_logic_vector(7 downto 0) := (others => '0');
signal dat0 : std_logic_vector(7 downto 0) := (others => '0');
-- command data application digits indicating control frequency data
signal dig5 : std_logic_vector(7 downto 0) := (others => '0');
signal dig4 : std_logic_vector(7 downto 0) := (others => '0');
signal dig3 : std_logic_vector(7 downto 0) := (others => '0');
signal dig2 : std_logic_vector(7 downto 0) := (others => '0');
signal dig1 : std_logic_vector(7 downto 0) := (others => '0');
signal dig0 : std_logic_vector(7 downto 0) := (others => '0');
-- AM receiver module signals
-- phase increment value necessary to steer frequency of primary nco of am receiver
signal phi_inc : std_logic_vector(31 downto 0) := (others => '0');
-- AM receiver output
signal I_out : std_logic_vector(15 downto 0) := (others => '0');
signal Q_out : std_logic_vector(15 downto 0) := (others => '0');
-- AM receiver input
signal I_in : std_logic_vector(15 downto 0) := (others => '0');
signal Q_in : std_logic_vector(15 downto 0) := (others => '0');
-- FPGA master reset signal
signal res_count : std_logic_vector(24 downto 0) := (others => '0');
signal reset_n : std_logic := '0';
-- ADAC (Audio card) reset signal
signal reset_n2 : std_logic := '0';
signal res_count2 : std_logic_vector(10 downto 0) := (others => '0');
-- clk_12288 12.288MHz clock live indicator
signal flash : std_logic_vector(22 downto 0) := (others => '0');
signal clk_count : std_logic_vector(8 downto 0) := (others => '0');
-- ADAC input and output signals
signal adata_L        : std_logic_vector(17 downto 0) := (others => '0');
signal adata_R        : std_logic_vector(17 downto 0) := (others => '0');
signal dacdata_L      : std_logic_vector(17 downto 0) := (others => '0');
signal dacdata_R      : std_logic_vector(17 downto 0) := (others => '0');
-- clipping indicator
```

```
signal c_ind            : std_logic_vector(7 downto 0);
--clock signals
signal clk                      : std_logic := '0';-- 36.684MHz master clock
signal clk_48KHz        : std_logic := '0';-- 48KHz sampling rate clock
signal clk_12288        : STD_LOGIC;                    -- 12.288MHz ADAC master
clock
signal clk_6144         : STD_LOGIC := '0';-- 6.144MHz pll2 input clock
signal clk_36864        : STD_LOGIC;                    -- 36.684MHz master clock

begin

-- master module connectors
clk <= clk_36864;
AC97_RESETN <= reset_n2;
--rx--p4_1 --> tx pin of usb2serial cable (green)
--tx--p4_2 --> rx pin of usb2serial cable (white)
--gnd--(black)
--audio--p4_3
-- ADAC connectors
--ADC outputs
I_in <= adata_L(17 downto 2);
Q_in <= adata_R(17 downto 2);
--DAC inputs
dacdata_L <= I_out(15) & I_out(15 downto 0) & "0";
dacdata_R <= Q_out(15) & Q_out(15 downto 0) & "0";
-- led indicator connectors, uncomment necessary and
-- comment out unnecessary
--led(6 downto 0) <= stage(6 downto 0);--sercom completion levels
--led(7) <= flash(22);-- clock live indicator
led(7 downto 0) <= c_ind(7 downto 0);--clipping indicators

-- master clock generator
Inst_pll2 : pll2
  port map
   (-- Clock in ports
    CLK_IN1 => clk_6144,
    -- Clock out ports
    CLK_OUT1 => clk_36864,
    CLK_OUT2 => clk_12288
          );
-- pll2 input reference frequency (6.144MHz) derivator
-- master reference used is half of ADAC Bit clock at 12.288MHz
clk_6144_proc:process(AC97_BIT_CLK)
        begin
                if rising_edge(AC97_BIT_CLK) then
                        clk_6144 <= not clk_6144;
                end if;
        end process;
```

```
-- Sampling rate clock generator derived from 36.864MHz master clock
--36864/(384*2) = 48KHz
clk48KHz_proc:process(clk)
	begin
		if rising_edge(clk) then
			if reset_n = '1' then
				if clk_count = 383 then
					clk_48KHz <= not clk_48KHz;
					clk_count <= (others => '0');
				else
					clk_count <= clk_count + 1;
					clk_48KHz <= clk_48KHz;
				end if;
			else
				clk_count <= clk_count;
				clk_48KHz <= '0';
			end if;
		end if;
	end process;

-- Master reset of FPGA fabric
	reset_proc: process(AC97_BIT_CLK)
		begin
			if rising_edge(AC97_BIT_CLK) then
				if res_count(24) = '1' then
					res_count <= res_count;
				else
					res_count <= res_count + 1;
				end if;
			end if;
		end process;

	reset_n <= res_count(24);

-- ADAC reset generator, completed before master FPGA reset
-- It is solely derived from 100.00MHz FPGA master clock which is
-- the only live and stable clock before ADAC reset is completed
	AC97_reset_proc: process(clk_in)
		begin
			if rising_edge(clk_in) then
				if res_count2(10) = '1' then
					res_count2 <= res_count2;
				else
					res_count2 <= res_count2 + 1;
				end if;
			end if;
		end process;

	reset_n2 <= res_count2(10);
```

```
-- 12.288MHz clock live indicator
	flash_proc: process(clk_12288)
		begin
			if rising_edge(clk_12288) then
				flash <= not flash;
			end if;
		end process;

-- command data receive complete indicator
	sample_proc: process(clk)
		begin
			if rising_edge(clk) then
				if data_valid = '1' then
					sample <= not sample;
				else
					sample <= sample;
				end if;
			end if;
		end process;

-- ADAC module (AC97 soundcard module)
	Inst_AC97_ADAC: AC97_ADAC PORT MAP(
		AC97_int_SDO => AC97_SDO,
		AC97_int_SDI => AC97_SDI,
		AC97_int_BIT_CLK => AC97_BIT_CLK,
		reset_n => reset_n,
		AC97_int_SYNC => AC97_SYNC,
		DAC_L => dacdata_L,
		DAC_R => dacdata_R,
		ADC_L => adata_L,
		ADC_R => adata_R,
		clk_48 => clk_48KHz
	);

-- Serial communication receive module
-- This module is used to accept commands from PC at 9600bps
	Inst_sercomrx: sercomrx PORT MAP(
		bin5 => dat5,
		bin4 => dat4,
		bin3 => dat3,
		bin2 => dat2,
		bin1 => dat1,
		bin0 => dat0,
		level => stage,
		data_valid => data_valid,
		clk => clk,
		serin => rx,
		reset_n => reset_n
```

```
	);

-- Serial communication transmit module
-- This module is used to echo received commands to PC at 9600bps
	Inst_sercomtx: sercomtx PORT MAP(
		binout5 => dig5,
		binout4 => dig4,
		binout3 => dig3,
		binout2 => dig2,
		binout1 => dig1,
		binout0 => dig0,
		bin5 => dat5,
		bin4 => dat4,
		bin3 => dat3,
		bin2 => dat2,
		bin1 => dat1,
		bin0 => dat0,
		data_valid => data_valid,
		clk => clk,
		serout => tx,
		reset_n => reset_n
	);

-- Frequency calculation module
-- Takes frequency data which comes from PC as input
-- and calculates phase increment factor necessary to steer
-- primary nco frequency used in the AM RX module
Inst_frecalc: frecalc PORT MAP(
		digit_in => dig5 & dig4(3 downto 0) & dig3(3 downto 0) & dig2(3
downto 0) & dig1(3 downto 0) & dig0(3 downto 0),
		phi_inc_out => phi_inc,
		reset_n => reset_n,
		data_valid => data_valid,
		clk => clk
	);

-- Actual AM xmitter module
	Inst_am_tx: am_tx PORT MAP(
		phi_inc => phi_inc,
		I_out => I_out,
		Q_out => Q_out,
		I_in => I_in,
		Q_in => Q_in,
		s_sel => SW,
		clip_indicator => c_ind,
		clk_48KHz => clk_48KHz,
		clk => clk
	);
```

```
-- Auxilary analog output port as Sigma-Delta DAC
	Inst_dac16: dac16 PORT MAP(
		Clk => clk,
		Data => not I_out(15) & I_out(14 downto 0),
		PulseStream => audio
	);

end Behavioral;
```

Ek C.2. AM_TX Am Verici Alt Modülü

```
----------------------------------------------------------------------------------
-- Company:    KARABÜK ÜNİVERSİTESİ
-- Engineer:     Bilgehan ERKAL - Caner KİREMİTÇİ
--
-- Create Date:    10:24:23 05/18/2020
-- Design Name:
-- Module Name:    am_tx - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity am_tx is
    Port (
          phi_inc : in std_logic_vector(31 downto 0);
          I_out  : out std_logic_vector(15 downto 0);
          Q_out          : out std_logic_vector(15 downto 0);
          I_in    : in std_logic_vector(15 downto 0);
          Q_in   : in std_logic_vector(15 downto 0);
          s_sel   : in std_logic_vector(3 downto 0);
          clip_indicator : out std_logic_vector(7 downto 0);
          clk_48KHz : in  STD_LOGIC;
          clk              : in  STD_LOGIC

          );
end am_tx;

architecture Behavioral of am_tx is
```

```
	COMPONENT clipping_indicator_multi_channel
	PORT(
		clip_in_I : IN std_logic_vector(15 downto 0);
		clip_in_Q : IN std_logic_vector(15 downto 0);
		clk : IN std_logic;
		clip_out : OUT std_logic
		);
	END COMPONENT;

COMPONENT nco1
  PORT (
    clk : IN STD_LOGIC;
    pinc_in : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
    cosine : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
    sine : OUT STD_LOGIC_VECTOR(15 DOWNTO 0)
  );
END COMPONENT;

component fir1
	port (
	clk: in std_logic;
	rfd: out std_logic;
	rdy: out std_logic;
	din: in std_logic_vector(15 downto 0);
	dout: out std_logic_vector(34 downto 0));
end component;

COMPONENT mult
  PORT (
    clk : IN STD_LOGIC;
    a : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
    b : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
    p : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
  );
END COMPONENT;

-- first complex multiplier and nco signals
signal cos : std_logic_vector(15 downto 0) := (others => '0');
signal sin : std_logic_vector(15 downto 0) := (others => '0');

-- Input LPF signals
signal lpf_din : std_logic_vector(15 downto 0) := (others => '0');
signal lpf_dout : std_logic_vector(34 downto 0) := (others => '0');

-- offset circuit signals
signal adder_in : std_logic_vector(15 downto 0) := (others => '0');
signal adder_out : std_logic_vector(15 downto 0) := (others => '0');
```

```
-- AM modulator signals
signal mod_in : std_logic_vector(15 downto 0) := (others => '0');
signal mod_out : std_logic_vector(31 downto 0) := (others => '0');

-- module outputs
signal I_out_x : std_logic_vector(15 downto 0) := (others => '0');
signal Q_out_x : std_logic_vector(15 downto 0) := (others => '0');

begin

-- intercomponent coarse level adjustment connectors
-- Input LPF inputs
lpf_din <= I_in;
--offset circuit inputs
adder_in <= lpf_dout(34) & lpf_dout(30 downto 16);
-- AM modulator input
mod_in <= adder_out(15) & adder_out(14 downto 0);
-- Module outputs
I_out_x <= mod_out(31) & mod_out(29 downto 15);
Q_out_x <= mod_out(31) & mod_out(29 downto 15);

-- Source selector process
s_sel_proc: process(clk)
	begin
		if rising_edge(clk) then
			CASE s_sel(3 downto 0) IS
				WHEN "1111" =>-- modulated final output
					I_out <= I_out_x;
					Q_out <= Q_out_x;

				WHEN "1110" =>-- Module input (input LPF input)
					I_out <= I_in;
					Q_out <= Q_in;

				WHEN "1101" =>-- Input LPF output, offset circuit
input
					I_out <= adder_in;
					Q_out <= adder_in;

				WHEN "1011" =>-- nco output (in complex form)
					I_out <= cos;
					Q_out <= sin;

				WHEN "0111" =>-- Offset circuit output, AM
modulator input
					I_out <= mod_in;
					Q_out <= mod_in;

				WHEN OTHERS =>-- module output
```

```
                                        I_out <= I_out_x;
                                        Q_out <= Q_out_x;

                                END CASE;
                        end if;
                end process;
-- Module input (input LPF input) clipping indicator
cind0: clipping_indicator_multi_channel PORT MAP(
                        clip_in_I => I_in,
                        clip_in_Q => Q_in,
                        clip_out => clip_indicator(0),
                        clk => clk
                );
-- offset circuit input clipping indicator
cind1: clipping_indicator_multi_channel PORT MAP(
                        clip_in_I => adder_in,
                        clip_in_Q => adder_in,
                        clip_out => clip_indicator(1),
                        clk => clk
                );
-- AM modulator input clipping indicator
cind2: clipping_indicator_multi_channel PORT MAP(
                        clip_in_I => mod_in,
                        clip_in_Q => mod_in,
                        clip_out => clip_indicator(2),
                        clk => clk
                );
-- Module output (AM Modulator output) clipping indicator
cind3: clipping_indicator_multi_channel PORT MAP(
                        clip_in_I => I_out_x,
                        clip_in_Q => I_out_x,
                        clip_out => clip_indicator(3),
                        clk => clk
                );
-- Empty indicator (reserved for future use)
clip_indicator(7 downto 4) <= "0000";

-- Input LPF (255 coefficient FIR, fs=48KHz, fc=4KHz)
lpf : fir1
                        port map (
                                clk => clk,
                                rfd => open,
                                rdy => open,
                                din => lpf_din,
                                dout => lpf_dout
                        );

-- Offset circuit
adder_proc:process(clk_48KHz)
```

```
        begin
                if rising_edge(clk_48KHz) then
                        adder_out <= adder_in + 16384;
                end if;
        end process;

-- First NCO (steered by Phase increment value calculated from PC input frequency data)
Inst_nco1 : nco1
  PORT MAP (
    clk => clk_48KHz,
    pinc_in => phi_inc,
    cosine => cos,
    sine => sin
  );

-- Modulator circuit (AM modulator)
Inst_mult : mult
  PORT MAP (
    clk => clk_48KHz,
    a => mod_in,
    b => cos,
    p => mod_out
  );

end Behavioral;
```

Ek C.3. frecalc Faz Artım Hesaplama Alt Modülü

```
----------------------------------------------------------------------------------
-- Company:    KARABÜK ÜNİVERSİTESİ
-- Engineer:    Bilgehan ERKAL - Caner KİREMİTÇİ
--
-- Create Date:    17:46:16 04/28/2020
-- Design Name:
-- Module Name:    frecalc - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity frecalc is
Port(
digit_in                    : in STD_LOGIC_VECTOR(27 downto 0);
phi_inc_out          : out std_logic_vector(31 downto 0) := (others => '0');
reset_n                     : in STD_LOGIC;
data_valid                  : in STD_LOGIC;
clk                                 : in std_logic

);
end frecalc;

architecture Behavioral of frecalc is

COMPONENT FREQ
        PORT(
                bin : IN std_logic_vector(27 downto 0);
```

```
		pinc : OUT std_logic_vector(31 downto 0)
		);
END COMPONENT;

signal phi_inc : std_logic_vector(31 downto 0) := (others => '0');--calculated phase increment
signal freq_counter : std_logic_vector(7 downto 0) := (others => '0');--wait counter
signal pinc : std_logic_vector(31 downto 0) := (others => '0');--precalculation register
signal bin : STD_LOGIC_VECTOR(27 downto 0);--input frequency data in BCD format + sign data


begin
phi_inc_out <= phi_inc;

	Inst_FREQ: FREQ PORT MAP(
		bin => digit_in,
		pinc => pinc
	);

	freq_proc:process(clk)
		begin
			if rising_edge(clk) then
				if reset_n = '1' then
					if freq_counter = X"00" then--wait for new data
						phi_inc <= phi_inc;
						if data_valid = '1' then--there is new data so start calculation
							freq_counter <= freq_counter + 1;
						else
							freq_counter <= freq_counter;
						end if;
					else-- calculation in progress
						freq_counter <= freq_counter + 1;
						if freq_counter = X"FF" then--calculation is complete
							phi_inc <= pinc;--update phase increment output with new one
						else--wait till calculation is complete
							phi_inc <= phi_inc;
						end if;
					end if;
				else--reset state
					freq_counter <= X"01";
					phi_inc <= (others => '0');
				end if;
			end if;
		end process;
```

Ek C.4. sercomrx Seri Veri Alıcı Alt Modülü

```
----------------------------------------------------------------------------------
-- Company:    KARABÜK ÜNİVERSİTESİ
-- Engineer:    Bilgehan ERKAL - Caner KİREMİTÇİ
--
-- Create Date:    17:48:58 04/01/2020
-- Design Name:
-- Module Name:    sercomrx - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity sercomrx is
Port(
                bin5            : OUT std_logic_vector(7 downto 0);
                bin4            : OUT std_logic_vector(7 downto 0);
                bin3            : OUT std_logic_vector(7 downto 0);
                bin2            : OUT std_logic_vector(7 downto 0);
                bin1            : OUT std_logic_vector(7 downto 0);
                bin0            : OUT std_logic_vector(7 downto 0);
                level    : OUT std_logic_vector(6 downto 0);
                data_valid: OUT std_logic;
                clk             : IN std_logic;
                serin    : IN std_logic;
                reset_n         : IN std_logic

);
end sercomrx;
```

```
architecture Behavioral of sercomrx is

--Received data register
signal data : std_logic_vector(7 downto 0) := (others => '0');
--Received command completion data
signal stage : std_logic_vector(6 downto 0) := (others => '0');
--Received data bins in ASCII form
signal dat5 : std_logic_vector(7 downto 0) := (others => '0');--Sign (+/-)
signal dat4 : std_logic_vector(7 downto 0) := (others => '0');--MSB
signal dat3 : std_logic_vector(7 downto 0) := (others => '0');
signal dat2 : std_logic_vector(7 downto 0) := (others => '0');
signal dat1 : std_logic_vector(7 downto 0) := (others => '0');
signal dat0 : std_logic_vector(7 downto 0) := (others => '0');--LSB
--Received bins counter
signal dat_count : std_logic_vector(2 downto 0) := (others => '0');
--Serial input resample shift register
--Input is resampled 4 times in order to catch bits appropriately (10bits *4 = 40bits)
signal bits : std_logic_vector(39 downto 0) := (others => '1');
--resampling clock divider
signal counter : std_logic_vector(9 downto 0) := (others => '0');
--Received byte ok signal
signal byte_ok : std_logic := '0';
--Received byte ok signal delay registers
signal dat_ok1 : std_logic := '0';
signal dat_ok  : std_logic := '0';
--Data valid signal, When dat_valid is active there is new and valid data at data bin outputs
signal blink  : std_logic := '0';
signal blink_del  : std_logic := '0';
signal dat_valid  : std_logic := '0';

begin
--calculate and generate byte ok signal (indicates that valid start-bit- 8-bit data and stop-bit sequence in the bits resampling register)
        byte_ok <= -- start-bit "0" bits(2:1) + LSB first, MSB last data bits (0:7) + stopbit "0" bits(38:37)
                                (bits(38) and bits(37)) and (bits(34) xnor bits(33)) and (bits(30) xnor bits(29)) and (bits(26) xnor bits(25)) and
                        (bits(22) xnor bits(21)) and (bits(18) xnor bits(17)) and (bits(14) xnor bits(13)) and (bits(10) xnor bits(9)) and
                        (bits(6) xnor bits(5)) and (bits(2) nor bits(1));
--module connectors
        bin5 <= dat5;
        bin4 <= dat4;
        bin3 <= dat3;
        bin2 <= dat2;
        bin1 <= dat1;
        bin0 <= dat0;
```

```
	data_valid <= dat_valid;
	level <= stage;
--serial communication process
	ser_comm: process(clk)--ticks at 36864MHz
		begin
			if rising_edge(clk) then
				if reset_n = '1' then
					if counter = 959 then --resampling clock at 36864/960 = 4* 9600 = 38400
						bits <= serin & bits(39 downto 1);--resample serial input, data comes as LSB first
						counter <= (others => '0');--reset counter
					else
						counter <= counter+1;--wait till next sample
					end if;
					if byte_ok = '1' then-- there is valid data in bits resample register transfer it to data register
						data <= bits(34) & bits(30) & bits(26) & bits(22) & bits(18) & bits(14) & bits(10) & bits(6);
						bits <= (others => '1');--setup bits register for next new data
					end if;
				else--reset in order
					data(7 downto 0) <= (others => '0');
					bits(39 downto 0) <= (others => '1');
					counter(9 downto 0) <= (others => '0');
				end if;
			end if;
		end process;
--byte_ok signal delay process
	dat_ok_proc: process(clk)
		begin
			if rising_edge(clk) then
				dat_ok1 <= byte_ok;
				dat_ok <= dat_ok1;
			end if;
		end process;
--transfer valid data to apropriate data bin process
-- a FSM is used to track the next data bin to reload
-- if data is not valid for the actual data bin
-- all data located in the data bins so far is discarded
-- and the process starts from the beginning
	dat_xfer_proc: process(clk)
		begin
			if rising_edge(clk) then
				if reset_n = '1' then
					if dat_ok = '1' then
						CASE dat_count(2 downto 0) IS
```

```
                                        WHEN "000" => --test data for f-
command
                                                dat5 <= dat5;
                                                dat4 <= dat4;
                                                dat3 <= dat3;
                                                dat2 <= dat2;
                                                dat1 <= dat1;
                                                dat0 <= dat0;
                                                blink <= blink;
                                            if data = 102 then--
character "f" is received, next incoming data must be sign(+ or -)
                                                stage(0) <= '1';--
first stage is completed successfully
                                                dat_count <=
dat_count + 1;--proceed with next data
                                            else
                                                dat_count <=
(others => '0');-- received data is not valid (other than "f" character), start from
beginning
                                                stage <= (others =>
'0');-- clear all stages
                                            end if;
                                        WHEN "001" => --test data for
sign
                                                dat4 <= dat4;
                                                dat3 <= dat3;
                                                dat2 <= dat2;
                                                dat1 <= dat1;
                                                dat0 <= dat0;
                                                blink <= blink;
                                            if ((data = 43) or (data =
45)) then--data is plus or minus character
                                                stage(1) <= '1';
                                                dat5 <= data;--
record it in first data bin from left
                                                dat_count <=
dat_count + 1;
                                            else--start all over again
                                                dat_count <=
(others => '0');
                                                stage <= (others =>
'0');
                                                dat5 <= dat5;

                                            end if;
                                        WHEN "010" =>--test for msb (it
must be either a "0", "1" or "2")
                                                dat5 <= dat5;
                                                dat3 <= dat3;
```

```
                                        dat2 <= dat2;
                                        dat1 <= dat1;
                                        dat0 <= dat0;
                                        blink <= blink;
                                    if ((data > 47) and (data <
51)) then--it is a "0", "1" or "2"
                                        stage(2) <= '1';
                                        dat4 <= data;--then
register it
                                        dat_count       <=
dat_count + 1;
                                    else--start all over again
                                        dat_count       <=
(others => '0');
                                        stage <= (others =>
'0');
                                        dat4 <= dat4;
                                    end if;
                                WHEN "011" =>--test for second
msb (it must be either a "0", "1", "2" or "3" if first msb is "2" otherwise 0-9)
                                        dat5 <= dat5;
                                        dat4 <= dat4;
                                        dat2 <= dat2;
                                        dat1 <= dat1;
                                        dat0 <= dat0;
                                        blink <= blink;
                                    if dat4 = 50 then--first msb
is "2"
                                        if ((data > 47) and
(data < 52)) then
                                            stage(3) <=
'1';
                                            dat3    <=
data;
                                            dat_count
<= dat_count + 1;
                                        else--start all over
again
                                            dat_count
<= (others => '0');
                                            stage    <=
(others => '0');
                                            dat3    <=
dat3;
                                        end if;
                                    else-- first msb is 0 or 1
                                        if ((data > 47) and
(data < 58)) then
```

```
                                        stage(3) <=
'1';
                                        dat3     <=
data;
                                        dat_count
<= dat_count + 1;
                                    else--start all over
again
                                        dat_count
<= (others => '0');
                                        stage     <=
(others => '0');
                                        dat3      <=
dat3;
                                    end if;
                                end if;
                            WHEN "100" =>--third msb (it
can be 0-9)
                                    dat5 <= dat5;
                                    dat4 <= dat4;
                                    dat3 <= dat3;
                                    dat1 <= dat1;
                                    dat0 <= dat0;
                                    blink <= blink;
                                if ((data > 47) and (data <
58)) then
                                    stage(4) <= '1';
                                    dat2 <= data;
                                    dat_count        <=
dat_count + 1;
                                else--start all over again
                                    dat_count        <=
(others => '0');
                                    stage <= (others =>
'0');
                                    dat2 <= dat2;
                                end if;
                            WHEN "101" =>-- fourth msb (0-
9)
                                    dat5 <= dat5;
                                    dat4 <= dat4;
                                    dat3 <= dat3;
                                    dat2 <= dat2;
                                    dat0 <= dat0;
                                    blink <= blink;
                                if ((data > 47) and (data <
58)) then
                                    stage(5) <= '1';
                                    dat1 <= data;
```

```
                                        dat_count       <=
dat_count + 1;
                                    else--start all over again
                                        dat_count       <=
(others => '0');
                                        stage <= (others =>
'0');
                                        dat1 <= dat1;
                                    end if;
                                WHEN "110" =>--lsb (0-9)
                                        dat5 <= dat5;
                                        dat4 <= dat4;
                                        dat3 <= dat3;
                                        dat2 <= dat2;
                                        dat1 <= dat1;
                                    dat_count  <=  (others  =>
'0');--start at the beginning for new data
                                    if ((data > 47) and (data <
58)) then
                                        stage(6) <= '1';
                                        dat0 <= data;
                                        blink     <=     not
blink;--changeover blink
                                    else--start all over again
                                        stage <= (others =>
'0');
                                        blink <= blink;
                                        dat0 <= dat0;
                                    end if;
                                WHEN   OTHERS   =>--start   all
over again
                                        dat_count       <=
(others => '0');
                                        dat5 <= dat5;
                                        dat4 <= dat4;
                                        dat3 <= dat3;
                                        dat2 <= dat2;
                                        dat1 <= dat1;
                                        dat0 <= dat0;
                                        blink <= blink;
                                        stage <= (others =>
'0');
                            END CASE;
                        else-- there is no valid new data so wait for one
to come
                            dat_count <= dat_count;
                            dat5 <= dat5;
                            dat4 <= dat4;
                            dat3 <= dat3;
```

```
                                        dat2 <= dat2;
                                        dat1 <= dat1;
                                        dat0 <= dat0;
                                        blink <= blink;
                                        stage <= stage;
                                end if;
                        else-- reset in order
                                dat_count <= (others => '0');
                                dat5 <= (others => '0');
                                dat4 <= (others => '0');
                                dat3 <= (others => '0');
                                dat2 <= (others => '0');
                                dat1 <= (others => '0');
                                dat0 <= (others => '0');
                                blink <= '0';
                                stage <= (others => '0');
                        end if;
                end if;
        end process;
-- data valid signal process
        dat_valid_proc: process(clk)
                begin
                        if rising_edge(clk) then
                                if reset_n = '1' then
                                        dat_valid <= blink xor blink_del;--there is changeover in blink so there is new valid data
                                        blink_del <= blink;-- delay blink signal so that data valid signal is active for only one clock cycle
                                else--reset in order
                                        dat_valid <= '0';
                                        blink_del <= '0';
                                end if;
                        end if;
                end process;
end Behavioral;
```

Ek C.5. sercomtx Seri Veri Vericisi Alt Modülü

```
----------------------------------------------------------------------------------
-- Company:    KARABÜK ÜNİVERSİTESİ
-- Engineer:    Bilgehan ERKAL - Caner KİREMİTÇİ
--
-- Create Date:    17:48:58 04/01/2020
-- Design Name:
-- Module Name:    sercomrx - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity sercomtx is
Port(
                binout5                 : OUT std_logic_vector(7 downto 0);
                binout4                 : OUT std_logic_vector(7 downto 0);
                binout3                 : OUT std_logic_vector(7 downto 0);
                binout2                 : OUT std_logic_vector(7 downto 0);
                binout1                 : OUT std_logic_vector(7 downto 0);
                binout0                 : OUT std_logic_vector(7 downto 0);
                bin5            : IN std_logic_vector(7 downto 0);
                bin4            : IN std_logic_vector(7 downto 0);
                bin3            : IN std_logic_vector(7 downto 0);
                bin2            : IN std_logic_vector(7 downto 0);
                bin1            : IN std_logic_vector(7 downto 0);
                bin0            : IN std_logic_vector(7 downto 0);
                data_valid: IN std_logic;
                clk             : IN std_logic;
```

```
                serout  : OUT std_logic;
                reset_n         : IN std_logic

);
end sercomtx;

architecture Behavioral of sercomtx is

COMPONENT text_rom
  PORT (
    clka : IN STD_LOGIC;
    addra : IN STD_LOGIC_VECTOR(5 DOWNTO 0);
    douta : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
  );
END COMPONENT;
-- output digit data registers
signal dig5 : std_logic_vector(7 downto 0) := (others => '0');
signal dig4 : std_logic_vector(7 downto 0) := (others => '0');
signal dig3 : std_logic_vector(7 downto 0) := (others => '0');
signal dig2 : std_logic_vector(7 downto 0) := (others => '0');
signal dig1 : std_logic_vector(7 downto 0) := (others => '0');
signal dig0 : std_logic_vector(7 downto 0) := (others => '0');
-- serial data transfer registers
signal busyshiftreg : std_logic_vector(9 downto 0) := (others => '0');
signal datashiftreg : std_logic_vector(9 downto 0) := (others => '1');
-- clock divider counter
signal txcounter : std_logic_vector(12 downto 0) := (others => '0');
-- data register, reloading register for serial data transfer register
signal data : std_logic_vector(7 downto 0) := (others => '0');
-- caption text rom signals
signal rom_counter : std_logic_vector(7 downto 0) := (others => '0');
signal douta : std_logic_vector(7 downto 0) := (others => '0');
-- internal control signals
signal state_counter : std_logic_vector(3 downto 0) := (others => '0');
signal timer : std_logic_vector(1 downto 0) := (others => '0');
signal valid_data : std_logic := '0';
signal data2send : std_logic := '0';

begin
--caption text rom
Inst_romtext : text_rom
  PORT MAP (
    clka => clk,
    addra => rom_counter(5 downto 0),
    douta => douta
  );
 -- module connectors
          serout <= datashiftreg(0);--data shifted as LSB first MSB last
          data2send <= (valid_data);
```

```
	binout5 <= dig5;
	binout4 <= dig4;
	binout3 <= dig3;
	binout2 <= dig2;
	binout1 <= dig1;
	binout0 <= dig0;
-- main serial communication process
	ser_comm_tx: process(clk)
		begin
			if rising_edge(clk) then
				if reset_n = '1' then
					if data2send = '1' then--there is data waiting to be
sent
						if busyshiftreg(0) = '0' then--sender is not
busy then reload new data from data register to data shift register
							busyshiftreg <= (others => '1');--
set busy signal to prevent unintended reloading of data shift register
							txcounter <= (others => '0');--
clear tx counter for the timing of new transfer
							datashiftreg <= '1' & data & '0';--
reload datashift register with fresh data and also include start and stop bits
						else
							if txcounter = 3839 then--bit clock
= 36864/3840 = 9.6kbps
								datashiftreg <= '1' &
datashiftreg(9 downto 1);--time to shift out a new bit
								busyshiftreg <= '0' &
busyshiftreg(9 downto 1);--count sent bits when complete busy signal is made inactive
automatically
								txcounter <= (others =>
'0');-- clear clock divider
							else--wait till next bit to out
								txcounter <= txcounter+1;
								datashiftreg <=
datashiftreg;
								busyshiftreg <=
busyshiftreg;
							end if;
						end if;
					else--there is no new data so wait in ready state
(not busy)
						busyshiftreg <= (others => '0');
						txcounter <= (others => '0');
						datashiftreg <= (others => '1');

					end if;
				else-- reset in order
					busyshiftreg <= (others => '0');
					txcounter <= (others => '0');
```

```
                                        datashiftreg <= (others => '1');
                                end if;
                        end if;
                end process;
-- outgoing data reloading process
        data_proc: process(clk)
                begin
                        if rising_edge(clk) then
                                if reset_n = '1' then
                                        CASE state_counter(3 downto 0) IS
                                                WHEN "0000" =>-- caption text state,
withdraw text data from rom
                                                        if (rom_counter = 64) and
(busyshiftreg(0) = '0') then-- end of rom data so proceed with sending out incoming
data (echo received data)
                                                                state_counter <=
state_counter + 1;
                                                                valid_data <= '0';
                                                                rom_counter <=
rom_counter;
                                                                data <= data;
                                                        else--withdraw text data from rom
                                                                state_counter <=
state_counter;
                                                                valid_data <= '1';
                                                                if busyshiftreg(0) = '0'
then-- serial transmitter is ready for new data
                                                                        rom_counter <=
rom_counter + 1;--proceed with next line
                                                                        data <= douta;--
reload new data
                                                                else--serial xmitter is busy
so wait until not busy
                                                                        rom_counter <=
rom_counter;
                                                                        data <= data;
                                                                end if;
                                                        end if;
                                                WHEN "0001" =>--xmit sign of
incoming data
                                                        data <= dig5;
                                                        if (timer = "11") and
(busyshiftreg(0) = '0') then--data accepted for xmit so proceed next state and wait till
xmitter is ready to accept new data
                                                                state_counter <=
state_counter + 1;
                                                                valid_data <= '0';
                                                                timer <= "00";
                                                        else--xmitter is not ready so wait
```

```
                                        state_counter           <=
state_counter;
                                        valid_data <= '1';
                                        if timer = "11" then
                                                timer <= timer;
                                        else
                                                timer <= timer + 1;
                                        end if;
                                end if;
                        WHEN "0010" =>--xmit first msb of
incoming data
                                data <= dig4;
                                if      (timer      =      "11")      and
(busyshiftreg(0) = '0') then
                                        state_counter           <=
state_counter + 1;
                                        valid_data <= '0';
                                        timer <= "00";
                                else
                                        state_counter           <=
state_counter;
                                        valid_data <= '1';
                                        if timer = "11" then
                                                timer <= timer;
                                        else
                                                timer <= timer + 1;
                                        end if;
                                end if;
                        WHEN "0011" =>--xmit second msb
                                data <= dig3;
                                if      (timer      =      "11")      and
(busyshiftreg(0) = '0') then
                                        state_counter           <=
state_counter + 1;
                                        valid_data <= '0';
                                        timer <= "00";
                                else
                                        state_counter           <=
state_counter;
                                        valid_data <= '1';
                                        if timer = "11" then
                                                timer <= timer;
                                        else
                                                timer <= timer + 1;
                                        end if;
                                end if;
                        WHEN "0100" =>--xmit third msb
                                data <= dig2;
```

```
                    if (timer = "11") and
(busyshiftreg(0) = '0') then
                        state_counter <=
state_counter + 1;
                        valid_data <= '0';
                        timer <= "00";
                    else
                        state_counter <=
state_counter;
                        valid_data <= '1';
                        if timer = "11" then
                            timer <= timer;
                        else
                            timer <= timer + 1;
                        end if;
                    end if;
                WHEN "0101" =>--xmit fourth msb
                    data <= dig1;
                    if (timer = "11") and
(busyshiftreg(0) = '0') then
                        state_counter <=
state_counter + 1;
                        valid_data <= '0';
                        timer <= "00";
                    else
                        state_counter <=
state_counter;
                        valid_data <= '1';
                        if timer = "11" then
                            timer <= timer;
                        else
                            timer <= timer + 1;
                        end if;
                    end if;
                WHEN "0110" =>--xmit lsb
                    data <= dig0;
                    if (timer = "11") and
(busyshiftreg(0) = '0') then
                        state_counter <=
state_counter + 1;
                        valid_data <= '0';
                        timer <= "00";
                    else
                        state_counter <=
state_counter;
                        valid_data <= '1';
                        if timer = "11" then
                            timer <= timer;
                        else
```

```
                                        timer <= timer + 1;
                                    end if;
                                end if;
                            WHEN "0111" =>--xmit CR
                                data <= X"0D";
                                if (timer = "11") and
(busyshiftreg(0) = '0') then
                                    state_counter <=
state_counter + 1;
                                    valid_data <= '0';
                                    timer <= "00";
                                else
                                    state_counter <=
state_counter;
                                    valid_data <= '1';
                                    if timer = "11" then
                                        timer <= timer;
                                    else
                                        timer <= timer + 1;
                                    end if;
                                end if;
                            WHEN "1000" =>--xmit LF so proceed
new data with new line
                                data <= X"0A";
                                if (timer = "11") and
(busyshiftreg(0) = '0') then
                                    state_counter <=
state_counter + 1;
                                    valid_data <= '0';
                                    timer <= "00";
                                else
                                    state_counter <=
state_counter;
                                    valid_data <= '1';
                                    if timer = "11" then
                                        timer <= timer;
                                    else
                                        timer <= timer + 1;
                                    end if;
                                end if;
                            WHEN OTHERS =>--start from sending
sign of new data (echo incoming data forever as there is new valid data)
                                    if data_valid = '1' then--
there is new valid data waiting to be echoed out
                                        state_counter <=
"0001";
                                    else--wait for new data
                                        state_counter <=
state_counter;
```

```
                                                    end if;
                                                    timer <= "00";
                                                    valid_data <= '0';
                                                    rom_counter             <=
rom_counter;
                                                    data <= data;
                                                END CASE;
                                        else--reset in order
                                                valid_data <= '0';
                                                rom_counter <= (others => '0');
                                                state_counter <= (others => '0');
                                                timer <= (others => '0');
                                                data <= X"61";
                                        end if;
                                end if;
                        end process;
-- sample incoming data process
                update_digit_proc: process(clk)
                        begin
                                if rising_edge(clk) then
                                        if reset_n = '1' then
                                                if data_valid = '1' then--there is valid data waiting
to be echoed
                                                        dig5 <= bin5;
                                                        dig4 <= bin4;
                                                        dig3 <= bin3;
                                                        dig2 <= bin2;
                                                        dig1 <= bin1;
                                                        dig0 <= bin0;
                                                else--wait
                                                        dig5 <= dig5;
                                                        dig4 <= dig4;
                                                        dig3 <= dig3;
                                                        dig2 <= dig2;
                                                        dig1 <= dig1;
                                                        dig0 <= dig0;
                                                end if;
                                        else--reset state (initial value +12000)
                                                dig5 <= X"2B";
                                                dig4 <= X"31";
                                                dig3 <= X"32";
                                                dig2 <= X"30";
                                                dig1 <= X"30";
                                                dig0 <= X"30";
                                        end if;
                                end if;
                        end process;

end Behavioral;
```

Ek C.6. AC97_ADAC LM4550 Ses Kartı Yönetim Alt Modülü

```
----------------------------------------------------------------------------------
-- Company:    KARABÜK ÜNİVERSİTESİ
-- Engineer:    Bilgehan ERKAL - Caner KİREMİTÇİ
--
-- Create Date:    11:26:16 04/28/2020
-- Design Name:
-- Module Name:    AC97_ADAC - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
-- Refer to LM4550 datasheet for details
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity AC97_ADAC is
Port(
                        AC97_int_SDO            : out STD_LOGIC;
                        AC97_int_SDI            : in STD_LOGIC;
                        AC97_int_BIT_CLK: in STD_LOGIC;
                        reset_n                         : in STD_LOGIC;
                        AC97_int_SYNC           : out STD_LOGIC;
                        DAC_L                           :                       in
STD_LOGIC_VECTOR(17 downto 0);
                        DAC_R                           :                       in
STD_LOGIC_VECTOR(17 downto 0);
                        ADC_L                           :                       out
STD_LOGIC_VECTOR(17 downto 0);
                        ADC_R                           :                       out
STD_LOGIC_VECTOR(17 downto 0);
                        clk_48                          : in STD_LOGIC
```

```
);
end AC97_ADAC;

architecture Behavioral of AC97_ADAC is

COMPONENT comrom
  PORT (
    clka : IN STD_LOGIC;
    addra : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
    douta : OUT STD_LOGIC_VECTOR(23 DOWNTO 0)
  );
END COMPONENT;

-- valid command present indicator signal
signal valbit                   : STD_LOGIC := '1';
-- output shift register for sending command and DAC data to soundcard
signal AC_97_out_sreg       : std_logic_vector(255 downto 0) := (others => '0');
-- input shift register for receiving command result data and ADC data
signal AC_97_in_sreg        : std_logic_vector(255 downto 0) := (others => '0');
-- command data signal
signal comrom_data          : std_logic_vector(23 downto 0) := (others => '0');
-- command data row signal used to address command rom
signal comrom_adr           : std_logic_vector(3 downto 0) := (others => '0');
-- DAC input registers
signal DAC_reg_L                : std_logic_vector(17 downto 0) := (others =>
'0');
signal DAC_reg_R                : std_logic_vector(17 downto 0) := (others =>
'0');
-- ADC output registers
signal ADC_reg_L                : std_logic_vector(17 downto 0) := (others =>
'0');
signal ADC_reg_R                : std_logic_vector(17 downto 0) := (others =>
'0');
-- preloaders used to resample module input before sending out to DAC
signal L_in                 : std_logic_vector(17 downto 0) := (others => '0');
signal R_in                 : std_logic_vector(17 downto 0) := (others => '0');
-- preregisters used to resample incoming ADC data before output by module
signal L_out                : std_logic_vector(17 downto 0) := (others => '0');
signal R_out                : std_logic_vector(17 downto 0) := (others => '0');
-- zero fill for output shift register
signal zero160                  : std_logic_vector(159 downto 0) := (others =>
'0');
-- synchronization signals
signal AC97_int_SYNC_reg : STD_LOGIC;
signal sync_count : std_logic_vector(7 downto 0) := (others => '0');

begin

AC97_int_SDO <= AC_97_out_sreg(255); -- DAC and command data output
```

```
AC97_int_SYNC <= AC97_int_SYNC_reg; -- soundcard sync input
-- connectors for ADC
ADC_L <= L_out;
ADC_R <= R_out;

-- Soundacard sync process
-- Generates a 48KHz sync clock necessary for soundcard using bit clock
AC97_SYNC_proc:process(AC97_int_BIT_CLK)
	begin
		if rising_edge(AC97_int_BIT_CLK) then
			if reset_n = '1' then -- end of reset
				sync_count <= sync_count + 1;
				if sync_count = 0 then
					AC97_int_SYNC_reg <= '1';-- start of sync impulse
				else
					if sync_count = 16 then -- end of sync impulse
						AC97_int_SYNC_reg <= '0';
					else
						AC97_int_SYNC_reg <= AC97_int_SYNC_reg;-- other times, conserve status
					end if;
				end if;
			else -- reset conditions
				sync_count <= (others => '0');
				AC97_int_SYNC_reg <= '0';
			end if;
		end if;
	end process;
-- command rom module
-- comrom holds initialization command data applied immediately after reset
Inst_comrom : comrom
  PORT MAP (
    clka => AC97_int_BIT_CLK,
    addra => comrom_adr,
    douta => comrom_data
  );

-- data and command data acquisition process
AC_97_ADAC_proc:process(AC97_int_BIT_CLK)
	begin
		if rising_edge(AC97_int_BIT_CLK) then
			if reset_n = '1' then -- end of reset
				-- serial data input from soundcard is shifted in to input shift register (ADC data)
				AC_97_in_sreg <= AC_97_in_sreg(254 downto 0) & AC97_int_SDI;
				-- start by the start of sync signal
				if sync_count = 1 then
```

```
                -- reload output shift register with fresh
command data from comrom and DAC data from DAC data loading registers
                -- Slot0, Slot1, Slot2, Slot3-4 DAC data
                AC_97_out_sreg <= '1' & valbit & valbit &
"11000" & X"00" & comrom_data(23 downto 16) & X"000" & comrom_data(15
downto 0) & X"0" & DAC_reg_L(17 downto 0) & "00" & DAC_reg_R(17 downto 0)
& "00" & zero160;
        else
                AC_97_out_sreg <= AC_97_out_sreg(254
downto 0) & '0';-- other times animate shift register and send data to soundcard
        end if;

        -- Time to withdraw ADC data coming from soundcard
and loaded to input shift register
        if sync_count = 2 then
                ADC_reg_L <= AC_97_in_sreg(199 downto
182);--slot3 data to ADC data input register Left channel
                ADC_reg_R <= AC_97_in_sreg(179 downto
162);--slot4 data to ADC data input register Right channel
        else -- other times conserve ADC data input registers
                ADC_reg_L <= ADC_reg_L;
                ADC_reg_R <= ADC_reg_R;
        end if;

        -- Time to load preloading output registers with fresh
data (DAC and command data)
        if sync_count = 3 then
                -- resampled module input loaded to preloading
output registers for DAC data
                DAC_reg_L <= L_in;
                DAC_reg_R <= R_in;

                -- if all the coomands listed in the command rom
is applied then enter wait state
                if comrom_adr = 5 then
                        comrom_adr <= comrom_adr; -- wait at
the last command
                else
                        comrom_adr <= comrom_adr + 1; --
proceed with the new command in the list
                end if;

                -- if last command is applied then mark repeating
last command as invalid by clearing the command valid bits
                if (comrom_adr = 4) or (comrom_adr = 5) then
                        valbit <= '0'; -- invalid command signaled
                else
                        valbit <= '1'; -- valid command signaled
                end if;
```

```
                        else -- other times conserve status, enter wait state
                                DAC_reg_L <= DAC_reg_L;
                                DAC_reg_R <= DAC_reg_R;
                                comrom_adr <= comrom_adr;
                                valbit <= valbit;
                        end if;
                else -- reset status
                        valbit <= '1';
                        AC_97_out_sreg <= (others => '0');
                        AC_97_in_sreg       <= (others => '0');
                        comrom_adr   <= (others => '0');
                        DAC_reg_L <= (others => '0');
                        DAC_reg_R <= (others => '0');
                        ADC_reg_L <= (others => '0');
                        ADC_reg_R <= (others => '0');
                        zero160 <= (others => '0');
                end if;
        end if;
    end process;

-- Process for resampling input and output of ADAC module at 48KHz
AC97_sample_proc:process(clk_48)
    begin
        if rising_edge(clk_48) then
            if reset_n = '1' then -- end of reset
                L_in <= DAC_L;
                R_in <= DAC_R;
                L_out <= ADC_reg_L;
                R_out <= ADC_reg_R;
            else -- reset in order, clear registers to inital values
                L_in <= (others => '0');
                R_in <= (others => '0');
                L_out <= (others => '0');
                R_out <= (others => '0');
            end if;
        end if;
    end process;

end Behavioral;
```

ÖZGEÇMİŞ

Caner KİREMİTCİ 1993 yılında Karabük'te doğdu; ilk ve orta öğrenimini aynı şehirde tamamladı. 2011 yılında Karabük Anadolu Öğretmen Lisesi'nden mezun oldu. 2011 yılında Eskişehir Osmangazi Üniversitesi Elektrik Elektronik Mühendisliği Bölümü'nde öğrenime başlayıp 2016 yılında mezun oldu. 2017 yılında Karabük Üniversitesi Fen Bilimleri Enstitüsü Elektrik Elektronik Mühendisliği Anabilim Dalı'nda başlamış olduğu yüksek lisans programını, 2021 yılında tamamladı.

ADRES BİLGİLERİ

Adres : Yenimahalle/ ANKARA
Tel : (543) 529 0710
E-posta: caner_kiremitci@hotmail.com