FATİH ÜNİVERSİTESİ

MÜHENDİSLİK FAKÜLTESİ

EEM 498

BİTİRME PROJESİ

Can EROL & Ejder BAŞTUĞ

USRP CİHAZLARIYLA DOSYA AKTARIMI ve MODÜLASYON TÜRLERİNİN PERFORMANS İNCELENMESİ

GRUP ÜYELERİ

Can EROL & Ejder BAŞTUĞ

Şubat 2009

ONAY SAYFASI

Bu nihai raporu okudum. Kanaatimce rapor içerik ve nitelik yönünden EEM 498 Bitirme Projesi dersi için rapor olarak uygunluğunu kabul ediyor ve onaylıyorum.

_____
Yard. Doç. Dr. Özgür ÖZDEMİR
Danışman

Sınavı Yapan Jüri Üyeleri

..... _____

..... _____

..... _____

..... _____

..... _____

Bu rapor Mühendislik Fakültesi tarafından konulan yazım kurallarına uygun olarak yazılmıştır.

_____
Yard. Doç Dr. Lokman ERZEN
Ders Koordinatörü

Şubat, 2009

İÇİNDEKİLER

Önsöz

Grubumuzun aldığı proje, USRP cihazlarıyla dosya aktarımı ve dosya atarımı sırasında kullanılan modülasyon çeşitlerinin performans incelenmesiydi. Bu proje özü itibariyle zaman ayrılması, özveriyle uğraşılması ve birçok alanda yoğun araştırma yapılması gereken bir projeydi.

Sonuç kısmında da bahsedeceğimiz gibi birçok açıdan grubumuz için oldukça verimli geçen bir tasarım aşamasıydı. Özellikle günümüzün, hızlı bir şekilde gelişen alanlarından haberleşmenin bir kolu olan kablosuz iletişim adına, bizim için oldukça öğretici bir safha oldu.

Umarız sizde tezimizi ve yaptığımız çalışmaları okurken, en az bizim kadar zevk alırsınız.

Can EROL & Ejder Baştuğ

2. Yazılım Tanımlı Radyo

Yazılım Tanımlı Radyo ya da bilinen ismiyle (Software Defined Radio) teknolojisi yavaş yavaş bildiğimiz, klasik anlamdaki radyo cihazlarının yerini almaya başlamaktadır. DSP teknolojilerinin gelişimiyle birlikte düşen maliyetler de göz önüne alınınca, önceleri yalnızca askeri alanlarda kullanılan bu teknoloji, artık herkesin cebine, evine, iş yerine girmeye başladı. Cep telefonlarımızı, evlerimizde, ofislerimizde internete bağlanmak üzere kullandığımız kablosuz ağ cihazlarını (Wi-Fi), Yazılım Tanımlı Radyo (SDR) teknolojisine borçluyuz.

Yazılım Tanımlı Radyo isminden de anlaşılabileceği gibi radyo sinyalini işlemek için dirençler, kondansatörler, geri besleme devreleri kullanmak yerine, yazılım kullanarak radyo fonksiyonlarının önemli bir kısmını yerine getiren bir radyo teknolojisidir. Sinyallerin gerek modülasyon gerekse demodülasyonu işlemleri için, yazılım kullanan radyo iletişim sistemi olarak tanımlanmaktadır. Bu teknoloji kullanılırken yazılımı işlemek amacıyla pc gibi genel amaçlı bir bilgisayar sistemi veya bu iş için özel olarak yapılmış ve yeniden programlanabilen özel işlemciler (ASIC -- tasarımcıların bir yapboz gibi birleştirip ayırabildikleri küçük çekirdek blokları) kullanılabilmektedir. Yazılım Tanımlı Radyo teknolojisinin en temel amacı; günümüzde ihtiyaç duyulabilen ve çeşitli amaçlar için kullanılan yeni radyo sinyal formlarını herhangi bir donanım değişikliği ve yüksek maliyetlere maruz kalmaksızın, yalnızca yazılım değişikliğine gidilerek kullanabilen radyo üretmektir.

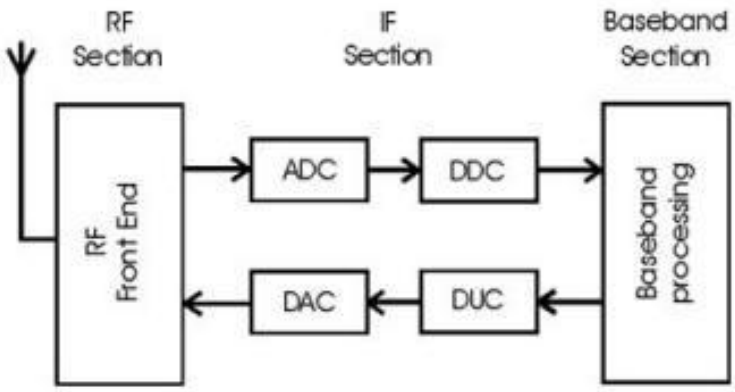
2.1 Yazılım Tanımlı Radyonun Avantajları

Yazılım tanımlı radyo her şeyden önce esnek (flexible) bir teknolojidir; Örneğin FM modülasyonu yapabilen bir radyo cihazının yalnızca yazılımını değiştirerek, kolayca AM yada SSB Modülasyonunu çalıştırabilecek bir hale getirebiliriz. Böyle bir değişikliği, klasik radyo sistemlerinde bu kadar kolay bir şekilde yapmayı ancak hayal edebiliriz.

Ucuz maliyetlidir; klasik anlamda RF dizaynı düşünüldüğünde, devrelerin karışıklığı ve gerçekten güvenilir, doğru çalışabilen devrelerin üretilmesi ciddi bir problemdir ve maliyeti artırıcı bir unsurdur. SDR sistemlerinde ise; DSP (Digital Signal Processing), FPGA (Field Programmable Gate-Array), GPP (General Purpose Processor) gibi teknolojiler kullanılarak, bu maliyetleri ve seri üretim aşamasında hata payını düşürülebilmektedir.

Yine kullanılan yazılımsal teknoloji nedeniyle kullanılan komponent listesi azalmaktadır. Bu sayede radyonun maliyetleri azalmakta ve azalan aktif pasif eleman sayısı, cihazların ömrünü de uzatmaktadır.

2.2 Basit Bir Yazılım Tanımlı Radyo Sistemi



Şekil 2.1 Basit Bir Yazılımsal Radyo Sistemi

Yukarıda SDR sistemlerinden bahsederken işin önemli bir kısmının yazılım tarafından gerçeleştirildiğinden bahsetmiştik, bu sistemlerin donanım tarafı ise şekilde görüleceği üzere, tipik olarak RF sinyallerini analog IF sinyallerine dönüştüren bir super-heterodin (supersonic heterodyn) front-end katı, analog IF sinyallerini sayısala ve sayısal olanları yeniden analoğa dönüştüren bir analog-dijital-analog (ADC) dönüştürücü katlarından ve tabii ki alıcı ya da verici bir antenden oluşmaktadır. Yine sistemin alıcı, verici ya da alıcı-verici olmasına göre de uygun mikrofon, ses yükselticisi (AF Amplifikatör), hoparlör gibi kat ve parçalar da kullanılmaktadır.

2.3 Radyo Amatörlüğü Açısından SDR Kullanımı

Yazılım tanımlı radyo tekonolojisi, askeri ve ticari alanlar ile birlikte artık Radyo Amatörleri tarafından da farkedilerek kullanılmaya başlanmıştır. SDR sayesinde tipik bir radyo amatörünün ihtiyacı olan Mors, SSB, FM, paket radyo ve SSTV uygulamalarının yanında COFDM (Coded Ortoghonal Frequency Divsion Multiplexing) gibi yeni, yüksek hızlarda, enterferansa daha az duyarlı veri taşımayı sağlayan teknolojiler de amatörlerin ilgi alanına girmeye başlamıştır.

Bir çok radyo amatörü bu sistemler üzerine çalışmakta, kullanmakta ya da gelişimini takip ederek, katkıda bulunmaktadır. Amatör radyocuların da geliştirilmesine

katkıda bulunduğu bir serbest yazılım projesi GNU Software Radio iyi bir örnek olarak verilebilir. GNU Software Radio, yazılım tanımlı radyonun yapması gerekenleri serbest yazılım (free software) mantığı içerisinde değerlendiren ve sıradan insanların da elektro manyetik spektrum ve radyo tekniğini kolayca anlayıp, uygulamalarını sağlamayı amaçlamaktadır. Uygulamada serbest dağıtılan bir yazılım, kişisel bilgisayara takılı olan ve analog, sayısal dönüştürücü olarak kullanılan bir ses kartı ve uygun bir front-end sağlayabilecek donanımdan oluşmaktadır.

2.4 USRP Nedir?

USRP'nin açılımı, Universal Software Radio Peripheral' yani Türçe'ye "Evrensel Yazılımsal Radyo Aygıtı" çevrilebilir. USRP, Ettus Research LLC tarafından üretilen, yazılım tabanlı radyo geliştirmeleri için kullanılan bir cihazdır.

USRP, üzerinde çalışan mühendislere oldukça hızlı ve rahat bir dizayn yapmayı, bu dizaynı rahatlıkla geliştirmeyi ve esnek yazılımsal radyo imkanlarını sağlamaktadır.

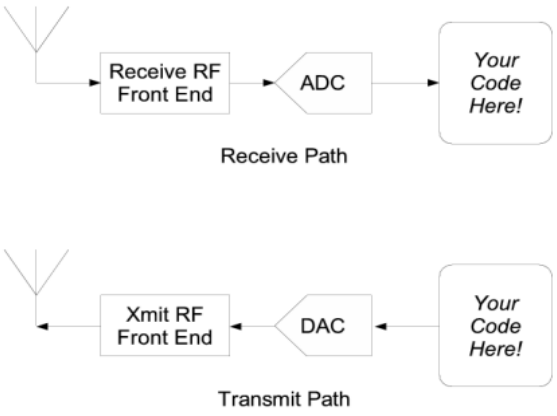
USRP cihazının içindeki elektronik kartta birçok modülasyonu sağlayabilecek modüller bulunmaktadır. USRP'de kullanılan antenin gücü ile doğru orantılı olarak belirli bir mesafede, desteklediği modülasyonlarla iletişim yapılabilmektedir. USRP birden fazla modülasyonu sağlayabilmesi ile merkezi radyo istasyonu gibi düşünülebilir.

USRP günlük hayatımızda pek fazla rastlayamadığımız; ancak takip edildiği taktirde üzerinde birçok çalışma yapılan ve gün geçtikçe geliştirilen bir cihazdır.

USRP cihazları birçok uygulamayla GNURadio tarafından desteklenmektedir. GNU Radio, internet sayfasında USRP kullanıcılarıyla Linux Python ve C diliyle yapılan birçok USRP yazılım kütüphanelerini ve örnek programlarını ücretsiz olarak paylaşmaktadır.

2.5 USRP ile Veri Aktarımındaki Safhalar

USRP'den bir veri göndermek için, göndericide bulunan program belirtilen veriyi alır, bir dijital analog çevirici ünitesinden geçirerek radyo dalga vericisine gönderir. Radyo dalga verici modülünde, istenilen modülasyona göre veriler güçlendirilerek antenden gönderir.



Şekil 2.2 USRP ile Dosya Aktarım Şeması

Göndericiden gelen bilgiler, alıcı anteni tarafından alınır ve alıcı modülünde demodülasyona tabi tutularak bir analog dijital çevirici ünitesinden geçirilir ve istenilen biçimde kullanılır.

İki USRP ile böyle bir iletişim için gönderici frekansı ile alıcı frekansı aynı olmalıdır. Gönderici ve alıcı USRP'lerin yazılımları üzerinde tanımlanan modülasyon biçimleri de aynı olmalıdır. Bu eşleşmeler olmadan, iki USRP cihazında iletişimin sağlıklı bir şekilde kurulması mümkün değildir.

3. Proje Geliştirme Ortamı

3.1 VMWare ve Linux Kurulumu

USRP cihazı için yapılan geliştirme araçları, çoğunlukla Linux ortamında tanımlıdır. Zaten GNURadio ve USRP'nin çıkış noktası, Linux ile özgür bir biçimde yazılımsal radio geliştirmektir. Haliyle USRP için kullanılacak kütüphaneler de Linux üzerinde tanımlıdır. Ayrıca Linux'un açık kaynak kodlu, serbest lisanslı ürün desteğinden dolayı, proje maliyeti azalmakta, barındırdığı binlerce hazır kütüphane desteği ile de proje verimliliğini arttırmaktadır.

Linux çekirdeği üzerine kurulu olan Linux Mint işletim sistemini, proje bilgisayarlarına ana işletim sistemi olarak kurmak yerine, Windows Vista üzerinde çalışabilen **VMWare Workstation** ile çalıştırmak esneklik açısından tercih edildi.

VMWare, gerçek bilgisayar (ev sahibi) üzerinde çalışanbilen, sanal bir makine yaratmaya yaramaktadır. Bu sanal makine, fiziksel olarak ev sahibinin kaynaklarını (işlemciyi, belleği vs.) kullanan, mantık olarak bağımsız bir bilgisayardır. Üzerinde tamamen farklı bir işletim sistemi çalıştırabilir.

İşletim sisteminin sanal olarak çalıştırılması elbette performans kaybını akla getirecektir, ancak bu performans kaybının %5 civarlarında olması proje için problem teşkil etmemektedir. Bu dezavantajına rağmen sanal bir işletim sisteminde çalışmak, projede kaynaklanabilecek problemleri, zaman kayıplarını minimuma indirmekte ve sistem çökmelerine karşı sınırsız deneme şansı vermektedir.

Öncelikli olarak VMWare Workstation'i çalıştırıyoruz. Linux ve Windows sürümleri mevcut. Elimizde Windows versiyonu olduğu için Windows altından kurulumu anlatacağız. VMWare download etmek isterseniz **http://www.vmware.com/** adresini kullanabilirsiniz.

VMWare' i ilk açıldığında, ekranda bir pencere görünür. Bu pencerede yapılması gereken, **New Virtual Machine**'e tıklayarak yeni bir Sanal Makine oluşturmak. Kullanıcıyı Windows altında program kurarken karşılaştığı, install ekranına benzeyen bir ekran karşılayacaktır. Bu ekranda, **Next** tuşuna basarak ilerlenir.

İlk ekranda sanal makine ayarları bulunur. Eğer server kurmak veya alttaki belirtilen eklentilerden yararlanmak istersenirse, **Custom** seçeneği seçilebilir. Linux Mint kurulumu yapılacağı için, **Typical** seçeneğini işaretleyip **Next** tuşuna basılır.

Kullanıcının karşısına çıkan ekranda hangi işletim sistemini kuracağı sorulur. Linux Mint listede bulunmadığı için, Linux altında en son kernel olan 2.6.0 i seçilmesi gerekir. Seçimi yaptıktan sonra, **Next** diyerek kuruluma devam edilir.

Bir sonraki adımda sanal makineye verilecek isim ve sanal makinenin nerede oluşturacağını belirleyen ekran ile karşılaşılır, Linux Mint kuracağımız ve daha sonra oluşturacağımız sanal makineler olabileceği için, isim olarak Linux Mint yazılır ve aşağıdaki kısma sanal makinenin kurulmak istenildiği yer belirtilir. Daha sonra **Next** denilir.

Sonraki adımımızda, ağ bağlantıları ile ilgili ayarları yapılır. Windows altından kurulum yaptığı için **NAT** seçilir ve **Next** butonuna basarak bir sonraki adıma geçilir.

Sonraki adımda, sanal makine için ne kadar yer ayıracağımız sorulur. **Allocate All Disk Space Now**'u yeterince yeriniz varsa işaretleyin. VMWare ayarları kaydeder ve bu ayarlara göre diski yaratmaya başlar. Bu adım için biraz zaman geçecektir.

Diskin oluşturulması bittiğinde, Linux Mint ile ilgili Sanal makine hazır durumdadır. Arkasindan **Edit Virtual Machine Settings** yazısına tıklanılır.

Sanal Makine ayarlarını açtığınızda, karşınızda birtakım seçenekler görürsünüz. Burada yapılması gereken **Memory** bölümünden, sanal makineye ne kadar **Ram**'in ayarlayacağını seçmek. Daha sonra sol menüden **Cd-Rom** sekmesine geçerek Linux Mint iso dosyasını göstermek gerekir. (Daha önceden Linux Mint'i indirip iso dosyanızın bulunduğunu varsayıyoruz) Burada yapmamız gereken ilk ayar, **Connect On Power On** seçeneğini seçerek, sanal makineyi başlattığımızda otomatik ise dosyasının çalıştırılmasını sağlamak. Daha sonra **Use ISO Image** seçeneğini seçerek, **Browser** denir. **ISO Image**'ini seçerek open tuşuna basıldığında, iso da ayarlanmış olur.

Sanal makinenin ayarlarından değiştirecek seçenekler bu kadar, **OK** diyerek ayarlar kapatılır. Ayaralar doğru yapıldıysa, makine kullanıma hazır durumdadır. **Start This Virtual Machine** seçeneğine tıklayarak sanal makine başlatılır.

Sanal makine başlatıldığında, ISO dosyası çalıştırıldığında başlamak üzere ayarladığı için, **Linux Mint CD**'sinden Boot olur gibi karşımıza gelir ve 10 sn sonra bu ekran kapanır, Enter'a basarak otomatik geçebilir. Linux Mint'i CD'den direk Hard Diskine aracısız kurmak isteyenler, bu adımdan sonrasını uygulayabilirler. Kalan adımlar için, Hard Diske kurmak ile sanal makineye kurmak benzerdir.

10 saniyelik süre bittikten sonra, Linux Mint'in yüklenmesi başlar. Yükleme tamamlandıktan sonra kullanıcıyı, Linux Mint cd'sinden canlı başlamış ekran karşılar. Bu ekrandan **Install**'u açarak, kurulum başlatılır.

İlk ekranda Linux Mint'i hangi dilde kuracağı seçilir. Daha sonra ileri diyerek, bir sonraki adıma geçilir. Bu ekranda tarih, saat ayarlamaları için, şehir belirtilmesi istenir. Şehir belirtilir ve bu adımda tamamlanır.

Bir sonraki ekranda klavye seçimi yapılır. Turkey Alt. Q Seçilirse Türkçe karakterler çalışmaz, bu yüzden Turkey International (With dead keys)'i seçilir. İstenirse, alttaki kutuda klavyedeki tuşlar kontrol edilebilir.

Dördüncü adımda disk için yer ayırılması istenir. Bu otomatik olarak veya elle ayarlanabilir. Projede sanal makine üzerinde kurulum yapılacağı için, diskin tamamının otomatik ayarlanması seçeneği seçilir. Elle düzenlemek isterseniz bu ekranda Elle diyerek ileri diyebilirsiniz.

Kimsiniz ekranı… Burada isim, Linux Mint için kullanıcı adı, şifre ve bilgisayarın ağ bağlantılarında nasıl görüneceğini ayarlanır. İleri diyerek, bu ekranda geçilir.

Bu adımda ayarlar kontrol edilebilir. Eğer ayarlar normal görünüyorsa, sağ alt köşeden yükle diyerek yüklemeyi başlatabilir. Kurulumun sonlarına doğru dil paketleri kurulur, istenirse bunları sağ alt köşeden atla diyerek atlanılabilir.

Kurulum tamamlandığında, yeni bir ekranla karşılaşılır. Bu ekranda **Şimdi Yeniden Başlat** veya **Çalışan Cd**'den devam et seçeneğini seçilebilir. Burada Linux Mint Cd'sini, CD-ROM'dan çıkarıp yeniden başlat seçilir.

VMWare' den kurulum yapılıp yeniden başlatıldığında, **Please Remove The Disc Close To Try** ekranı ile karşılaşılır. Sağ alt köseden VMWare'deki CD-ROM'a sağ tıklayıp **Disable** diyerek Enter' a basıldığında devam edebilir.

Bilgisayar yeniden başladığında, Bilgisayarı Kapat diyerek son CD-ROM ayarının yapılması gerekir. İlk olarak menüden **Quit** seçilir, sonraki adımda **Shutdown**'a basarak bilgisayar kapatılır.

Bilgisayar kapandıktan sonra, yine sanal makine seçeneklerini açıp; CD-Rom ayarlarına girilir ve daha önce işaretlediğimiz **Connect At Power On**'un işaretini kaldırıp CD'nin otomatik çalışması önlenir.

Yeniden Start Virtual Machine dediğinizde, CD'den kurulmuşsa Linux Mint'in Boot menusu ile karşılaşılır. Boot menusunu geçtikten sonra, daha önceden belirlediğimiz kullanıcı adı ve şifre ekranı karşımıza çıkar. Kullanıcı adı ve şifresi girilip Enter'a basıldığında, Linux Mint açılır.

Root Password sayfasında, bir Root hesabi yaratıp yaratılmayacağı sorulur. İstersenirse yaratılabilir, ancak bu önerilmeyen bir durumdur. Karar verildikten sonra **Forward** yapılır.

En son olarak **VMWare Tool** kurulur. VMWare Tool ciddi bir şekilde, performans artışı sağlar.

3.2 Yazılım Geliştirme Araçları ve Gerekli Kütüphanelerin Kurulumu

Python dilinde yazılım geliştireceğimiz için yazılım geliştirme araçları konusunda pek bir arayışa girmeye gerek yoktur. Çünkü Python betik bir dildir. Herhangi bir metin düzenleyicisi ile uygulama yapılabilir. Linux Mint'teki metin editörü Gedit bu iş için gayet yeterlidir. Lakin Gedit programının yeteneklerini biraz geliştirmek için; plugin eklemek, projenin verimliliği açısından iyi olacaktır. Aşağıdaki tavsiye edilen pluginler ve açıklamaları yer almakta.

- Auto Tab: Tab ayarlarını öğrenip, bunu Tab tuşuna basıldığında kullanır.
- Better Python Console: F5 tuşuna basıldığında yazılmış olan kodu, ayrı pencerede açılan bir Python Consol'unda çalıştıran bir eklentidir.
- External Tools: Gedit ile gelir.
- File Browser Pane: Gedit ile gelir.
- Indent Lines: Gedit ile gelir.
- Insert Date/Time: Gedit ile gelir.
- Modelines: Gedit ile gelir.
- Project Manager: Bir IDE'de vazgeçilmez özellik Proe yöneticisidir.
- Python Outline: Kodun taslağını çıkartır ve istenilen fonksiyon ya da sınıfa ulaşabilir.
- Snippets, Gedit ile gelir.

Tüm bu eklentileri .gnome2/gedit/pugins klasörü altına atın. Klasör yoksa oluşturun. Daha sonra Gedit'i kapatıp açın ve Tercihlerden gerekli Plugin'leri aktif edin.

3.3 USRP İle İlgili Kütüphane ve USRP Sürücülerinin Kurulumu

USRP kartının kullanımı için aşağıdaki paketler gereklidir. Bu paketler "synaptic", "dselect", veya"apt-get" ile kurulabilir. Aksi belirtilmedikçe aşağıdaki paketlerin en son sürümünü indirin. Synaptic üzerinden kurulum yapıyorsanız, rahat bir şekilde paket araması yapabilirsiniz. Bulduğunuz paketlere bağlı olan diğer paketler de otomatik kurulacak.

Derleme Araçları

- g++
- subversion
- make
- autoconf, automake, libtool
- sdcc ("universe" kategorisinden; 2.4 veya yenisi)
- guile (1.6 veya yenisi)
- ccache (mecburi değil)

Kütüphaneler

- python-dev
- FFTW 3.X (fftw3, fftw3-dev)
- cppunit (libcppunit and libcppunit-dev)
- Boost 1.35 (veya yenisi)
- libusb ve libusb-dev
- wxWidgets (wx-common) ve wxPython (python-wxgtk2.8)
- python-numpy (python-numpy-ext) (2007-Mayıs-28 den sornaki versiyonları)
- ALSA (alsa-base, libasound2 and libasound2-dev)
- Qt (libqt3-mt-dev)
- SDL (libsdl-dev)
- GSL GNU Scientific Librarby (libgsl0-dev >= 1.10)

Diğer kullanışlı araçlar

- swig
- doxygen
- usbview ("universe" seçeneğinden)
- octave ("universe" seçeneğinden)

4. Yapılan Araştırmalar

4.1Linux İşletim Sistemi

İşletim sistemi, bilgisayar donanımının doğrudan denetimi ve yönetiminden, temel sistem işlemlerinden ve uygulama programlarını çalıştırmaktan sorumlu olan sistem yazılımıdır.

İşletim sistemi, bütün diğer yazılımların belleğe, girdi/çıktı aygıtlarına ve kütük sistemine erişimini sağlar. Birden çok program aynı anda çalışıyorsa, işletim sistemi her programa yeterli sistem kaynağını ayırmaktan ve birbirleri ile çakışmamalarını sağlamaktan da sorumludur.

2005 yılı itibari ile, en yaygın olarak kullanılan işletim sistemleri iki ana grupta toplanabilir: Microsoft Windows grubu ve UNIX benzeri işletim sistemlerini içeren grup (Bu grup içinde pek çok Unix versiyonu, Linux ve Mac OS sayılabilir).

Projede kullanılacak bilgisayarda, Unix benzeri işletim sistemi olduğu için, Unix benzeri işletim sistemlerinin kullanımı ve genel yapısı hakkında bilgi sahibi olmak, işletim sistemine hakim olmayı sağlar.

Günümüzde tüm masaüstü bilgisayarlarda grafiksel kullanıcı arabirimi bulunmasına rağmen; bazı işlemleri yapabilmek için püf noktaları bilmek gerekir. Bu aşamada Linux kullanırken dizin işlemleri, dosya açma, güncelleme, çekirdek derleme, yükleme yapma, sürücü yükleme, konsol komutları ve bunların yapılışı incelendi.

4.2 Python Nesnel Programlama Dili

Python nesne yönelimli, yorumlanabilen, birimsel(modüler) ve etkileşimli bir programlama dilidir.

Girintilere dayalı basit sözdizimi, dilin öğrenilmesini ve akılda kalmasını kolaylaştırır. Bu da ona söz diziminin ayrıntıları ile vakit yitirmeden programlama yapılmaya başlanabilen bir dil olma özelliği kazandırır.

Modüler yapıyı, sınıf dizgesini (sistem) ve her türlü veri alanı girişini destekler. Hemen hemen her türlü platformda çalışabilir. (Unix , Linux, Mac, Windows, Amiga, Symbian Os bunlardan birkaçıdır). Python ile sistem programlama, kullanıcı arabirimi programlama, ağ programlama, uygulama ve veritabanı yazılımı programlama gibi birçok alanda yazılım geliştirebilirsiniz. Büyük yazılımların, hızlı bir şekilde prototiplerinin üretilmesi ve denenmesi gerektiği durumlarda da C ya da C++ gibi dillere tercih edilir.

4.3 OSI Referans Modeli ve TCP\IP Protokol Yapısı

Bilgisayar ağları kullanılmaya başlandığı ilk zamanlarda sadece aynı üreticinin ürettiği cihazlar birbirleriyle iletişim kurabiliyordu. Bu da şirketleri tüm cihazlarını sadece bir üreticiden almalarını zorunlu kılıyordu. 1970'lerin sonlarına doğru, ISO (International Organization for Standardization) tarafından, OSI (Open System Interconnection) modeli tanımlanarak bu kısıtlamanın önüne geçildi. Böylece farklı üreticilerden alınan cihazlar aynı ağ ortamında birbirleriyle haberleşebileceklerdi.

Projemiz için, bu model hakkında bilgi aldık. Bu modelin bize şöyle bir avantajı oldu: Yapacağımız noktadan noktaya kayıpsız veri iletişim işlemi için, nasıl bir protokol yapmamız gerektiği hakkında fikir sağladı.

4.4 Pythonda Dosya İşlemleri

Yapacağımız dosya transfer işleminin esnek olması için python dilinde dosya işlemleri için hangi komutları kullandığını irdeledik. Dosyayı parçalara ayırma, dosyada arama, aynı anda birden fazla dosyada işlem yapmayı inceledik. TCP/IP üzerinden basit

bir sunucu-istemci mimarisi nasıl olabilir ve performans artışı nasıl sağlanabilir gibi sorulara da cevap aradık.

4.5 Uygulama Geliştirme Ortamı ve Kullanılacak Kütüphaneler

Her ne kadar pythonda kodlama yaparken Gedit kullansak da, daha esnek bir arabirim arayışında iken (özellikle form tasarımı için) Eric4, BoaCsontructor programları incelendi. Ayrıca kodlama ortamında ürün verimini arttıran QT4, GTK kütüphaneleri de incelendi. Sonuç itibariyle programlama geliştirme ortamı olarak gedit, arabirim için GTK tercih edildi.

5. İncelenen Örnek Projeler ve Veri Aktarım Denemeleri

Daha önceden de bahsettiğimiz gibi, GNU Radio tarafından USRP kullanıcıları için internet ortamında, Python yazılım diliyle yapılmış birçok kütüphane ve örnek program mevcuttur. Bu kütüphaneleri ve örnek programları, USRP ile çalışmaya yeni başlayanların incelemesi; onlar için USRP'nin çalışma yapısının nasıl olduğunu öğrenmeleri açısından oldukça faydalı ve öğrenme sürecini hızlandıran bir işlev olacaktır.

5.1 Benchmark_RX ve Benchmark_TX Programları

Kaynak Kodlarının İncelenmesi

Bu iki program birlikte karşılıklı olarak çalıştırıldığında, hiçbir değişiklik yapılmadan, benchmark_tx.py programı içerisinden rastgele üretilen veriler gönderici USRP ile alıcı USRP'ye gönderilir.

Benchmark.tx programı içerisinde göze çarpan ilk olarak, ilgili kütüphanelerin çağrılmasıdır. Kütüphaneleri ikiye ayırabiliriz. Birinci tip kütüphane Python'un dahili kütüphaneleridir. Bu kütüphaneleri çağırırken "**import (kütüphanenin ismi)"** kullanılması yeterli olacaktır. Diğer tip kütüphane tipi ise Python'un dahili kütüphanesinde olmayan GNU Radio'nun USRP kütüphaneleridir. Bu kütüphaneleri çağırmak için "**from (bulunduğu yer) import (kütüphane ismi)"** girmek gerekmektedir.

Kütüphanelerden sonra göze çarpan "**class my_top_block(gr.top_block):"** kısmıdır. Burada nesnel programlamanın bir özelliği göze çarpmaktadır. Bir üst class oluşturulur ve buraya birden fazla kullanılan ve gereklilik arzeden fonksiyolar yazılır. Burada görüldüğü gibi **"def __init__(self, modulator, options):"** fonksiyonu tanımlanmıştır. Bu fonksiyonun görevi modülasyon tipi ile frekansını ayarlamak ve bağlantıyı kanalını aktif hale getirmektir.

Daha sonra göze çarpan "**main**" fonksiyonudur. Main fonksiyonu programın çalışması için gereken kısımdır. Main, fonksiyonu içerisinde **"def send_pkt(payload='', eof=False)"** fonksiyonu ile dosya yollanır. **"def rx_callback(ok, payload):"** fonksiyonuyla, gelen bilgiler ekrana yazdırılır. Ancak Benchmar_tx içindeki "**def rx_callback**(ok, payload)" fonksiyonu dışarıdan bilgi almadığı için sadece ekrana; yollanan paket sayısını basmaktadır. Bunu için programın en son kısmında bir "**while**" döngüsü tanımlanmıştır. Ekrana gönderilen payload verisi, iletişim sürdüğü sürece sürekli arttırılır ve ekrana gönderilir.

Benchmark_rx programı içerisinde tanımlanan **top block class' ının** içerisinde farklı olarak alıcı ayarları yapılmıştır. Arkasından gelen **main** fonksiyonu içinde sadece **rx_callback** fonksiyonu tanımlanmıştır. Bu fonksiyonun görevi, bilgi geldikçe sayıcıyı arttırmak ve bu sayıcının değerini payload değişkeni üzerinde ekrana basmak. Yine bunun içi "**while**" döngüsü tanımlanmıştır.

Bu iki program içerisinde tanımlanan parser.print fonksiyonları, kullanıcı tarafından bir hata yapıldığında, kullanıcıyı uyarmak için uyarıyı ekrana basmaktır.

Programın Kullanımı ve Örnek Veri Aktarımı

Benchmark_RX ve Benchmark_TX programlarını GNU.Radio Examples klasöründeki USRP ile ilgili dijital klasöründe bulunabilir. (Örnek Yol: /home/canerol/gnuradio-3.1.3/gnuradio-examples/python/digital/)

Benchmark_rx.py programı USRP'yi alıcı konumuna gelmesi için yapılan bir programdır. Bu dosyanın çalıştırılması:

- Doyanın bulunduğu klasörde boş bir yere Mouse ile sağ tıklanır.
- Karşımıza çıkan "Open in Terminal" seçeneğine gelinir. Bu seçenek klasörü komut sisteminde yönetmemize yardımcı olur.
- Linux de çekirdekle iletişimi olan programları çalıştırmak için gereken kök şifresi girilir: (Bu kısımla ilgili daha ayrıntılı bilgiler Linux kullanımında verilecektir.)

“canerol@3jd3r-mint:~/Desktop/gnuradio-3.1.3/gnuradio-examples/python/digital$ su

Password:”

- Kök şifresini girdiğimizde terminal üzerinde yönetici konumuna geçeriz ve burada programı çalıştırmak için:

  “3jd3r-mint digital # python benchmark_rx.py --freq=2400M”

Kodu girilir. Programın çalışması için tipini, adını ve programa kullanıcı tarafından girilmesi gereken modülasyon frekansı belirtilmelidir. Böylelikle program çalışmaya başlayacaktır.

Benchmark_tx.py programı USRP’yi gönderici konumuna getirir. Bu dosyanın çalıştırılması için yine yukarıdaki kodlara benzer yapı gereklidir ancak son basamakta programı çalıştırmak için:

“3jd3r-mint digital # python benchmark_tx.py --freq=2400M”

Kodu girilmelidir.

Bu programlarla USRP’lerden birini alıcı, diğerini ise verici pozisyonuna getirerek ilk aşamada 4kb’nin altında dosya aktarımını gerçekleştirdik. Ancak dosya karşı tarafa binary file olarak geçiyordu. Dosyayı açmak için ek bir program gerekmekteydi.

Bu aktarımı yapmak için paket yollama fonksiyonundaki **payload** değişkenine, göndericeğimiz veriyi okutup eşitledik. Veri ise transfer etmek istediğimiz dosyanın verilerinin seri dize haline getirilmiş biçimiydi. Benchmark.tx’deki bu kodlar:

```
nbytes = int(1e6 * options.megabytes)
n = 0
pktno = 0
pkt_size = int(options.size)

fileobj = open("ec.xt", mode='rb')
dosyaVerisi = fileobj.read()
fileobj.close()
veriArray = ["ec.txt", dosyaVerisi]
seriVeri = cPickle.dumps(veriArray)

while n < nbytes:
  if options.from_file is None:
    data = (pkt_size - 2) * chr(pktno & 0xff)
  else:
    data = source_file.read(pkt_size - 2)
    if data == '':
      break;
```

```
            payload = struct.pack('!H', pktno & 0xffff) + data
            send_pkt(seriVeri) #Bizim verilerimiz buradan yollanıyor.
            n += len(payload)
            sys.stderr.write('.')
            if options.discontinuous and pktno % 5 == 4:
              time.sleep(1)
            pktno += 1

        send_pkt(eof=True)
```

Özet olarak benchmark_tx programında ufak bir değişiklikle 4kb'nin altında dosyalar yollanabilir. 4 kb olmasının sebebi verici kapasitesinden kaynaklanmaktadır. Dolayısıyla daha büyük dosyaların yollanabilmesi için bir protokol ihtiyacı doğmaktadır.

5.2 Tunnel Programı

Kaynak Kodlarının İncelenmesi

Tunnel programı karşılıklı USRP'lerin iletişini sağlayan programdır. Ancak bu programın çalışabilmesi için daha önceden tanımlanmış bir protokolün çalıştırılmış olması gerekmektedir. Örneğin USRP ile bağlantımızı sağlayan USB çıkışını Ethernet çıkışı olarak tanıtıp, TCP/IP evrensel protokolünü kullanabiliriz.

Bu programı ana yapısı içerisinde değerlendirirsek **class mac' i** incelememiz yeterli olacaktır. Bu class içerisinde ilk göze çarpan "def __init __" fonksiyonudur. bu fonksiyon, programın protokolü kullanmasına olanak sağlar.

"**def set__flow_graph(self,fg)"** fonksiyonu, tunnel programının karşılıklı iletişimini sağlar. Bunun için verici ve alıcı ayarlarını yapar. Yine bu fonksiyon içinde bilgi yollamak için "**def send__pkt**" ve bilgi geldiğini gösteren "**def carrier_sensed(self)**" fonksiyonları da bulunur.

"**def phy_rx_callback(self, ok, payload)"** fonksiyonu, USRP gelen bilgileri ekrana ekrana basar.

"**def main_loop(self)"** fonksiyonu ise kullanıcının diğer USRP kullanıcısına bilgi yollamasını sağlar.

Daha önceden de bahsettiğimiz gibi Tunnel programı karşılık iletişime imkan bir programdı. Tunnel programının kullanılması için bir protokol tanımlanması gerekmekteydi.

Programın Kullanımı ve Örnek Veri Aktarımı

Bu programda **TCP\IP protokolü** kullanıldığından dolayı, kullanıcılardan birinin Server olması gerekir. Diğer kullanıcı ise bu Server ile iletişim kuracaktır. Burada USRP' yi Ethernet tanıtmak için konsoldan **ifconfig** seçeneği ile ayarlama yapılır. Server olan kullanıcı kendine bir Server numarası verir ve bir IP numarası alır. Diğer kullanıcıda kendisine bir IP numarası alır ve gireceği Server'ın numarasını belirler. Böylelikle sanki Ethernet kablosu varmış gibi, iletişim rahatlıkla USRP cihazlarıyla sağlanır. İletişim kurulduktan sonra, konsol ekranından girilen her bir bilgi diğer USRP kullanıcısına ulaşır. Bu protokolü Tunnel.py örnek kodu ile test ettik.

TCP\IP protokolü üzerinden yapılan iletişimde dikkat edilmesi gereken Tunnel.py çalıştırılırken çaprazlama frekans seçimidir. Bir USRP'nin alıcı frekansı öteki USRP'nin verici frekansı olmalıdır. Tunnel.py'nin çalıştırılması aşağıdaki gibidir:

Her iki konsoldan kök şifresi girilmelidir:

>>> su

>>> ****** (**Kök şifresi**)

Birinci Kullanıcının Tunnel.py Ayarları:

>>>python Tunnel.py **–-rx-freq 2400M –-tx-freq 2401M**

İkinci Kullanıcının Tunnel.py Ayarları:

>>>python Tunnel.py **–-rx-freq 2401M –-tx-freq 2400M**

Görüldüğü gibi bir çaprazlama söz konusudur.

Programın Konsol Uygulama Çıktısı:

```
3jd3r-mint canerol # ifconfig
eth1      Link encap:Ethernet  HWaddr 00:0c:29:67:bf:66
          inet addr:192.168.73.128  Bcast:192.168.73.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:7902 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6214 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:7301793 (6.9 MB)  TX bytes:576817 (563.2 KB)
          Interrupt:16 Base address:0x2024

gr0       Link encap:Ethernet  HWaddr 00:ff:ac:66:f8:dc
          inet addr:192.168.200.2  Bcast:192.168.200.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:39 errors:0 dropped:0 overruns:0 frame:0
          TX packets:27 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:4995 (4.8 KB)  TX bytes:3497 (3.4 KB)
```

lo Link encap:Local Loopback
inet addr:127.0.0.1 Mask:255.0.0.0
UP LOOPBACK RUNNING MTU:16436 Metric:1
RX packets:212 errors:0 dropped:0 overruns:0 frame:0
TX packets:212 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:14758 (14.4 KB) TX bytes:14758 (14.4 KB)

3jd3r-mint canerol # nc -v 192.168.200.1 1234
3jd3r-mint.local [192.168.200.1] 1234 (?) open
Selam (Mesaj Kısmı)

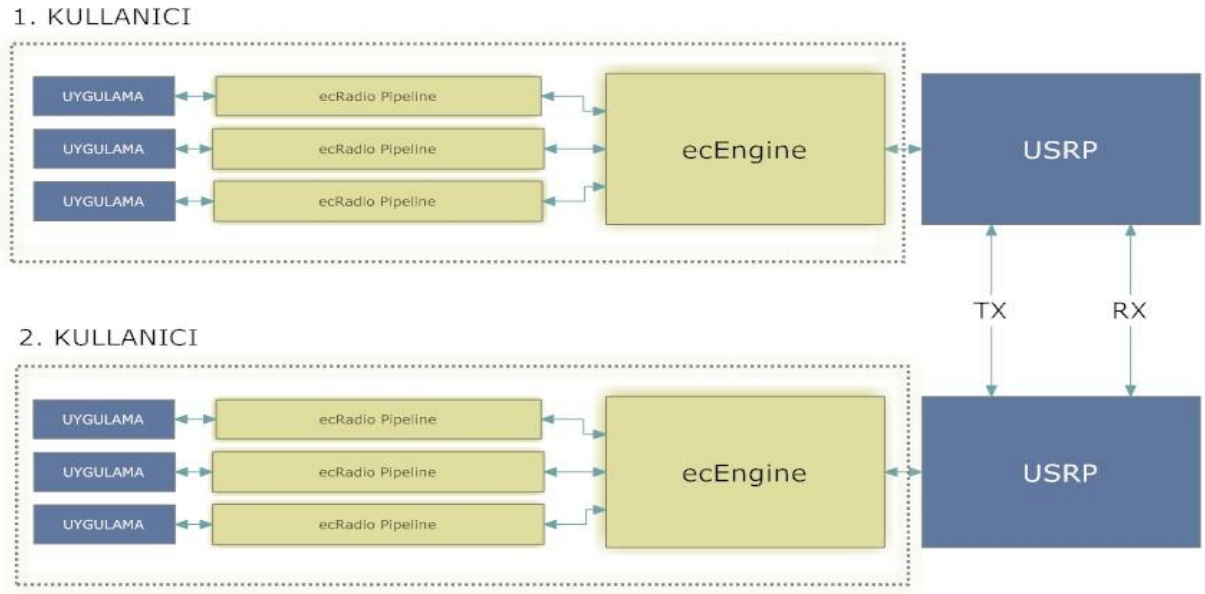
5.3 Elde Edilen Sonuçların Yorumlanması

Yaptığımız örnek programlardan da anlaşılacağı gibi, öncelikli olarak elimizde bir veri iletişim protokolü olması bir şarttır. Bu protokol karşılıklı olarak iletişimi sağlamalı ve gönderilecek dosyayı paketler halinde dize şekline dönüştürebilmelidir. Protokol üzerinde bir iş hattı olmalı, gönderilen verilerin doğruluğu kontrol edilmeli, eğer dosya alma sırasında bir paket kaybı olduysa, bu paket tekrar istenmelidir.

Buradan çıkartacağımız sonuçlardan bir tanesi de, dosya aktarımı için USRP'ler tek taraflı çalışamaz, çünkü bir USRP dosyayı yollarken; diğer USRP gelen verileri kontrol eder eksik paketler varsa bunları tekrar ister yani karşı tarafa bilgi yollar, gönderici USRP ise gelen hata mesajını göre işine devam eder yani gönderici olmasının yanında alıcıdırda. Yani dosya aktarımı için karşılıklı USRP'ler, hem alıcı hem de gönderici konumlarında olmalıdır.

6.Noktadan Noktaya Veri Aktarım Protokolü: ecRadio

6.1 ecRadio Çalışma Şeması

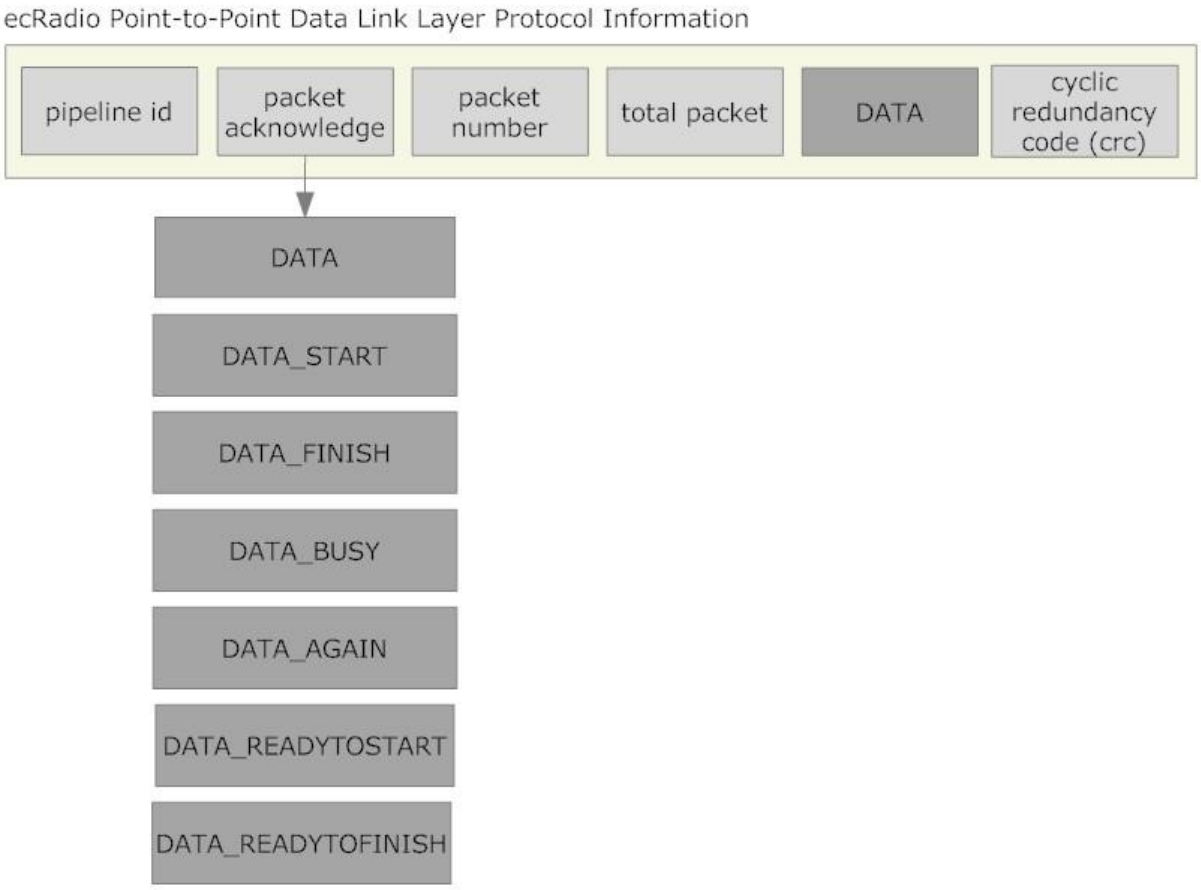


Şekil 6.1 ecRadio Çalışma Şeması

ecRadio projesinde, modülerlik önplandadır. Donanımsal katmandaki USRP'ye bağlanabileceği gibi başka bir donanımsal katman üzerinden de haberleşme yapabilir. OSI referans modeline göre tanımlı olduğu katman Data Link Layer'dir. Üst katmanlardan ya da uygulama tarafından gelen veri ecRadioPipeline sınıfı sayesinde parçalara ayrılıp tampon belleğe alınır ve ecRadioEngine'a iletir. Sayısız iş hattı desteği vardır. Bu sayede birden fazla uygulamadan gelen veri bu işhatları sayesinde ecRadioEngine' a iletilir. ecRadioEngine'da, ecRadio protokolüne uygun olarak paket takibi yapılır ve paketler donanımsal katmana gönderilir. Kısacası ecRadioEngine, veri iletim ve senkronizasyonun yapıldığı çekirdek sınıftır.

Projemizde örnek bir uygulama olarak ecRadioForm sınıfını tanımladık. ecRadioForm sınıfı tek bir işhattı kullanan, dosya aktarım ve kayıt arayüzü bulunan bir uygulamadır. ecRadioEngine ecRadioConnection ecRadioPipeline nesne örneklerini içinde barındırır.

6.2 ecRadio Protokolü Veri Paket Yapısı



Şekil 6.2 Paket Yapısı

ecRadio protokolünün paket yapısı yukarıda görüldüğü gibidir. Her paketin ilk elemanı pipeline ID (işhattı kimliği)'dir. İki nokta arası işhatları arasında veri iletişimi olabilmesi için her iki tarafta da aynı işhattı kimliği çağrılması gerekir. Aksi taktirde işhatları birbirleriyle haberleşme yapamaz.

Packet Acknowledge (paket durum mesajları)'nde veri başlangıcı, veri bitişi, veri, tekrar veri gönderme ve onay mesajları içerir. Bu sayede alıcı ve gönderici noktalar, verinin gidişatı hakkında bilgi sahibi olur.

Packet number (paket numarası) bölümünde o an gönderilen paketin numarası saklanır.

Total Packet (toplam paket)'te verinin toplam paket sayısı tanımlıdır. Bu bilgi önemlidir; çünkü veri göndermeye başlamadan önce karşı tarafın toplam paket sayısını bilmesi, ona göre tampon bellek miktarını ayarlaması gerekir.

Data (veri) kısmında ise gönderilecek veri saklanır.

CRC bölümünde, tüm paketin hata kodu saklanır. CRC sınamasını geçemeyen paket silinir çünkü kayıplıdır. Kayıplı paketler aktarım bittikten sonra, alıcı tarafından tekrar istenilir.

6.3 ecRadio Sınıf Tanımlamaları

ecRadioUsrpConnection Sınıfı:

Metodlar ve Özellikler:

```
class ecRadioUSRPConnection(gr.top_block,threading.Thread):
def __init__(self, mod_class, demod_class, rx_callback, options):
def send_pkt(self, payload='', eof=False):
def carrier_sensed(self):
```

def __init__(self, mod_class, demod_class, rx_callback, options):

Bu metotta USRP için kurulum seçenekleri ayarlanır. USRP için hangi modülasyon ve demodulasyon kullanılacağı, veri geldiğinde hangi metodu çağıracağı ve diğer usrp parametrelerinin ayarı burada yapılır.

def send_pkt(self, payload='', eof=False):

Payload olarak tanımlanan veriyi, USRP gönderim kanalı ile birlikte iletir.

def carrier_sensed(self):

Taşıyıcı frekansı algılandığında metod True(Doğru) döndürür.

ecRadioEngine Sınıfı:

Metodlar ve Özellikler:

```
class ecRadioEngine(threading.Thread):
def __init__ (self):
self.usrp        = None
self.pipelinePool = []
self.engineID = int(random.random() * 100)
```

```
self.engineIdleFlag = threading.Event()
self.packetSize = 1024 # in bytes
self.waitTime = 0.01 # in seconds
def makeConnection(self,connection):
def addPipeline(self,pipeline):
def encapsulateData(self, pipelineID, pipelineAcknowledge, packetNumber, packetTotal,
data):
def decapsulateData(self, encapsulatedData):
def onPayloadReceived(self, ok, usrpPayload):
def actionSendPayload(self, payload=None):
def run(self):
def stop(self):
```

def __init__ (self):

ecRadioEngine sınıfının kurulumu bu metot ile yapılır. ecRadio protokol tanımlamaları, başlangıç değerleri burada atanır.

self.usrp:

ecRadioEngine ın kullanacağı USRP aygıtı tanımlıdır. USRP aygıtı tanımlaması, ecRadioConnection sınıfının örneğinin çağrılması ile geçekleşir.

self.pipelinePool = []:

ecRadioPipeline sınıf örneklerinin tutulduğu dizidir. Böylece birden fazla işhattının yönetimi yapılabilir.

self.engineID = int(random.random() * 100):

ecRadioEngine için benzersiz bir kimlik ataması yapılır.

self.engineIdleFlag = threading.Event():

ecRadioEngine'ın meşgul yada boşta olduğu durumları algılayabilmesi için tanımlanan olay nesnesidir.

self.packetSize = 1024 # in bytes:

ecRadioEngine sınıfının bütün veriyi göndermek için parçalara ayırırken kullanacağı paket boyutudur.

self.waitTime = 0.01 # in seconds:

ecRadioEngine'da herhangi bir işlem yapıldıktan sonra beklenecek süreyi belirler.

def makeConnection(self,connection):

self.usrp özelliğinin atanması için gerekli metoddur. Herhangi bir yerden örneği çağırılan ecRadioUsrpConnection sınıfı bu şekilde ecRadioEngine'a bağlanır.

def addPipeline(self,pipeline):

ecRadioEngine a işhattı eklemek için kullanılan metottur. Herhangi bir yerde örneği çağırılan ecRadioPipeline bu şekilde ecRadioEngine a bağlanır.

def encapsulateData(self, pipelineID, pipelineAcknowledge, packetNumber, packetTotal, data):

Dizi halinde duran veriyi ecRadio protokolüne uygun bir şekilde kapsüller ve sonucu döndürür.

def decapsulateData(self, encapsulatedData):

USRP den gelen verinin ecRadio prokolüne uygun bir şekilde çözümlenmesi için kapsülden çıkarmak gerekir. Kapsüllü veriyi bu metod çözer ve dizi döndürür.

***def onPayloadReceived(self, ok, usrpPayload*)**:

USRP'den gelen verinin ecRadio protokolüne uygun bir şekilde işlendiği yerdir. Gelen veri istekleri, veri sonlanma istekleri, hatalı paketlerin takibi, crc kontrolü burada yapılır. Bu işlemler her bir iş hattı için tek tek yapılır.

def actionSendPayload(self, payload=None):

ecRadioUsrpConnection sınıfının send_pkt metodu ile aynı işlevdedir. Böyle bir metodun tekrar tanımlanma gereği USRP'den bağımsız çalışabilme durumudur.

def run(self):

ecRadioEngine da gelen herhangi bir veri olmadığı zaman işhatlarında bekleyen verileri protokole uygun bir şekilde gönderir.

def stop(self):

ecRadioEngine işparçacığının durdurulması sırasında yapılacak işlemler burada tanımlıdır.

ecRadioPipeline Sınıfı:

```
class ecRadioPipeline(threading.Thread):
def __init__(self, pipelineId=None):
self.pipelineID = pipelineId
self.txIdleFlag = threading.Event()
self.rxIdleFlag = threading.Event()
self.txBuffer = []
self.rxBuffer = []
self.txError = []
self.rxError = []
self.rxErrorFound = []
self.txPacketNo = 0
self.rxPacketNo = 0
self.txPacketTotal = 0
self.rxPacketTotal = 0
self.txIdleFlag.set()
self.rxIdleFlag.set()
self.packetSize = 1024
def run(self):
def sendData(self,txdata):
def stop(self):
```

def __init__(self, pipelineId=None):

ecRadioPipeline sınıfı çağrılma esnasında yapılacak işlemler burada tanımlıdır. Burada ilk değerler verilir ve bekleme durumuna geçilir.

self.pipelineID = pipelineId:

İşhattının kimliğine benzersiz bir numara atama yapar. Noktadan noktaya iletişimde her iki noktanında kimliğinin aynı olması gerekir. Aksi halde işhattı kurulmaz.

self.txIdleFlag = threading.Event():

İşhattının alım kanalının meşgul veya boş olduğunu belirten olay nesnesidir. Bu olay, işhattını kullanan uygulamaların işhattı yoğunluğunu öğrenmesi ve bekleme durumuna geçmesi için gereklidir.

self.rxIdleFlag = threading.Event():

İşhattının gönderim kanalının meşgul veya boş olduğunu belirten olay nesnesidir. Gönderim kanalı olay nesnesi ile aynı işleve sahiptir.

self.txBuffer = []:

Gönderilecek verinin tutulduğu tampon dizidir. Bu tampon dizide paketler belirlenen boyutlarda saklanır.

self.rxBuffer = []:

Alım esnasında gelen paketler burada tututulur.

self.txError = []:

Gönderim esnasında hangi paketlerin hatalı olduğu burada numaraları ile tutulur. Bunu yapabilmesi için karşı tarafın hatalı paketleri bildirmesi gerekir.

self.rxError = []:

Alım esnasında crc kontrolünden geçemeyen ya da protokole uygun olmayan paketlerin bilgisi burada tutulur.

self.rxErrorFound = []:

Karşı noktadan gelen verinin veri kesme isteğinden sonra hesaplanan eksik paketler burada tutulur. self.rxError[]'dan farklı olarak sadece eksik kalan paketleri tutar.

self.txPacketNo = 0:

Bir sonraki işlemde hangi paketin gönderileceği burada tutulur ve gönderildikten sonra paket numarası bir arttırılır.

self.rxPacketNo = 0:

Alım esnasında gelen paketin numarası burada tutulur.

***self.txPacketTotal = 0*:**

Gönderilecek verinin toplam paket sayısı tanımlıdır.

self.rxPacketTotal = 0:

Alınacak verinin toplam paket sayısı tanımlıdır.

self.packetSize = 1024:

Byte cinsinden paket boyutu tanımlıdır.

def run(self):

Bu metot şu aşamada herhangi bir işlev içermemekle birlikte geleceğe yönelik olarak tanımlanmıştır. İşhattının çalışma esnasında yapacağı işlemler burada tanımlanır.

def sendData(self,txdata):

İşhattındaki veriyi parçalara bölüp tampona yerleştirir ve ecRadioEngine'a iletmesi için uyarı gönderir.

def stop(self):

İşhattının görev sonlandırmasında yapacağı işlemler burada tanımlıdır.

ecRadioForm Sınıfı:

```
class ecRadioForm:
self.engine1
self.pipe1
self.usrp1
def __init__(self):
def actionQuitProgram(self, obj):
def action1SendData(self, button, data=None):
def action2SendData(self, button, data=None):
def main(self):
```

self.engine1:

ecRadioEngine örneğini barındırır.

self.pipe1:

ecRadioPipeline örneğini barındırır.

***self.usrp1*:**

ecRadioUsrpConnection örneğini barındırır.

def __init__(self):

Uygulamanın başlangıç aşamasında yapılacak işlemler ve değer atamaları tanımlıdır. ecRadioEngine, ecRadioPipeline ve ecRadioUsrpConnection nesnelerini çağırır ve ayarlamaları yapar. Ayrıca konsoldan ilgili usrp parametrelerinin alımı ve bunun USRP'ye bildirilmesi burada tanımlıdır.

def actionQuitProgram(self, obj):

Uygulama kapatıldıktan sonra yapılacak işlemler tanımlıdır. Bunlar arasında çağrılan sınıf örneklerinin silinmesi ve işletim sistemine kapatma isteği gönderme vardır.

def action1SendData(self, button, data=None):

Formdan okuduğu dosya ismini kendi dizininde arar ve bulduğu taktirde veriyi okur. Veriyi bütün olarak işhattına gönderir. İşhattı meşgul ise bekler.

def action2SendData(self, button, data=None):

İşhattında biten verinin kontrolünü yapar ve alım işleminin bittiğini algılaması halinde bu veriyi dosyaya kaydeder.

def main(self):

Uygulama arayüzü için gerekli Form tanımlamaları buradadır.

6.4 Modülasyon Türleri Hız ve Hata Testleri

GMPSK - DOSYA ALAN PC				
Dosya Boyutu	10KByte	100Kbyte	1200KByte	
Gönderilen Paket Sayısı	3	8	64	adet
Alınan Paket Sayısı	12	115	1369	adet
Rx Crc Hatasi	0	8	101	adet
Geçen Süre	0.28	4.97	53.79	saniye
Veri Aktarım Hızı	35.31	26.22	22.40	kb/sn

GMPSK - DOSYA GÖNDEREN PC				
Dosya Boyutu	10KByte	100Kbyte	1200KByte	
Gönderilen Paket Sayısı	13	125	1435	adet
Alınan Paket Sayısı	2	8	63	adet
CRC Hatası	0	0	0	adet
Geçen Süre	0.33	4995	53.85	saniye
Veri Aktarım Hızı	30.26	20.16	22.38	kb/sn

DQPSK - DOSYA ALAN PC				
Dosya Boyutu	10KByte	100Kbyte	1200KByte	adet
Gönderilen Paket Sayısı	7	36	19	adet
Alınan Paket Sayısı	23	200	1282	adet
CRC Hatası	8	65	54	saniye
Geçen Süre	0.82	7.96	49.12	kb/sn
Veri Aktarım Hızı	12.14	12.55	24.53	

DQPSK - DOSYA GÖNDEREN PC				
Dosya Boyutu	10KByte	100Kbyte	1200KByte	adet
Gönderilen Paket Sayısı	28	210	1289	adet
Alınan Paket Sayısı	7	36	19	adet
CRC Hatası	2	1	0	saniye
Geçen Süre	0.85	7.99	49.16	kb/sn
Veri Aktarım Hızı	11.75	12.51	24.51	

Şekil 6.3 Test Sonuçları

6.5 Modülasyon Türleri Performans Analizi

USRP cihazında varsayılan modülasyon türü GMPSK'dır. Bunun yanında değişik modülasyon türlerini 'cpm', 'd8psk', 'qam8', 'dbpsk', 'dqpsk' tipi modülasyon/demodülasyon desteği de bulunmaktadır. Dosya transferi denemelerimizde bu modülasyon türlerinin çoğunda problemle karşılaştık. Varsayılan modülasyon gmpsk ve dqpsk ile başarılı aktarım yapabilmemize rağmen diğerlerinde gönderim ya da alımda problemler oldu. Çalışmayan bu modülasyonların ecRadio protokolü gibi çift taraflı (alım/gönderim) iletişim yapan protokoller için eksik tanımlandığı kanısına vardık. GNURadio projesi sürekli gelişen bir proje. İlerleyen zamanlarda bu modülasyon tanımlamaların gelişeceği konusunda hemfikiriz.

Hata oranlarını ele aldığımızda dqpsk daha başarılı. Ayrıca dosya boyutu büyüdükçe gmsk ya göre daha hızlı aktarım yapabilmektedir. Bunun yanında gmsk ufak dosya boyutlarında hız açısından gözle görülür bir fark atmaktadır.

Yaptığımız bu hız ve hata testi şunu gösteriyor. Data/Link layer seviyesinde tanımlı bir protokol olan ecRadio için en iyi modülasyon/demodülasyon seçimi gmsk dır. Zaten düşük seviye bir protokolde büyük parçalı veri aktarımı sağlıklı değildir. Büyük parçalı veri aktarımından OSI üst katmanlarındaki protokoller sorumldur.

7. SONUÇ

USRP cihazlarıyla iletişimi ve dosya aktarımını gerçekleştir. Böyle bir cihazı programlamak için isteğin yanında en az orta seviyede programlama bilgisi ve network bilgisine sahip olması gerekir.

Yaptığımız dosya aktarım programına bakarsak, ilk aşamada iki cihaz birbirini görür ve daha sonra dosya aktarımını yapar. Dosya aktarımı içinse bir protokoltanımlanmalıdır. Bu protokolevrensel yapıdaki protokoller olabileceği gibi; USRP geliştiricisinin kendine has bir protokolüde olabilir. Bu protokolün amacı; dosyanın ulaşacağı yere kayıpsız ve hatalardan arındırılmış olarak gitmesidir.

Biz projemizi, hem evrensel yapıdaki TCP\IP protokolü hem de kendi yazdığımız ECRadio programı içindeki protokolle gerçekleştirdik.

8. Proje Maliyeti ile İlgili Bilgiler, Ayrılan Zaman ve Çalışma Şartlarımız

Cihaz Adı	Miktar	Birim Fiyat	Toplam Fiyat
USRP	2	650 Dolar	1300 Dolar
Bilgisayar	2	1000 Dolar	2000 Dolar
Toplam			3200 Dolar

Tablo 7.1 Maliyet Tablosu

Harcana Zaman = 4 (Saat/Hafta) x 10 (Hafta) = 40 Saat

	Pazartesi	Salı	Çarşamba	Perşembe	Cuma
8:00-9:00	EEM 498 / A Yrd.Doç.Dr. Lokman ERZEN			EEM 401 / A Prof.Dr. Sadık Kara E-310 MATE 348 / A Yrd.Doç.Dr. Nuran GÜZEL B-231	
9:00-10:00				EEM 419 / A Yrd.Doç.Dr. Özgür ÖZDEMİR E-216 MATE 348 / A Yrd.Doç.Dr. Nuran GÜZEL B-231	
10:00-11:00				EEM 419 / A Yrd.Doç.Dr. Özgür ÖZDEMİR E-216 MATE 348 / A Yrd.Doç.Dr. Nuran GÜZEL A-351	
11:00-12:00	EEM 423 / A Öğr.Gör.Dr. Salim BAŞKAN E-302	EEM 432 / A Doç.Dr. Erkan İMAL E-310	EEM 419 / A Yrd.Doç.Dr. Özgür ÖZDEMİR B-144		
12:00-13:00		EEM 401 / A Prof.Dr. Sadık Kara E-310			EEM 423 / LAB A Öğr.Gör.Dr. Salim BAŞKAN E-402
13:00-14:00		EEM 401 / A Prof.Dr. Sadık Kara E-310			EEM 423 / LAB A Öğr.Gör.Dr. Salim BAŞKAN E-402
14:00-15:00	EEM 432 / A Doç.Dr. Erkan İMAL E-216				
15:00-16:00	EEM 432 / A Doç.Dr. Erkan İMAL E-216		EEM 423 / A Öğr.Gör.Dr. Salim BAŞKAN E-310	EEM 432 / LAB A Doç.Dr. Erkan İMAL E-402	
16:00-17:00			EEM 423 / A Öğr.Gör.Dr. Salim BAŞKAN E-310	EEM 432 / LAB A Doç.Dr. Erkan İMAL E-402	

Tablo 7.2 Haftalık Ders Programı ve Boş Saatlerimiz

Projenin yapıldığı saatler esnek olup hafta içi verilen ödev, laboratuar raporları ve dinlenme saatleri dışında kalan zamanlarda uygun laboratuarda çalışmalar yapılmaktaydı. Ağırlıklı olarak mikroişlemciler laboratuarı kullanılmaktaydı. Çalışmalar kişisel dizüstü bilgisayarda yapıldığından ve kullanılan cihazlar taşınabilir olduğundan yer konusunda bir sıkıntı olmamaktaydı.

Projenin ilerleyişi hakkında bilgi vermek, tıkanılan noktalarda danışmak için Çarşamba günleri proje danışmanı Dr. Özgür ÖZDEMİR ile görüşülmekteydi. USRP ve GNURadio hakkında daha önce bilgi ve tecrübemiz olmadığı için görüşmeler son derece verimli geçmekteydi.

EK A: Kullanılan USRP'nin Özellikleri

Şekil EK A. Kullanılan USRP

- 4 adet analog-dijital çevirici, 85dB SFDR (AD9862).
- 4 adet digital-analog çevirici, 83dB SFDR (AD9862).
- Altera Cyclone EP1C12Q240C8 FPGA.
- Cypress EZ-USB FX2 High-speed USB 2.0 controller.
- 4 uzatma soketi (2 TX, 2 RX) (Ek aygıt takmak için)

EK B: Kullanılan Terim ile Kısaltmalar ve Bunların Anlamları

USRP: Universal Software Radio Peripheral

TCP/IP : Transmission Control Protocol\Internet Protocol

Kernel : İşletim Sistemi Çekirdeği

Root : Sistemdeki En Yetkili Kullanıcı

GMSK : Gaussian Minimum Shift Keying

EK C: Kullanılan Tablo ve Şekillerin Listesi

EK D: Benchmark_rx ve Benchmark_tx Programları

Benchmark_rx:

```
import random
import struct
import sys
# from current dir
from receive_path import receive_path
import fusb_options
#import os
#print os.getpid()
#raw_input('Attach and press enter: ')
class my_graph(gr.flow_graph):
    def __init__(self, demod_class, rx_callback, options):
        gr.flow_graph.__init__(self)
        self.rxpath = receive_path(self, demod_class, rx_callback, options)
# ////////////////////////////////////////////////////////////////////////
#                                     main
# ////////////////////////////////////////////////////////////////////////

global n_rcvd, n_right
def main():
    global n_rcvd, n_right
    n_rcvd = 0
    n_right = 0
    def rx_callback(ok, payload):
        global n_rcvd, n_right
        (pktno,) = struct.unpack('!H', payload[0:2])
        n_rcvd += 1
        if ok:
            n_right += 1
        print "ok = %5s  pktno = %4d  n_rcvd = %4d  n_right = %4d" % (
            ok, pktno, n_rcvd, n_right)
    demods = modulation_utils.type_1_demods()
    # Create Options Parser:
    parser = OptionParser (option_class=eng_option, conflict_handler="resolve")
    expert_grp = parser.add_option_group("Expert")
    parser.add_option("-m", "--modulation", type="choice", choices=demods.keys(),
                      default='gmsk',
                      help="Select modulation from: %s [default=%%default]"
                            % (', '.join(demods.keys()),))
    receive_path.add_options(parser, expert_grp)
    for mod in demods.values():
```

```
        mod.add_options(expert_grp)
    fusb_options.add_options(expert_grp)
    (options, args) = parser.parse_args ()
    if len(args) != 0:
        parser.print_help(sys.stderr)
        sys.exit(1)
    if options.rx_freq is None:
        sys.stderr.write("You must specify -f FREQ or --freq FREQ\n")
        parser.print_help(sys.stderr)
        sys.exit(1)
    # build the graph
    fg = my_graph(demods[options.modulation], rx_callback, options)
    r = gr.enable_realtime_scheduling()
    if r != gr.RT_OK:
        print "Warning: Failed to enable realtime scheduling."
    fg.start()        # start flow graph
    fg.wait()         # wait for it to finish
if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        pass
```

Benchmark_tx:

```
_dac_rate = 128e6
n2s = eng_notation.num_to_str
class tx_bpsk_block(gr.top_block):
    def __init__(self, options):
      gr.top_block.__init__(self, "tx_mpsk")
        self._transmitter = transmit_path(options.sps,
                                          options.excess_bw,
                                          options.amplitude)
        if_rate = options.rate*options.sps
        interp = int(_dac_rate/if_rate)
        print "Modulation:", n2s(options.rate), "bits/sec"
        print "TX IF rate:", n2s(if_rate), "samples/sec"
        print "USRP interpolation:", interp
        print "DAC amplitude:", options.amplitude
        self._setup_usrp(options.which,
                         interp,
                         options.tx_subdev_spec,
                         options.freq)
      self.connect(self._transmitter, self._usrp)
    def _setup_usrp(self, which, interp, subdev_spec, freq):
        self._usrp = usrp.sink_c(which=which, interp_rate=interp)
      if subdev_spec is None:
          subdev_spec = usrp.pick_tx_subdevice(self._usrp)
      self._usrp.set_mux(usrp.determine_tx_mux_value(self._usrp, subdev_spec))
```

```
        self._subdev = usrp.selected_subdev(self._usrp, subdev_spec)
        tr = usrp.tune(self._usrp, self._subdev.which(), self._subdev, freq)
         if not (tr):
             print "Failed to tune to center frequency!"
         else:
             print "Center frequency:", n2s(freq)
         gain = float(self._subdev.gain_range()[1]) # Max TX gain
        self._subdev.set_gain(gain)
        self._subdev.set_enable(True)
         print "TX d'board:", self._subdev.side_and_name()
def get_options():
    parser = OptionParser(option_class=eng_option)
    parser.add_option("-w", "--which", type="int", default=0,
                      help="select which USRP (0, 1, ...) default is %default",
                      metavar="NUM")
    parser.add_option("-T", "--tx-subdev-spec", type="subdev", default=None,
                      help="select USRP Tx side A or B (default=first one with a
daughterboard)")
    parser.add_option("-f", "--freq", type="eng_float", default=None,
                      help="set frequency to FREQ", metavar="FREQ")
    parser.add_option("-a", "--amplitude", type="eng_float", default=2000,
                      help="set Tx amplitude (0-32767) (default=%default)")
    parser.add_option("-r", "--rate", type="eng_float", default=250e3,
                      help="Select modulation symbol rate (default=%default)")
    parser.add_option("", "--sps", type="int", default=2,
                      help="Select samples per symbol (default=%default)")
    parser.add_option("", "--excess-bw", type="eng_float", default=0.35,
                      help="Select RRC excess bandwidth (default=%default)")
    (options, args) = parser.parse_args()
    if len(args) != 0:
        parser.print_help()
        sys.exit(1)
    if options.freq == None:
        print "Must supply frequency as -f or --freq"
        sys.exit(1)
    return (options, args)
if __name__ == "__main__":
    (options, args) = get_options()
    tb = tx_bpsk_block(options)
    try:
        tb.run()
    except KeyboardInterrupt
```

EK E: ECRadio Projesi Kaynak Kodları

```
from gnuradio import gr, gru, modulation_utils
from gnuradio import usrp
from gnuradio import eng_notation
from gnuradio.eng_option import eng_option
from optparse import OptionParser

import gobject
import threading
import random, time
import struct
import gtk
import sys
import os
import string
import array
import cPickle

from transmit_path import transmit_path
from receive_path import receive_path
import fusb_options

class ecRadioUSRPConnection(gr.top_block,threading.Thread):
    def __init__(self, mod_class, demod_class,
                 rx_callback, options):
        threading.Thread.__init__(self)
        gr.top_block.__init__(self)
        self.txpath = transmit_path(mod_class, options)
        self.rxpath = receive_path(demod_class, rx_callback, options)

       self.connect(self.txpath);
       self.connect(self.rxpath);

    def send_pkt(self, payload='', eof=False):
        return self.txpath.send_pkt(payload, eof)

    def carrier_sensed(self):
        return self.rxpath.carrier_sensed()

class ecRadioEngine(threading.Thread):
       def __init__ (self):
              threading.Thread.__init__(self)
              self.usrp    = None
              self.pipelinePool = []
              self.engineID = int(random.random() * 100)
              self.engineIdleFlag = threading.Event()
              self.engineIdleFlag.set()
              self.packetSize = 1024 # in bytes
```

```
                self.waitTime = 0.01 # in seconds
                self.DATA = 0x01
                self.DATA_START = 0x02
                self.DATA_FINISH = 0x03
                self.DATA_BUSY = 0x04
                self.DATA_AGAIN = 0x05
                self.DATA_READYTOSTART = 0x06
                self.DATA_READYTOFINISH = 0x07
                #Test variables
                self.txTotal = 0
                self.rxTotal = 0
                self.startTime = 0
                self.finishTime = 0
                self.crcError = 0

                #print "ecEngine" + str(self.engineID) +" created..."

        def makeConnection(self,connection):
                self.usrp = connection

        def addPipeline(self,pipeline):
                self.pipelinePool.append(pipeline)
                #print "Engine"+str(self.engineID) + ":Pipeline" +
str(pipeline.pipelineID) + " added..."

        def encapsulateData(self, pipelineID, pipelineAcknowledge, packetNumber,
packetTotal, data):
                dataArray =
[pipelineID,pipelineAcknowledge,packetNumber,packetTotal,data]
                encapsulatedData = cPickle.dumps(dataArray)
                return encapsulatedData

        def decapsulateData(self, encapsulatedData):
                try:
                        return cPickle.loads(encapsulatedData)
                except Exception,error:
                        #print "Engine"+str(self.engineID) + ": decapsulate error..."

                        return None

        def onPayloadReceived(self, ok, usrpPayload):
                #parsedMsg[0] = pipeline ID
                #parsedMsg[1] = packet acknowledge
                #parsedMsg[2] = packet no
                #parsedMsg[3] = packet total
                #parsedMsg[4] = data
                #cekilis = int(random.random() * 100)
                self.rxTotal += 1
                self.engineIdleFlag.clear()
                if ok:
```

```
                    parsedMsg = self.decapsulateData(usrpPayload)
                    if parsedMsg:
                          for pipe in self.pipelinePool:
                                if parsedMsg[0] == pipe.pipelineID:
                                      #Receiving request to start data
                                      if parsedMsg[1] == self.DATA_START:
                                            #print "Engine"+str(self.engineID) +
": DATA RX [START REQUEST]"
                                            #Lets check if our receiver is busy or
not
                                            if pipe.rxIdleFlag.isSet() or
pipe.rxPacketNo == 0:
                                                  #Our receiver is idle so lets
inform that we are ready and prepare rx buffer
                                                  #Test variables
                                                  self.txTotal = 0
                                                  self.rxTotal = 0
                                                  self.startTime = 0
                                                  self.finishTime = 0
                                                  self.crcError = 0
                                                  self.startTime = time.time()
                                                  pipe.rxError = []
                                                  pipe.rxBuffer = []

      pipe.rxError.extend(range(0,parsedMsg[3]))

      pipe.rxBuffer.extend(range(0,parsedMsg[3]))
                                                  for n in range(0,parsedMsg[3]):
                                                        pipe.rxBuffer[n] = None
                                                  pipe.rxPacketNo = 0
                                                  pipe.rxPacketTotal =
parsedMsg[3]
                                                  response =
self.encapsulateData(pipe.pipelineID,

                          self.DATA_READYTOSTART,

                          0,

                          pipe.rxPacketTotal,

                          None)

      self.actionSendPayload(response)
                                                  pipe.rxIdleFlag.clear()

                                            else:
                                                  #Our receiver is busy so lets
inform that we are busy
                                                  response =
self.encapsulateData(pipe.pipelineID,

                          self.DATA_BUSY,
```

```
                        0,

                        pipe.rxPacketTotal,

                        None)

        self.actionSendPayload(response)

                                    #Receiving data packet

                                    elif parsedMsg[1] == self.DATA:
                                        #print "Engine"+str(self.engineID)+":
DATA RX:"+str(parsedMsg[2])+"/"+str(parsedMsg[3])
                                        if pipe.rxPacketTotal == parsedMsg[3]:
                                            #print str(parsedMsg[2]) + "/"
+ str(parsedMsg[3])
                                            pipe.rxBuffer[parsedMsg[2]-1] =
parsedMsg[4]
                                            pipe.rxError[parsedMsg[2]-1] =
None
                                            pipe.rxPacketNo += 1
                                    #Receiving request to finish data
                                    elif parsedMsg[1] == self.DATA_FINISH:
                                        if pipe.rxPacketTotal == parsedMsg[3]:
                                            #print
"Engine"+str(self.engineID) + ": DATA RX [FINISH REQUEST]"
                                            #Calculating missed packets
                                            pipe.rxErrorFound = []
                                            for n in
range(0,pipe.rxPacketTotal):
                                                if pipe.rxError[n]:

        pipe.rxErrorFound.append(pipe.rxError[n] + 1)
                                            if pipe.rxErrorFound:
                                                error =
self.encapsulateData(pipe.pipelineID,

                        self.DATA_AGAIN,

                        pipe.rxPacketNo,

                        pipe.rxPacketTotal,

                        pipe.rxErrorFound[0:10])

        self.actionSendPayload(error)
                                                pipe.rxErrorFound =
pipe.rxErrorFound[10:]
                                            else:
                                                accepted =
self.encapsulateData(pipe.pipelineID,

                        self.DATA_READYTOFINISH,
```

```
                        pipe.rxPacketNo,

                        pipe.rxPacketTotal,

                        None)

      self.actionSendPayload(accepted)
                                                        self.finishTime =
time.time()
                                                        pipe.rxIdleFlag.set()
                                                        print "-----------------
-------------------------"
                                                        print "FILE TRANSFER
FINISHED"
                                                        print "Total TxPacket:
"+str(self.txTotal)
                                                        print "Total RxPacket:
"+str(self.rxTotal)
                                                        print "Total RxCRC
Error: "+str(self.crcError)
                                                        print "Elapsed Time:
"+str(self.finishTime - self.startTime)+" secs"
                                                        if pipe.txBuffer:
                                                              print "File
Transfer Speed TX: " + str(len(pipe.txBuffer)/((self.finishTime - self.startTime)))
+ " KByte/sec"
                                                        if pipe.rxBuffer:
                                                              print "File
Transfer Speed RX: " + str(len(pipe.rxBuffer)/((self.finishTime - self.startTime)))
+ " KByte/sec"
                                                  else:
                                                        #print "THERE ARE ERRORS SO WE
WILL REQUEST AGAIN"
                                                        #print
"Engine"+str(self.engineID) + ": DATA RX [ERROR]"
                                                        again =
self.encapsulateData(pipe.pipelineID,
                    self.DATA_AGAIN,
                    (pipe.rxPacketNo),
                    pipe.rxPacketTotal,
                    None)
                                                        self.actionSendPayload(again)
                                        #Receiving request to send data again
                                        elif parsedMsg[1] == self.DATA_AGAIN:
                                              #print "Engine"+str(self.engineID) +
": DATA RX [ERROR ACCEPTED]" + str(parsedMsg[2])
                                              if pipe.txPacketTotal == parsedMsg[3]:
                                                    pipe.txError = parsedMsg[4]
                                                    #sys.exit(0)
                                        #Receiving status of receiver that is ready
                                        elif parsedMsg[1] == self.DATA_READYTOSTART:
```

```
                                                #print "Engine"+str(self.engineID) +
": DATA RX [START ACCEPTED]"
                                                pipe.txPacketNo = 1
                                                pipe.txIdleFlag.clear()
                                          elif parsedMsg[1] ==
self.DATA_READYTOFINISH:
                                                #print "Engine"+str(self.engineID) +
": DATA RX [FINISH ACCEPTED]"
                                                pipe.txIdleFlag.set()
                                                self.finishTime = time.time()
                                                print "------------------------------
-----------"
                                                print "FILE TRANSFER FINISHED"
                                                print "Total TxPacket:
"+str(self.txTotal)
                                                print "Total RxPacket:
"+str(self.rxTotal)
                                                print "Total RxCRC Error:
"+str(self.crcError)
                                                print "Elapsed Time:
"+str(self.finishTime - self.startTime)+" secs"
                                                if pipe.txBuffer:
                                                      print "File Transfer Speed TX:
" + str(len(pipe.txBuffer)/((self.finishTime - self.startTime))) + " KByte/sec"
                                                if pipe.rxBuffer:
                                                      print "File Transfer Speed RX:
" + str(len(pipe.rxBuffer)/((self.finishTime - self.startTime))) + " KByte/sec"
                                          #Receiving status of receiver that is busy

                                          elif parsedMsg[1] == self.DATA_BUSY:
                                                print "Waiting"
                                          else:
                                                pass
                                                #print "Engine"+str(self.engineID) +
": Unhandled Protocol Acknowledge Flag"
                      else:
                            #print "Engine"+str(self.engineID) + ": Decapsulate
ERROR"
                            pass
              else:
                    #print "Engine"+str(self.engineID) + ": CRC ERROR"
                    self.crcError += 1
                    pass
              self.engineIdleFlag.set()
       def actionSendPayload(self, payload=None):
              #self.readyToSend.set()
              self.txTotal += 1
              self.usrp.send_pkt(payload)
       def run(self):
              while 1:
                    for pipe in self.pipelinePool:
                          self.engineIdleFlag.wait()
                          if not pipe.txIdleFlag.isSet():
```

```
                                    #Transmits errored packets again
                                    if pipe.txError:
                                          #print "Engine"+str(self.engineID)+": DATA
TX AGAIN:"+str(pipe.txError[0])+"/"+str(pipe.txPacketTotal)
                                          dataNext =
self.encapsulateData(pipe.pipelineID

      ,self.DATA

      ,pipe.txError[0]

      ,pipe.txPacketTotal

      ,pipe.txBuffer[pipe.txError[0]-1]);
                                          self.actionSendPayload(dataNext);
                                          del pipe.txError[0]
                                          #time.sleep(1)
                                    #Transmits and manages new data session
                                    elif pipe.txPacketNo == 0:
                                          #print "Engine"+str(self.engineID)+": DATA
TX [START REQUEST]"
                                          #Test variables
                                          self.txTotal = 0
                                          self.rxTotal = 0
                                          self.startTime = 0
                                          self.finishTime = 0
                                          self.crcError = 0
                                          self.startTime = time.time()

                                          dataStart =
self.encapsulateData(pipe.pipelineID

      ,self.DATA_START

      ,pipe.txPacketNo

      ,pipe.txPacketTotal

      ,None);
                                          self.actionSendPayload(dataStart)
                                          time.sleep(0.1)
                                    elif pipe.txPacketNo <= pipe.txPacketTotal:
                                          #print "Engine"+str(self.engineID)+": DATA
TX:"+str(pipe.txPacketNo)+"/"+str(pipe.txPacketTotal)
                                          dataNext =
self.encapsulateData(pipe.pipelineID

      ,self.DATA

      ,pipe.txPacketNo

      ,pipe.txPacketTotal

      ,pipe.txBuffer[pipe.txPacketNo - 1]);
```

```
                                        self.actionSendPayload(dataNext);
                                        pipe.txPacketNo += 1
                                elif pipe.txPacketNo > pipe.txPacketTotal:
                                        #print "Engine"+str(self.engineID)+": DATA
TX [FINISH REQUEST]"
                                        dataFinish =
self.encapsulateData(pipe.pipelineID

        ,self.DATA_FINISH

        ,pipe.txPacketNo

        ,pipe.txPacketTotal

        ,None);
                                        self.actionSendPayload(dataFinish)
                                        time.sleep(0.1)
                        else:
                                time.sleep(0.1)
                                pass

        def stop(self):
                print "Engine"+str(self.engineID)+" stopped.."
                self = None


class ecRadioPipeline(threading.Thread):
        def __init__(self, pipelineId=None):
                threading.Thread.__init__(self)
                if not pipelineId:
                        self.pipelineID = int(random.random() * 100000000);
                else:
                        self.pipelineID = pipelineId

                self.txIdleFlag = threading.Event()
                self.rxIdleFlag = threading.Event()
                self.txBuffer = []
                self.rxBuffer = []
                self.txError = []
                self.rxError = []
                self.rxErrorFound = []

                self.txPacketNo = 0
                self.rxPacketNo = 0
                self.txPacketTotal = 0
                self.rxPacketTotal = 0

                self.txIdleFlag.set()
                self.rxIdleFlag.set()

                self.packetSize = 1024
```

```
            print "Pipeline"+str(self.pipelineID)+" created..."

      def run(self):
            pass

      def sendData(self,txdata):
            #self.txIdleFlag.wait()
            #Checking pipeline tx buffer is busy or not
            if self.txIdleFlag.isSet():
                  #Calculates total packet number
                  totalPacket = int(len(txdata) / self.packetSize) + 1

                  #Clears tx buffers
                  self.txError = []
                  self.txBuffer = []

                  #Splits data
                  self.txBuffer.extend(range(0,totalPacket))
                  for n in range(0,totalPacket):
                        self.txBuffer[n] =
txdata[n*self.packetSize:n*self.packetSize + self.packetSize]

                  #Tx buffer information
                  self.txPacketNo = 0
                  self.txPacketTotal = totalPacket
                  #Makes pipeline tx status to busy
                  self.txIdleFlag.clear()
            else:
                  print "Pipeline"+str(self.pipelineID)+" TX busy..."

      def getData(self):
            self.rxIdleFlag.wait()
            returnData = ""
            if self.rxPacketTotal != 0:
                  #Rebuilds data
                  for n in range(0,self.rxPacketTotal):
                        if n == 0:
                              returnData = self.rxBuffer[0]
                        else:
                              returnData += self.rxBuffer[n]
                  self.rxError = []
                  self.rxErrorFound = []
                  self.rxBuffer = []
                  self.rxPacketNo = 0
                  self.rxPacketTotal = 0
                  return returnData
            else:
                  return None
```

```
	def stop(self):
		print "Pipeline"+str(self.pipelineID)+" stopped.."
		self = None

class ecRadioForm:
	def __init__(self):
		gtk.gdk.threads_init()
		mods = modulation_utils.type_1_mods()
		demods = modulation_utils.type_1_demods()

		parser = OptionParser (option_class=eng_option,
conflict_handler="resolve")
		expert_grp = parser.add_option_group("Expert")

		parser.add_option("-m", "--modulation", type="choice",
choices=mods.keys(),
                                  default='gmsk',
                                  help="Select modulation from: %s
[default=%%default]"
                                        % (', '.join(mods.keys()),))

		parser.add_option("-v","--verbose", action="store_true",
default=False)
		expert_grp.add_option("-c", "--carrier-threshold", type="eng_float",
default=30,
                                      help="set carrier detect threshold (dB)
[default=%default]")
		expert_grp.add_option("","--tun-device-filename",
default="/dev/net/tun",
                                      help="path to tun device file
[default=%default]")

		transmit_path.add_options(parser, expert_grp)
		receive_path.add_options(parser, expert_grp)

		for mod in mods.values():
			mod.add_options(expert_grp)

		for demod in demods.values():
			demod.add_options(expert_grp)

		fusb_options.add_options(expert_grp)

		(options, args) = parser.parse_args ()
		if len(args) != 0:
			parser.print_help(sys.stderr)
			sys.exit(1)

		if options.rx_freq is None or options.tx_freq is None:
			sys.stderr.write("You must specify -f FREQ or --freq FREQ\n")
```

```
                parser.print_help(sys.stderr)
                sys.exit(1)
        #BURDAN ASAGISI ENTEGRE EDILECEK
        # Attempt to enable realtime scheduling
        r = gr.enable_realtime_scheduling()
        if r == gr.RT_OK:
                realtime = True
        else:
                realtime = False
                print "Note: failed to enable realtime scheduling"


        # If the user hasn't set the fusb_* parameters on the command line,
        # pick some values that will reduce latency.

        if options.fusb_block_size == 0 and options.fusb_nblocks == 0:
                if realtime:                        # be more aggressive
                        options.fusb_block_size = gr.prefs().get_long('fusb',
'rt_block_size', 1024)
                        options.fusb_nblocks    = gr.prefs().get_long('fusb',
'rt_nblocks', 16)
                else:
                        options.fusb_block_size = gr.prefs().get_long('fusb',
'block_size', 4096)
                        options.fusb_nblocks    = gr.prefs().get_long('fusb',
'nblocks', 16)

        #print "fusb_block_size =", options.fusb_block_size
        #print "fusb_nblocks    =", options.fusb_nblocks

        # INITIALIZE COMPONENTS
        self.engine1 = ecRadioEngine()
        self.pipe1 = ecRadioPipeline(1)
        self.usrp1 = ecRadioUSRPConnection(mods[options.modulation],
                                demods[options.modulation],
                                self.engine1.onPayloadReceived,
                                options)

        self.engine1.makeConnection(self.usrp1)
        self.engine1.addPipeline(self.pipe1)

        if self.usrp1.txpath.bitrate() != self.usrp1.rxpath.bitrate():
                print "WARNING: Transmit bitrate = %sb/sec, Receive bitrate =
%sb/sec" % (
                        eng_notation.num_to_str(self.usrp1.txpath.bitrate()),
                        eng_notation.num_to_str(self.usrp1.rxpath.bitrate()))

        print "modulation:     %s"   % (options.modulation,)
        print "freq:           %s"      %
(eng_notation.num_to_str(options.tx_freq))
```

```
            print "bitrate:        %sb/sec" %
(eng_notation.num_to_str(self.usrp1.txpath.bitrate()),)
            print "samples/symbol: %3d" %
(self.usrp1.txpath.samples_per_symbol(),)
            #print "interp:         %3d" % (usrp.txpath.interp(),)
            #print "decim:          %3d" % (usrp.rxpath.decim(),)

            self.usrp1.rxpath.set_carrier_threshold(options.carrier_threshold)
            print "Carrier sense threshold:", options.carrier_threshold, "dB"
            self.engine1.start()
            self.pipe1.start()
            self.usrp1.start()

      def actionQuitProgram(self, obj):
            self.engine1.stop()
            self.pipe1.stop()
            self.usrp1.stop()
            #self.usrp1.wait()
            gtk.main_quit()
            sys.exit()

      def action1SendData(self, button, data=None):
            print "Testing 10kb..."
            fileobj = open("test1.pdf", mode='rb')
            dosyaVerisi = fileobj.read()
            fileobj.close()
            #veriArray = ["djshadow.mp3", dosyaVerisi]
            #seriVeri = cPickle.dumps(veriArray)
            self.pipe1.sendData(dosyaVerisi)
            self.pipe1.txIdleFlag.wait()
            self.pipe1.sendData(dosyaVerisi)
            self.pipe1.txIdleFlag.wait()
            self.pipe1.sendData(dosyaVerisi)
            self.pipe1.txIdleFlag.wait()

            print "Testing 100kb..."
            fileobj = open("test2.pdf", mode='rb')
            dosyaVerisi = fileobj.read()
            fileobj.close()
            #veriArray = ["djshadow.mp3", dosyaVerisi]
            #seriVeri = cPickle.dumps(veriArray)
            self.pipe1.sendData(dosyaVerisi)
            self.pipe1.txIdleFlag.wait()
            self.pipe1.sendData(dosyaVerisi)
            self.pipe1.txIdleFlag.wait()
            self.pipe1.sendData(dosyaVerisi)
            self.pipe1.txIdleFlag.wait()

            print "Testing 1200kb..."
            fileobj = open("test3.pdf", mode='rb')
```

```
                dosyaVerisi = fileobj.read()
                fileobj.close()
                #veriArray = ["djshadow.mp3", dosyaVerisi]
                #seriVeri = cPickle.dumps(veriArray)
                self.pipe1.sendData(dosyaVerisi)
                self.pipe1.txIdleFlag.wait()
                self.pipe1.sendData(dosyaVerisi)
                self.pipe1.txIdleFlag.wait()
                self.pipe1.sendData(dosyaVerisi)
                self.pipe1.txIdleFlag.wait()
                #self.pipe1.sendData(seriVeri)
        def action2SendData(self, button, data=None):
                dosyaGelen = self.pipe1.getData()
                fileobj2 = open("_"+self.entry.get_text(), mode='wb')
                fileobj2.write(dosyaGelen)
                fileobj2.close()
        def main(self):
                window = gtk.Window()
                vbox = gtk.VBox()
                self.entry = gtk.Entry()
                button1 = gtk.Button("SEND DATA ->" + str(self.engine1.engineID))
                button2 = gtk.Button("SAVE DATA <-" + str(self.engine1.engineID))
                vbox.pack_start(self.entry, False, True)
                vbox.pack_start(button1, True, True)
                vbox.pack_start(button2, True, True)
                window.add(vbox)
                window.show_all()
                button1.connect("clicked", self.action1SendData)
                button2.connect("clicked", self.action2SendData)
                window.connect('destroy', self.actionQuitProgram)

                gtk.gdk.threads_enter()
                gtk.main()
                gtk.gdk.threads_leave()

if __name__ == '__main__':
        ec = ecRadioForm()
        ec.main()
```

EK F: Kaynakçalar

www.python.org/

www.vmware.com

www.linuxmint.com/

www.ettus.com/

www.epo.usra.edu/usrp/

www.gnuradio.org/trac/

education.nasa.gov/usrp

www.usrp.net/

projects.gnome.org/gedit/

en.wikipedia.org/wiki/Gedit

www.belgeler.org/uygulamalar/python-tutorial_basliyoruz.html

http://en.wikipedia.org/wiki/Software_radio

http://www.tomshardware.com.tr/business/20021015/index.html

http://searchnetworking.techtarget.com/....

http://www.radio-electronics.com/info/receivers/software_defined_radio/sdr.php

http://www.cognitiveradio.wireless.vt.edu/radiohas.html

http://drm.sourceforge.net/

http://www.arrl.org/tis/info/sdr.html

http://www.gnu.org/software/gnuradio/

http://www.flex-radio.com/shacks.htm