T.C.
FIRAT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR AĞLARI İÇİN SALDIRI TESPİT SİSTEMİ TASARIMLARI VE FPGA ORTAMINDA GERÇEKLEŞTİRİLMESİ

DOKTORA TEZİ

Yük. Müh. Taner TUNCER

Anabilim Dalı: Elektrik-Elektronik Mühendisliği

Programı : Elektrik Makinaları

ŞUBAT-2010

T.C.
FIRAT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR AĞLARI İÇİN SALDIRI TESPİT SİSTEMİ TASARIMLARI VE FPGA ORTAMINDA GERÇEKLEŞTİRİLMESİ

DOKTORA TEZİ

Yük. Müh. Taner TUNCER

(03113208)

Tezin Enstitüye Verildiği Tarih : 2 Şubat 2010

Tezin Savunulduğu Tarih : 22 Şubat 2010

Tez Danışmanı: Doç.Dr Yetkin TATAR (FIRAT.Ü)

Diğer Jüri Üyeleri : Prof. Dr. Yakup DEMİR(FIRAT.Ü)

Doç.Dr. Ali KARCI (İNÖNÜ Ü.)

Doç.Dr. Mehmet KAYA (FIRAT Ü.)

Yrd.Doç.Dr. Burhan ERGEN(FIRAT.Ü)

ŞUBAT-2010

ÖNSÖZ

Bu tezde, gerçek zamanda çalışabilen bir Saldırı Tespit Sisteminin (STS) FPGA ortamında tasarımı ve uygulaması gerçekleştirilmiştir. Yaptığım bu tez çalışmasının, konuyla ilgilenen gerek lisans, gerek lisansüstü öğrencilerine ve gerekse bu konuda çalışmak isteyen akademisyenlere bir rehber olacağını düşünüyor ve umuyorum.

Sistemin gerçek zamanda gerçekleştirilmesi ve tezin tamamlanması için gerekli Donanım ve Yazılım ürünlerinin temin edilmesinde maddi desteklerinden dolayı Türkiye Bilimsel ve Teknolojik Araştırma Kurumuna ve Fırat Üniversitesi Bilimsel Araştırma Projeleri Birimine Teşekkür ederim.

Akademik hayatım boyunca bilgileriyle beni yönlendiren, maddi ve manevi desteğini benden esirgemeyen danışman hocam, Doç.Dr. Yetkin TATAR' a teşekkür ederim.

Her türlü zorluklarla mücadele ederek bu günlere gelmemde emeği geçen Annem ve Babama en derin saygılarımı ve şükranlarını sunarım. Ayrıca tez süresi boyunca desteklerini benden esirgemeyen Eşim Seda ve Kızım Zeynep'e teşekkür ederim.

Taner TUNCER

ELAZIĞ-2010

İÇİNDEKİLER

Sayfa No

Sayfa No

ÖZET

Bilgisayar ağlarının güvenliğinin sağlanması için araştırmacılar güvenlik duvarları, antivirüs programları ve Saldırı Tespit Sistemleri (STS) gibi hem yazılım hem donanım tabanlı ürünler geliştirmektedirler. STS'ler ağ üzerindeki paketleri, değişik kriterlere göre değerlendirip saldırı paketi olup olmadığına karar veren sınıflandırma sistemleridir. Özellikle yoğun trafiğe sahip ağlarda STS'lerin çok hızlı çalışması ve geciktirme kaynağı olmaması çok önemlidir. Bundan dolayı donanım tabanlı STS'lerin geliştirilmesi için yapılan araştırmalar artarak devam etmektedir. STS'ler ile ilgili araştırmalar iki yönde sürdürülmektedir. Bunlar paketlerin daha hızlı ve daha doğru bir şekilde sınıflandırılabilmesi için, farklı sınıflandırma algoritmalarının kullanılması, geliştirilmesi ve bu algoritmaların üzerinde çalışacağı donanımsal yapının geliştirilmesidir.

Bu tezde, Bilgisayar ağlarında çevrim içi çalışabilen bir STS'nin FPGA ortamında gerçekleştirilmesi ve uygulaması amaçlanmıştır. Bunun için; Yapay Sinir Ağları, Naive Bayesian ve Karar Ağacı gibi üç farklı sınıflandırma algoritması kullanılmıştır. Uygulama için ilk olarak önceden belirlenmiş eğitim verileri ile her üç algoritma eğitilmiş olup, test verileri kullanılarak çevrim dışı çalışmada algoritmaların başarımları gözlemlenmiştir.

Tasarımı yapılan STS'nin çevrim içi çalışması için Altera Cyclone III tabanlı EPC3C40F484C7N FPGA platformunda, donanımsal bir SOPC sistem geliştirilmiştir. SOPC sistem, bilgisayar ağı ile iletişim sağlayan MAC modülü ve bir soft işlemciye sahiptir. STS'nin oluşturulması için önce FPGA'da donanımsal olarak oluşturulan MAC modülü ve soft işlemciyi konfigüre eden yazılım oluşturulmuştur. Daha sonra, ağdan gerçek zamanda alınan paketlerin özetlerinin elde edilmesini sağlayan yazılım ile bu paket özetlerini kullanarak çevrim içi paket sınıflandırmasının yapılması için üç farklı algoritmayı gerçekleştirecek yazılımlar oluşturulmuştur. Yazılımlar soft işlemcinin program hafızasına yüklenerek STS'nin çevrim içi çalışması sağlanmıştır. Çevrim içi çalışan sistem için gerekli testler yapılarak uygun sonuçların alındığı belirlenmiştir.

Anahtar Kelimeler: Saldırı Tespit Sistemi, Gömülü Sistem, Ethernet IP MAC Core, Alan Programlanabilir Kapı Dizileri (FPGA), Yongada Programlanabilir Sistem (SOPC)

SUMMARY

INTRUSION DETECTION SYSTEM DESIGNS FOR COMPUTER NETWORKS AND THEIR IMPLEMENTATIONS IN FPGA ENVIRONMENT

In order to ensure computer network security, researchers have been developing both software and hardware based products such as firewalls, antivirus programs, and Intrusion Detection Systems (IDS). IDSs are classifier systems deciding whether the pockets on the networks are intrusion pockets or not. In the networks with heavy traffic, fast running of IDSs and the lack of delay resources are very important. Therefore, the researches are continuing increasingly to be developed hardware based IDSs. The investigations concerning IDS are separated into two areas. One of them is to employ different classification algorithms to classify these packets quickly and correctly and the other is the improvement of hardware structure on which those algorithms will run.

In this thesis, an IDS design which can run on real-time in computer networks and its implementation in FPGA (Field Programmable Gate Arrays) environment is intended. For this purpose, classification of packets which flows from network and have attack characteristics were performed according to the packet header structure using Artificial Neural Networks, Naive Bayesian and Decision Tree method. All three algorithms were trained using a predetermined training data and test data in the system and IDS's work has been observed against test data.

A hardware based SOPC (System on Programmable Chip) system has been developed in Altera Cyclone III based EPC3C40F484C7N FGPA platform for IDS to work in real time. SOPC system has a MAC module which communicates with the computer network and has a soft processor. MAC module created in FPGA as hardware is configured and soft processor is programmed to create real-time IDS. Then, the software ensuring the summaries of the pockets received in real time from the network and the software performing online pocket classification with three different algorithms have been developed. The software has performed the online running of IDS, to be loaded the program memory of soft processor. With the necessary tests for the online running system, the appropriate results have been obtained.

Key words: Intrusion Detection System, Embedded System, Ethernet IP MAC Core, Field Programmable Gate Array, System on Programmable Chip.

ŞEKİLLER LİSTESİ

Sayfa No

Sayfa No

Sayfa No

TABLOLAR LİSTESİ

Sayfa No

1. GİRİŞ

Bilgisayar ve sayısal iletişim teknolojilerindeki hızlı gelişmeler, insanların sosyal yaşamlarında da önemli değişikliklere neden olmuştur. Öyle ki kişiden kişiye veya kişiyle devlet ve özel sektör arasındaki iletişim ve etkileşimin önemli bir kısmı, artık elektronik ortamda yapılmaktadır. İletişim ağları üzerinden yapılan e-devlet, e-ticaret uygulamaları, görüntülü telefon uygulamaları analog iletişim sistemlerinin yerini almakta ve hızla yayılmaktadır. İnternet kullanımı günlük hayatın bir parçası haline gelmiş olup, insanlar iletişime yönelik ihtiyaçlarının büyük bir kısmını internet üzerinden karşılamaya başlamıştır.

Ancak İnternet ortamında veya bilgisayar ağları üzerinde taşınan verilerin, istenmeyen erişimlere yada izinsiz kullanımlara karşı korumasız olduğu da bir gerçektir. Ayrıca İnternet veya ağ sistemine bağlı bilgisayar, sunucu v.b cihazlarında her an için bir elektronik saldırıya maruz kalabileceği göz ardı edilemez. Hem taşınan verilerin hem de sistemdeki cihazların saldırılara ve izinsiz işlemlere maruz kalmaması için yapılan çalışmaların geneli, bilgi sistemleri güvenliği ve ağ güvenliği konusudur.

Güvenlik için; şifreleme, kimlik doğrulama, veri bütünlüğü, sayısal imza, sanal özel ağlar, biyometrik sistemler gibi birçok güvenlik sistemleri üzerine araştırmalar devam etmektedir[1].

Ancak tüm bu güvenlik tedbirlerine rağmen bilgisayar sistemleri, dışardan veya içerden yapılan iyi yada kötü amaçlı saldırılarla daima karşı karşıyadır. Saldırı, sistemin güvenlik servislerini etkisiz hale getirmeyi amaçlayan ve bunun için uygun ortamlarda ortaya çıkan tehditlerdir. Saldırılar, hedeflenen sistemlere ulaşıp verileri değiştirmek yada bozmak, dosyaları almak, verilen hizmetleri servis dışı bırakmak veya hedef bilgisayarı izlemek gibi değişik biçimlerde olabilir.

Saldırıları genelde 4 kategoride toplamak mümkündür.

1. Engelleme: Sistemin çalışması için gerekli olan bir yazılım kısmının yok edilmesi, bir donanımın çalışamaz hale getirilmesidir. Engelleme aktif bir saldırı türüdür.

2. Dinleme: İzin verilmediği halde bir kaynağa erişip, sadece sistemi izleme veya ağdaki dosyaları kopyalayabilmektir. Dinleme bir pasif saldırı türüdür.

3. Değiştirme: İzin verilmediği halde bir kaynağa erişip o kaynakta değişiklikler yapabilmektir. Aktif bir saldırı türüdür.

4. Oluşturma: İzin verilmediği halde bir sistemdeki bir kaynağa erişip, kaynağa istenmeyen yeni eklentilerin yapılmasıdır. Aktif bir saldırı türüdür [1].

Yukarıda bahsedilen saldırıların bir kısmının üstesinden gelmek için önemli teknolojilerden birisi güvenlik duvarları (Firewall)'dır. Güvenlik duvarları, bir ağa giren ve çıkan paket trafiğini belirli bir güvenlik politikasına göre denetleyen donanım ve yazılımdan oluşabilen bir sistemdir. OSI modelinin değişik katmanlarında çalışabilen, güvenlik duvarları mevcuttur. Güvenlik duvarlarının en önemli fonksiyonu, bir geçit oluşturarak iç ağı (korunacak ağı), dış ağdan gelebilecek saldırılara karşı korumaktır. Ancak, güvenlik duvarları kendisine uğramadan geçen saldırılara karşı koyamadığı gibi, bir dış saldırganla ilişkisi olan dahili tehditlere karşı etkisizdir. En önemlisi ağın içindeki tehdit ve saldırılara karşı duyarlı değildir.

Güvenlik duvarını aşan veya iç ağdaki bir saldırganın kötü amaçlar için yapacağı saldırıları tespit etmek, en az güvenlik duvarı oluşturmak kadar önemlidir. Bunun için, güvenlik duvarının arkasında çalışan STS, ağ güvenliği açısından son derece önemlidir. STS'ler temelde ağ üzerindeki paketleri yakalayıp, değişik kriterleri baz alarak değerlendirip yorumlayan yazılım veya yazılım-donanımdan oluşmuş birimlerdir. Hem paketleri yorumlayıp sınıflandırabilen, hem de belirlenmiş kriterleri yerine getirmeyen paketleri engelleyen birimlere ise Saldırı Tespit ve Engelleme Sistemleri (STES) denir.

Herhangi bir nesneyi önceden tanımlanmış bir sınıfa atama işlemi, sınıflandırma olarak adlandırılır. STS'ler, paketlerin saldırı niteliği taşıyıp taşımadığına sınıflandırma işlemi yaparak karar vermektedirler. Saldırıların sınıflandırılması için, Karar Ağaçları [2,3], Genetik Algoritmalar veya Genetik Programlama [4], Naive Bayesian [5], Yapay Sinir Ağları (YSA) [6,7] metodları, kullanılmaktadır.

STS'de, ilk işlem ağdaki paketlerin yakalanması, ikinci işlem paketlerin belirlenen özelliklerine göre sınıflandırılması, son işlem ise paketlerin saldırı özelliği taşıyıp taşımadığına karar verilmesidir. Bu tür sistemler hızlı ve doğru çalışabilmeli, güncellenebilir, ucuz ve esnek bir yapıya sahip olmalıdırlar. Özellikle yüksek hızlı ağlarda yazılım tabanlı STS'lerin kullanılması, yeterli başarımı sağlayamaz. Bu nedenle, donanım tabanlı STS ağ güvenliğinde sıkça kullanılmaya başlanmıştır. Donanım tabanlı STS' ler daha yüksek performans ve başarı sağlar. Bu STS'lerin temel taşı, saldırıları tespit etmek için üretilmiş Uygulamaya Özel Tümleşik Devre (Application Spesific Integrated Circuit, ASIC- UÖTD) üniteleridir.

FPGA'lar (Field Programmable Gate Arrays) uygulamalara özel programlanabilir genel amaçlı entegre devrelerdir. Hızlı, ucuz, kolay programlanabilir ve güncellenebilir olmasından dolayı bilgisayar biliminde sıkça kullanılmaktadır[8,9]. ASIC tasarlama işleminde, FPGA' lar ve onların içerisinde oluşturulabilen SOC (System on Chip) veya SOPC (System on Programmable Chip) sistem teknolojisi gittikçe yaygın hale gelmektedir.

FPGA' nın STS'de kullanımı ile ilgili yapılan bir çalışmada, FPGA içerisinde oluşturulacak STS, YSA kullanılarak gerçekleştirilmiştir [10]. Diğer bir çalışmada, STS'de kullanılması gerekli ağ verilerinin paketlerden elde edilmesi ile ilgili olarak, hızlı ve yeniden konfigüre edilebilir bir sistem sunulmuştur [11]. Benzer bir çalışmada ise ağ tabanlı STS'ler için, hızlı paket sınıflandırma gerçekleştiren bir FPGA tasarımı gerçekleştirilmiştir[12]. Ağ tabanlı gerçekleştirilen bir başka çalışmada ise, IPv4 ve IPv6 paket yapısındaki saldırıların tespiti için, bir FPGA platformu sunulmuştur[13]. İmza tabanlı bir STS'nin tasarlanması için kullanılan sonlu durum makinelerinin gerçekleştirilmesinde Aho-Corasick algoritması kullanılmıştır [14]. Anormallik tespiti ve imza temelli STS'nin FPGA ortamında hibrit bir yapıda gerçekleştirilmesi ile ilgili birçok çalışma yapılmıştır [15,16,17,18].

[19] nolu çalışmada, gömülü bir STS önerilmiştir. Önerilen sistem, bir mikro kontroller ve bir ağ arabirimi içermektedir. Bu sistem ARP saldırılarını tespit etmek için, Ethernet çerçevelerini ve ARP istek paketlerinin analizini gerçekleştirmektedir. [20]'de DOS ve Portscan saldırılarını tespit etmek için bir başka sistem sunulmuştur. Bu sistemde, paket başlık alanları kullanılmıştır. Bu gömülü sistemde paketlerin gömülü bir hafıza alanında olduğu varsayılarak paket başlıklarından özellik çıkarılmıştır.

[21]'de paketlerin saldırı olup olmadığını tespit etmek için, Karar Ağacı algoritması kullanan bir sistem FPGA ortamında gerçekleştirilmiştir [21]. Ayrıca, ağ arabirim donanımı ve paket analiz modülünün birleştirilmesiyle bir ağ tabanlı STS [22]'de sunulmuştur. Bu çalışmada, paketlerin analizi için Snort saldırı veritabanından elde edilen kurallar kullanılmıştır. Bu kurallar ile ağdan elde edilen paketler değerlendirilmiştir. Geliştirilen sistemde, Snort kuralları VHDL koduna çevrilerek, NFA string eşleme yardımıyla paketlerin saldırı özelliği taşıyıp taşımadığına karar verilmiştir[22].

STS tasarımlarında, FPGA'ların kullanılması STS'nin karar verme hızını arttırması bir avantaj olsa da saldırıların tespiti için kullanılan algoritmaların başarımları da bir o kadar önemlidir. STS'nin performans ölçümleri için genel olarak Yanlış Negatif Oranı

(False Negative Rate, FNR), Doğru Pozitif Oranı (True Positive Rate, TPR), F-ölçütü ve Alıcı İşlem Karakteristik (Receiver Operating Characteristic, ROC) eğrileri kullanılmaktadır[23].

STS'ler için çok önemli bir kriterde, elde edilen modelin gerçek zamanda uygulanabilir olmasıdır. Yani ağdaki paketlerin modele göre gerçek zamanda değerlendirilip saldırı varsa bu paketin filtreleme işleminin yapılmasıdır. Bu nedenle, hem modelin çok hızlı karar verebilmesi hem de bu modele göre çalışacak donanımın uygun tasarlanması gerekir.

1.1 Tezin Amacı

Bilgisayar sistemlerindeki güvenlik sorunu, teknolojinin kullanımı arttıkça daha önemli bir hal almış olup, araştırmaların arttığı bir konu haline gelmiştir. Araştırmalar, saldırıların hızlı ve doğru bir şekilde tespit edilebileceği sınıflandırma algoritmalarının, gerçek zamanlı olarak çalıştırılabileceği sistem tasarımları üzerine yoğunlaşmıştır. Günümüzdeki FPGA teknolojileri, tasarım aşamasındaki büyük ölçekli sayısal sistemlerin gerçek zamanlı uygulamalarının defalarca denenmesine imkan vermektedir.

Bu tez çalışmasının amacı ağ tabanlı bir STS'nin değişik sınıflandırma algoritmalarına göre tasarlanması ve her birinin gerçek zamanda çalıştırılıp denenebilmesi için uygun bir donanımsal sistemin oluşturulmasıdır. Bunun için, FPGA üzerine gömülebilen bir SOPC geliştirilmiştir. SOPC içerisinde, dışarıdan programlanabilir bir hızlı mikroişlemci ve çevre birimleri oluşturularak, değişik STS algoritmalarının gerçek zamanda denenebilmesine imkan sağlanmıştır. Bu FPGA platformu, TCP/IP protokolu esaslarına uygun olarak bilgisayar ağı ile haberleşmektedir. Ağdan gelen paketlerin OSI (Open Systems Interconnection) modelindeki Veri bağı katmanı seviyesinde algılanabilmesi için tasarlanan SOPC içerisine, Çok Yüksek Hızlı Tümleşik Devre Donanım (Very High Speed Integrated Circuit Hardware Description Language, VHDL) dilinde yazılmış olan 10/100 Ethernet MAC modülü gömülmüştür. Böylelikle, FPGA platformu bilgisayar ağı ile veri bağı katmanı seviyesinde haberleşebilmektedir. Dolayısıyla bu FPGA platformu, bilgisayar ağı ile haberleşmeyi gerektiren her türlü gerçek zamanlı işlemler için kullanılabilecektir. Sistemin bu şekilde gerçekleştirilmesi, tezdeki önemli katkılardan birisidir.

Gerçekleştirilen sistem, ağdaki paketleri yakalayarak bu paketlerin saldırı olup olmadığının tespitini yapmaktadır. Paketlerin sınıflandırılması; Naive Bayesian, Karar Ağacı ve YSA algoritmaları kullanılarak belirlenmiştir. Karar Ağacı algoritmasındaki sınıflandırma için saldırı sınıf sayılarının gruplandırılarak azaltılması ve buna göre saldırı kurallarının elde edilmesi ve uygulanması tezdeki katkılardan bir diğeridir.

Bu tezde kullanılan algoritmaları gerçekleştiren yazılımlar, C/C++ dilinde yazılmıştır. SOPC içindeki işlemci programlanarak, bilgisayar ağından gelen paketlerin gerçek zamanda değerlendirilmesi sağlanmıştır. Kullanılan üç yöntemde de, STS' nin başarımları TPR, FNR, ROC eğrileri ile değerlendirilmiş ve paketlerin gerçek zamandaki sınıf tespit süreleri ölçülmüştür.

1.2. Tezin Yapısı

Bir STS'nin, FPGA platformunda tasarlanması, gerçekleştirilmesi ve gerçek zamanda uygulanmasının yapıldığı bu tez çalışmasının yapısı aşağıdaki gibi organize edilmiştir.

Tezin ikinci bölümünde, saldırı ve STS tanımlamaları verilmiştir. Literatürdeki STS'lerin sınıflandırılmasından bahsedilmiştir. Ayrıca STS' ler için kullanılan sınıflandırma algoritmalarından Naive Bayesian, Karar Ağacı ve YSA hakkında bilgi verilmiş ve ilgili algoritmalara göre sınıflandırma yapan yazılımların akış şemaları açıklanmıştır.

Üçüncü bölümde, önce FPGA' lar hakkında bilgi verilmiş sonra FPGA' lar ile uygulama geliştirmek için gerekli aşamalar anlatılmıştır. Ayrıca bu bölümde, tasarlanacak programlanabilir STS gerçekleştirilmesinde kullanılan, donanım tanımlama dili VHDL özetlenmiştir.

Dördüncü bölümde, gerçek zamanlı ve donanım tabanlı bir STS tasarımı için gerekli donanım elemanları açıklanmıştır. Paketlerin ağdan yakalanması için yapılması gereken ilk aşamada fiziksel katman ve görevleri açıklanmıştır. Daha sonra, fiziksel katman yardımıyla ağdan elde edilen işaretlerin veri bağı katmanında çerçeve formatına dönüştürülmesi verilmiştir. Ayrıca bu bölümde çerçevelerin ağdan alınması veya ağa gönderilmesi için gerekli olan 10/100 Ethernet IP MAC Core özellikleri, kayıtçı yapıları ve çerçeve alma ve gönderme prosedürleri açıklanmıştır. Çerçevelerin alınması ve gönderilmesi için gerekli olan, işlemci ve çevre birimleri açıklanmıştır. Donanım ve yazılım geliştirme süreçleri, adım adım açıklanmıştır.

Beşinci bölümde, STS'nin FPGA ortamında donanımsal olarak nasıl gerçekleştirildiği, sistemde kullanılan donanım elemanları ile birlikte adım adım açıklanmıştır.

Altıncı bölümde, önceki bölümde açıklanan ve FPGA da tasarımı gerçekleştirilen donanımın programlanma aşamaları açıklanmıştır.

Yedinci bölümde, kullanılan üç farklı sınıflandırma algoritması ile paketlerin çevrim dışı ve gerçek zamanda nasıl sınıflandırılacağından bahsedilmiştir. Paketlerin sınıflandırılması için, öncelikle bir eğitim verisi oluşturulmuştur. Bu eğitim verisi kullanılarak, paketlerin sınıflandırma başarımlarında etkili olan parametreler ve elde edilen sonuçlar verilmiştir.

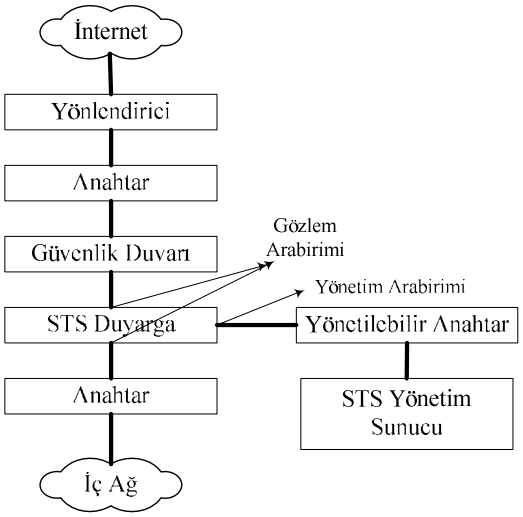
Sonuç bölümünde, genel bir değerlendirme yapılmış ve tezde elde edilen sonuçlar açıklanmıştır.

Tezi desteklemek amacı ile FÜBAP 1675 nolu ve TUBİTAK 108E166 nolu projeler alınmış ve teze paralel olarak yürütülmüştür. Sonuçlandırılan 108E166 numaralı ve "Bilgisayar Ağlarında Bir Saldırı Tespit Ve Engelleme Sisteminin (STES) Gerçekleştirilmesi" başlıklı TUBİTAK projesinin sonuç raporundan, tezin yazımı sürecinde yararlanılmıştır.

2. SALDIRI TESPİT SİSTEMLERİ VE SALDIRI SINIFLANDIRMA

2.1 Saldırı Tespit Sistemleri

STS'ler, İnternet ve yerel ağdan gelebilecek ve sisteme zarar verebilecek çeşitli paketleri belirlemek için, değişik kriterleri baz alarak değerlendirip yorumlayan yazılım veya yazılım-donanımdan oluşmuş birimlerdir. Hem paketleri yorumlayıp sınıflandırabilen hem de belirlenmiş saldırı paketlerini engelleyen birimlere ise Saldırı Tespit ve Engelleme Sistemi (STES) denir. Şekil 2.1' de bir saldırı tespit sisteminin genel birimleri görülmektedir.



Şekil 2.1 STS'nin temel birimleri [24]

Şekil 2.1' de görüldüğü gibi ağ tabanlı bir STS, duyargalardan (sensör), bir veya daha fazla yönetim sunucularından, bilgisayarlardan ve veritabanı sunucularından oluşur. Duyargalar bir veya daha fazla ağ segmentinin aktivitelerini gözlemler ve analiz eder. Ağın gözlemlenmesi için kullanılacak ağ arabirim kartı geçirgen (promiscous) modunda olmalıdır. Bu moddaki ağ kartı bütün gelen paketleri hedeflerine bakmaksızın gözlemler. STS' nin en önemli bileşenlerinden olan duyargalar bir ağda bir veya birden fazla kullanılabilir. Genel olarak donanım ve yazılım olmak üzere iki çeşit duyarga vardır

1. **Donanımsal Duyarga:** Bu duyarga özel bir donanım ve yazılım biriminden oluşur. Bir ağ arabirim kartı ve onun sürücüleri, işlemci ve yardımcı komponentler ile paketin yakalanmasını ve analizini gerçekleştirir.

2.**Yazılımsal Duyarga:** Yazılım bir sunucuya kurulur. Kurulan yazılım paketlerin yakalanması ve analiz edilmesi işlemini gerçekleştirir. Duyarga yazılımı, bir işletim sistemi içermeli veya standart bir işletim sistemine kurulmalıdır.

Özetle, ağ veya bilgisayar trafiğinin analiz edilebilmesi için duyargalar yardımıyla paketler yakalanır. Elde edilen paketlerin sınıflandırma birimine gönderilmeden önce düzenlenerek ham verilerin elde edilmesi gereklidir. Elde edilen veriler, paketlerin sınıflandırılması için STS yönetim birimine gönderilerek paketler hakkında kararlar verilir.

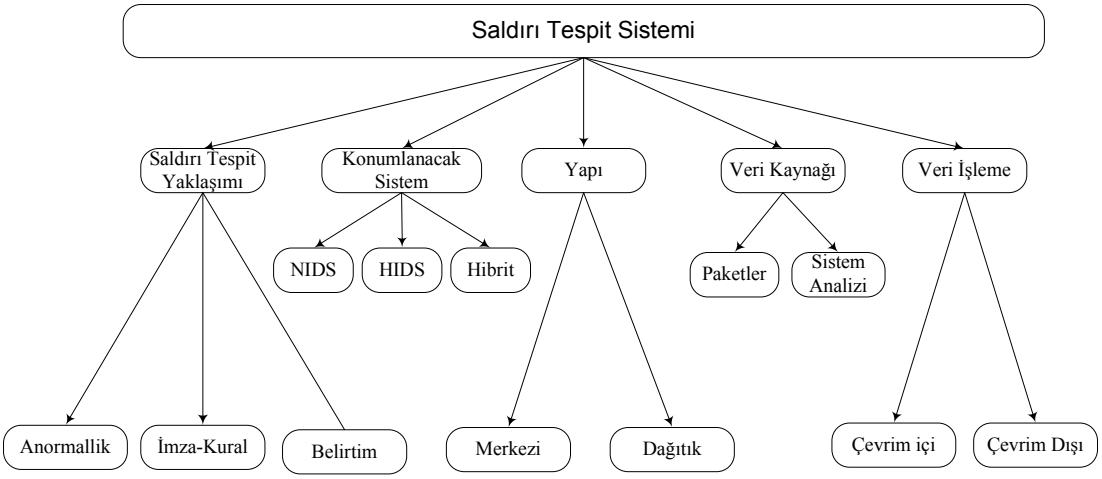
STS'ler kullanıcılara üç temel noktada önemli faydalar sağlamaktadır.

1. **Saldırıların erken tespiti:** Sistem, gerçekleşen bir saldırıyı hedef ağın ya da sistemlerin yöneticilerinden önce tespit edebilir. Bu özellik sayesinde bir saldırı tespit edilir edilmez sorumlu yöneticilere SMS, e-posta gibi yollarla alarm ulaştırılması ve saldırılara karşı olabildiğince erken önlem alınması sağlanabilir.
2. **Saldırılara ilişkin detaylı bilgi toplanması:** STS'ler sayesinde, sona ermiş ya da sürmekte olan bir saldırı hakkında detaylı bilgiler edinilebilir. Bu bilgiler, saldırının boyutları ve muhtemel saldırganlar konusunda analizler yapılması noktasında anlamlı olacaktır.
3. **Saldırılara ilişkin delil toplanması:** STS tarafından toplanan bilgiler saldırı sonrasında, mahkemeye ya da saldırının kaynağı olarak görüşen ağın sorumlularına başvururken delil olarak kullanılabilir.

STS'ler ağ mühendislerine, ağda olası saldırıları tespit etmede ve engellemede kolaylık sağlar. Yerleştirildikleri sistem gereği gerek sunuculara özel gerekse tüm ağa özel koruma sağlamaktadırlar. Güvenlik Duvarı ve Yönlendirici gibi pasif güvenlik araçları değildirler, aktif olarak raporlama, engelleme ve öğrenme gibi önemli özelliklere sahiptirler. STS'lerin kullanılması durumunda, ürettikleri sonuçlar ile potansiyel olarak tehlike arz eden güvenlik zaaflarını, saldırganların davranış ve tepkilerinden belirlenebilmesine olanak sağlar. Gerekiyorsa bu sonuçları değerlendirerek çeşitli önlemler alınabilmesine olanak sağlar. Ağın veya kullanıcıların eğilimlerini, saldırıların karakteristiğini saptama ve ağırlık verilmesi gereken ağ birim veya noktalarını kontrol etmede kolaylık sağlar.

STS'lerin en büyük dezavantajı, saldırı olmayan aktiviteleri bazen saldırı olarak algılaması (False-Negative) ve saldırı olan aktiviteleri ise bazen saldırı olarak algılamaması (False-Positive) olayıdır. Bu yüzden, STS uygulanmasında uzman faktörü ve bilgisi çok

önemlidir. Ağ mühendisleri tarafından kayıtların incelenmesi gerekmektedir. Kayıtların doğruluğu tespit edildikten sonra, harekete geçilip çok basit önlemler alınabilir. Alınabilecek önlemlere örnek olarak kuralların veya eğitim verilerinin güncellenmesi verilebilir. STS'ler Şekil 2.2' deki gibi sınıflandırılabilir[25, 26].



Şekil 2.2 STS' lerin sınıflandırılması

2.1.1. Saldırı Tespit Yaklaşım Yöntemlerine Göre STS'ler

Saldırı tespit yöntemlerine göre 3 temel tip STS söz konusudur. Birincisi anormallik tespiti, ikincisi kötüye kullanım tespiti ya da bazı kaynaklarda tanımlandığı üzere imza-tanıma temelli, üçüncüsü ise belirtim temelli STS' lerdir [26].

1.Anormallik Tespiti: Anormallik (Anomaly) normal davranıştan ayrılma olarak tanımlanır. Burada normal davranıştan farklılık gösteren davranışların saldırı olarak nitelendirilmesi söz konusudur. Normal bir sistemde kullanıcı istekleri tahmin edilebilirdir ve istatistiksel değerlerle uyuşur. Burada normal davranışın bilinmesi ve modellenmesi esastır. Ancak bundan sonra bir anormallik varsa tespit edilebilir. Normal davranış bir dağılımın momentleriyle modellenebilir. Bu yöntemin avantajı daha önceden tanınmayan saldırıların keşfedilmesi olasılığıdır. Dezavantajı ise yanlış alarmların (False Alarm) sayısının yüksek olmasıdır. Anormallik tespitinde istatistiksel yöntemler, YSA, veri madenciliği, bilgisayar bağışıklık sistemi (computer immunology) gibi birçok yaklaşım uygulanabilir.

2. İmza-Tanıma Temelli Saldırı Tespiti: Kötüye kullanım tespiti (Misuse Detection) ya da imza-tanımaya dayalı sistemlerde, her davranışın bir imzası-karakteri vardır. Bunlar

daha önce görülen davranış şablonlarıdır. Eğer gözlemlenen davranış daha önceden bilinen bir saldırı imzasıyla uyuşuyorsa, saldırı olarak sınıflandırılır. Daha önce karşılaşılmadıysa, saldırı olarak nitelenmez. Avantajı saldırıyı kesin olarak tanıyabilmesidir. Yani yanlış alarm vermezler. Ancak tanımlanmamış bir saldırı gelirse bunu tespit edemezler.

3.Belirtim Tabanlı Sızma Belirleme: Belirtim tabanlı sızma belirleme, bir dizi komutun, programın çalışma şekline zarar verip vermediğinin belirlenmesidir. Bu modelde belirtimlerin dışında her olay, potansiyel sızma olarak değerlendirilir. Bu da bilinmeyen saldırı tekniklerine karşı iyi bir savunma çözümüdür.

2.1.2 Konumlanacak Sisteme Göre STS'ler

Konumlanacak sisteme göre STS' ler üç tiptir. Bunlardan ilki ağ tabanlı STS'lerdir. Ağ tabanlı STS'ler (Network Intrusion Detection System, NIDS), ağın kendi segmenti üzerinden geçen trafiği kontrol ederek, ağa yapılan saldırıların tespitini gerçekleştirir. İkinci tip STS ise host tabanlıdır (Host Intrusion Decision System, HIDS). Bu STS'ler belli bir bilgisayar üzerinde çalışarak, kullanıcıların yapacakları hatalardan dolayı oluşacak zararları önlemeye çalışırlar. Böylelikle, işletim sistemine yönelen saldırılar için hangi kullanıcıların sorumlu olduğu belirlenebilir. Her iki sistemin birleştirilmesiyle de hibrit yapıda STS sistemi gerçekleştirilebilir (Hybrid Intrusion Detection System).

2.1.3 Mimari Yapılarına Göre STS'ler

Ağ üzerinden birçok duyargadan verilerin toplanıp bir merkezde işlenmesi ile oluşan mimariye dağıtık STS'ler denir. Ağ üzerinde tek bir duyargadan veri toplayıp değerlendiren STS'ler en çok kullanılan STS'lerdir.

2.1.4 Veri Alma Kaynaklarına Göre STS'ler

Veri kaynağına göre STS'ler iki grupta incelenebilir. Bunlar paket gözleme ve sistem analizi tabanlıdır. Paket gözlemeye dayalı STS'lerde, paketlerin analiz edilmesi esastır. Paketler genelde bir ağ kartı geçirgen moda getirilerek sistemdeki tüm ağ trafiği yakalanır. Yakalanan paketler, STS'ye girdi olarak verilir ve paketlerin saldırı olup olmadığı tespit edilir.

Veri kaynağına göre ikinci tip STS, sistem analizi ile elde edilen verilerin analizini gerçekleştirir. Elde edilecek veriler, sistem dosyalarının kullanımı, servis hizmetlerinin istatistiksel değerleri olabilir.

2.1.5 Veri İşleme Zamanına Göre STS'ler

Çevrim dışı sistemlerde yada gerçek zamanlı olmayan STS' lerde veri önce depolanır, sonra analiz için ilgili STS'ye gönderilir. Çevrim içi sistemlerde ise veri anında analiz edilir. Bu tip STS'ler fazla veri trafiği olan ağlarda, aktif olarak cevap üretilmesi gereken durumlarda kullanılır. Uygulaması zordur.

STS bir veri paketinin saldırı olup olmadığını, kendi saldırı veritabanında bulunan saldırı tipleriyle karşılaştırarak karar verir. Bu yüzden veritabanının doğru bir şekilde oluşturulabilmesi ve güncelleştirilebilmesi STS'ler için son derece önemlidir. STS veri tabanı; paket özellikleri, hareketleri, kullanıcı davranışları gibi kriterlerden istatistiksel veya yumuşak hesaplama metotları kullanılarak elde edilen kurallar veya kararlardan oluşur. Bir saldırı belirleme modeli, bir dizi durumu veya hareketi önceden hazırlanmış veri tabanına göre sınıflandırarak, saldırı yok veya saldırı var tespiti yapar. Anormallik (Anomaly) modeli, Kural tabanlı saldırı belirleme modeli, Belirtim (Spesification) tabanlı saldırı belirleme modelleri literatür de kabul gören modellerdir [26]. Zaman zaman bu modeller beraber kullanılarak birleşik modeller de oluşturulabilir. Tüm bu STS'lerde amaç, saldırının en kısa zamanda tespiti, en az yanlış ve maksimum doğru tespittir.

Ağdan yakalanan veya bir bilgisayara gelen paketlerin saldırı olup olmadığının kontrolü için, veri madenciliği teknikleri kullanılabilir. Saldırılar ile ilgili kurallar veya özellikler veri madenciliği ile elde edilebilir[27,28]. Paket başlıklarının ve paket veri alanlarının depolandığı bir veri tabanından Apriori Algoritması kullanılarak kurallar elde edilebilir[29,30]. Elde edilecek kuralların doğruluğunu geliştirmek için ise FP-Tree, kullanılmış ve Apriori algoritmasından daha iyi sonuçlar elde edilebilmiştir[31].

Anormallik modeli kullanan, CUSUM ve Adaptive Threshold algoritması ile DoS gibi saldırıların tespit edilmesi için bazı modeller önerilmiştir[32,33,34,35] . Ayrıca İşaret işleme teknikleri ve bulanık mantık kullanılarak, DoS gibi saldırıların tespiti gerçekleştirilebilmiştir[36,37,38].

Anormallik tespitinin yanı sıra STS'ler, kural tabanlı olarak ta gerçekleştirilebilir[39,40]. Mobil ajanlar kullanılarak kural tabanlı STS'lerden bazıları

[41,42,43,44]' de verilmiştir. Her iki saldırı tespit modelini kullanarak çalışan hibrit STS modeli çalışmaları da [45,46,47,48] ' da sunulmuştur.

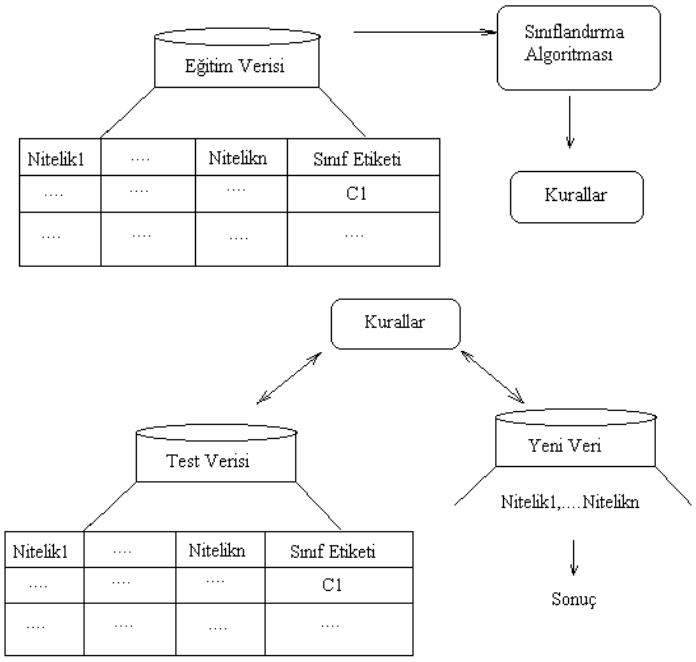
STS'ler için çok önemli bir kriter ise, elde edilen modelin gerçek zamanda uygulanabilir olmasıdır. Yani ağdaki paketlerin modele göre gerçek zamanda değerlendirilip, saldırı varsa bu paketin filtreleme işleminin yapılmasıdır. Bunun için hem modelin çok hızlı karar verebilmesi hem de bu modele göre çalışacak donanımın uygun tasarlanması gerekir.

2.2 Saldırı Sınıflandırma Yöntemleri

Temel olarak elimizde varolan sınıflara, yeni gelen verinin yerleştirilmesi işlemi sınıflandırma olarak adlandırılır. Bunu şöyle ifade edebiliriz; herhangi bir sınıfa ait olmayan bir veri mevcut sınıflardan en yakın hangisine dahil olabilir? sorusunun cevabı sınıflandırma metodunun amacıdır. Örneğin ağdan elde edilen bir paketin protokol tipine bakarak paketin, önceden belirlenmiş bir sınıfa atama işlemi sınıflandırma olarak adlandırılabilir. Bu tür bir sınıflandırma ile ağdaki saldırı özelliğindeki veya bozuk paketlerin eleme işlemi gerçekleştirilebilir. Bir başka örnek olarak, bir ağ üzerinde akan paketlerin video, ses yada veri özelliğinde olup olmadığının kontrolü ile, ağ üzerindeki kişilerin eğilimleri, yapılacak bir sınıflandırma algoritmasına göre belirlenebilir. Böylece, ağın ileriki zamanlarda nasıl bir genişlemeye sahip olabileceği tahmin edilebilir.

Veri sınıflandırma iki süreçten oluşur. İlk süreç öğretmenli öğrenme olarak adlandırılır. Bu yöntemde verinin sınıflandırılması için bir model tanımlanır. Eğitim verisinin her bir satırı bir sınıfa aittir. Her bir satır kayıt yada nesne olarak adlandırılır. Tanımlanmış olan model ile eğitim verilerinin ait olduğu sınıflar (elde edilmiş kural veya istatistiksel değerlerle) belirlenir. Örnek olarak bir ağdan akan herhangi bir paketin sorgulama paketi olup olmadığının belirlenmesi verilebilir. Eğitim paketlerinin sınıflandırılmasıyla, sınıflandırma modeli tamamlanmış olur. Böylelikle, sınıflandırma işlemini gerçekleştiren kural veya istatistiksel veriler elde edilmiş olunur.

Sınıflandırmanın ikinci süreci öğretmensiz öğrenmedir. Daha önceden elde edilmiş kural veya istatistiksel değerler, test verilerine uygulanır. Test verilerinin hangi sınıfa ait olacağı belirlenir. Modelin doğruluğu bu süreçte test edilir. Şekil 2.3 veri sınıflandırma sürecini göstermektedir.



Şekil 2.3 Sınıflandırma süreci

Sınıflandırma yapılmadan önce verilerin hazırlanması gereklidir. Bu ön aşama Veri temizleme, İlişki Analizi ve Veri dönüşümünden oluşur.

Veri Temizleme: Bu ön aşamada gürültü olarak adlandırılan veri, eğitim veri setinden çıkarılır. Bu işlem sınıflandırmanın doğruluğuna direkt etki eder.

İlgi Analizi: Verideki niteliklerin çoğu sınıflandırma için ilişkisiz olabilir. Gereksiz verilerin veri tabanından elenmesi, sınıflandırmanın etkinliğini ve ölçeklenebilirliğini geliştirmeye yardım eder.

Veri Dönüşümü: Sürekli değer alan niteliklerin ayrıklaştırılması önemlidir. Verilerin ayrıklaştırılması sınıflandırmanın maliyetine direkt etki eder.

Sınıflandırma yapmak için seçilen metodun sağlamlığı aşağıdaki kriterlere bağlıdır.

Tahmin Doğruluğu: Yeni veya önceden görülmeyen verinin hangi sınıfa ait olduğunun tahmin edilme yeteneğidir.

Hız: Modelin hesaplama maliyetidir.

Sağlamlık: Gürültülü veya eksik niteliklere sahip verinin bile doğru tahmin edilmesi gereklidir.

Ölçeklenebilirlik: Büyük miktarda veri kümeleri ile etkili çalışabilme yeteneğidir.

Bir sınıflandırıcı model için yanlış alarm ve doğru tespit önemlidir. Yanlış tespit sınıflandırılacak bir örneğin yanlış olarak değerlendirilmesidir. Bir örneğin yanlış olarak

değerlendirilmesi iki türlü olabilir. Bunlardan ilki var olan bir değeri kaçırmak diğeri var olmayan bir değeri varmış gibi tespit etmektir. Örneğin bir paket saldırıysa sınıflandırıcı bu paketi normal olarak tespit ediyorsa, paket yanlış tespit edilmiştir ve yanlış alarm meydana gelmiştir denir. Yanlış alarm sınıflandırıcı sistemlerinin başarımını ölçmekte önemli bir parametredir. Çünkü bir sistemde izin verilen yanlış alarm sayısı ve doğru tespit miktarı birbiriyle ilişkilidir. Yanlış alarmlara izin verildikçe doğru tespit oranı artmaktadır. Sistem parametreleri ikisinin de optimum olduğu noktaya ayarlanmalıdır. Bu durumda da yapılacak analiz ve başarım hesaplamasında ROC eğrileri kullanılır. Bir sınıflandırıcının başarımı Tablo 2.1' deki Karışıklık Matrisi (Confusion Matrix) ile elde edilebilir.

Tablo 2.1 Karışıklık Matrisi

		Tahmin	
		Negatif	Pozitif
Gerçek	Negatif	a	b
	Pozitif	c	d

a, negatif örneklerin doğru tespit sayısını, b, pozitif örneklerin yanlış tespit sayısını, c, negatif örneklerin yanlış sayısını ve d, pozitif örneklerin doğru sayısını göstermektedir. Bu matristen aşağıdaki eşitlikler sınıflandırıcının başarımını ölçmek için elde edilebilir.

$$AC = \frac{a+d}{a+b+c+d} \tag{2.1}$$

$$TPR = \frac{d}{d+c} \tag{2.2}$$

$$FPR = \frac{b}{b+a} \tag{2.3}$$

$$TNR = \frac{a}{a+b} \tag{2.4}$$

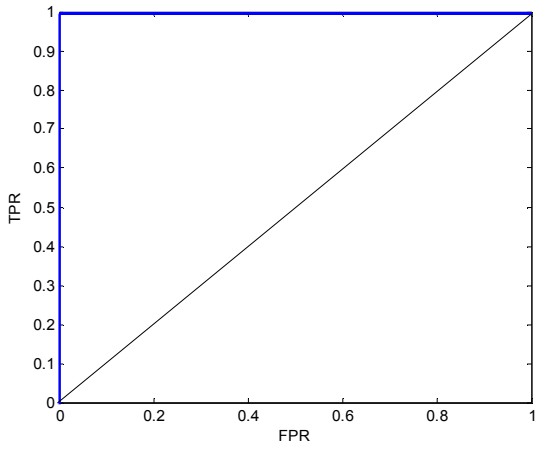
$$FNR = \frac{c}{c+d} \tag{2.5}$$

$$Ke\sin lik = \frac{d}{d+b} \tag{2.6}$$

$$F = \frac{2 * TPR * Ke\sin lik}{TPR + Ke\sin lik} \tag{2.7}$$

Yukarıdaki denklemler sırasıyla doğruluk(Accuracy), duyarlılık veya doğru pozitif oranı, özgüllük veya yanlış pozitif oranı, doğru negatif oranı (True Negative Rate, TNR), yanlış negatif oranı (False Negative Rate, FNR), Kesinlik ve f-ölçütü olarak adlandırılırlar. Sınıflandırıcının sınıflandırma işlemini doğru yapıp yapmadığını öğrenmede her zaman doğruluk oranı yeterli olmayabilir. Bu durumda, duyarlılık ve hassasiyet parametreleri de dikkate alınmalıdır. A ve B gibi iki sınıflandırma algoritmasına göre elde edilmiş değerlere göre, hassasiyeti ve duyarlılığı yüksek olan sınıflandırma algoritması daha iyidir denir. Ancak, bazı durumlarda A' nın duyarlılığı B' den büyük, hassasiyeti ise B' nin hassasiyetinden küçük olabilir. Bu durumda ROC eğrileri hangi sınıflandırıcının daha iyi olduğunu belirlemede kullanılır. Şekil 2.4 ideal ROC eğrisini göstermektedir.

Şekil 2.4' e göre (0,0) noktası bütün örneklerin negatif sınıflandırıldığını, (1,1), Bütün örneklerin pozitif sınıflandırıldığını ve (0,1) ideal durumu göstermektedir. Buna göre, A ve B sınıflandırıcının hangisinin ROC eğrisi sol üst köşeye yakınsa o sınıflandırıcı daha iyidir denir.

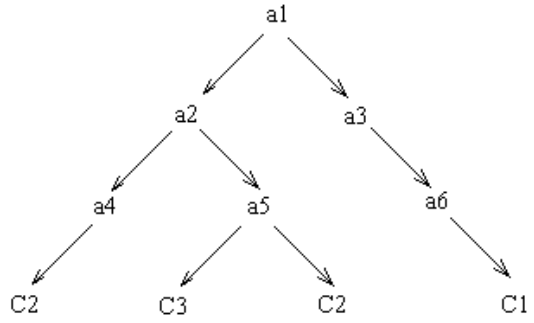


Şekil 2.4 İdeal ROC eğrisi

2.2.1 Karar Ağacı Algoritması ile Sınıflandırma

Doğru ve tanımlayıcı özelliklere sahip, karar ağaçlarının, gerçekleştirilmesi ve yorumlanmasının kolay olması, güvenilirliklerinin iyi olması nedenleri ile sınıflama modelleri içerisinde en yaygın kullanıma sahip algoritmadır. Karar ağacındaki her bir iç düğüm, test verisindeki bir örneğin bir niteliğini gösterir. Her bir dal ise testin sonucunu

verir. Her bir yaprak da sınıf etiketini gösterir. Ağacın en üst düğümü kök düğüm olarak adlandırılır. Tipik bir karar ağacı Şekil 2.5 de gösterilmiştir.



Şekil 2.5 Karar ağacı

Şekil 2.5' deki karar ağacında, **a1, a2, a3, a4, a5, a6** veri tabanındaki kayıtların niteliklerini, **C1, C2, C3** ise sınıf etiketlerini göstermektedir. Kök düğüm a1 özelliğidir. Kök düğümden yapraklara doğru gidilerek karar ağacı sınıflandırma kuralları elde edilir.

Algoritması verilen karar ağacı böl ve yönet mantığına dayalı, öz yinelemeli bir algoritma olup, Greedy algoritması tabanlıdır. Algoritma ilk olarak eğitim verisindeki nitelikleri kullanarak bir tek düğümden başlatılır. Kök düğüm Entropi ve bilgi kazancına göre belirlenir. Kök düğüm dallara ayrılarak yapraklar oluşturulur. Algoritmanın durdurma kriterlerine ulaşılana kadar her yeni dal yapraklara ayrılır. Her bir yaprak eğitim verisindeki sınıf etiketini içerir. Algoritmanın sonlandırılması aşağıdaki üç şarta bağlıdır.

1. Tüm örnekler aynı sınıfa ait ise,
2. Örneklerde kalan nitelik yok ise
3. Dal için örnek yok ise

Durdurma kriteri sağlanırsa eğitim verisinden karar ağacı oluşturulmuş olur. Test verilerinin sınıflandırılması için karar ağacının kök düğümünden yapraklara doğru gidilerek, eğer-o zaman kuralları elde edilir.

Karar ağaçlarına dayalı olarak geliştirilen birçok algoritma vardır. Literatürde en yaygın olarak bilinen algoritmalar ID3, C4.5 ve C5'dir. C4.5 algoritması, sınıflandırmada en ayırıcı özelliğe sahip değişkeni bulurken, entropi kavramından yararlanır [49]. Entropi matematiksel olarak aşağıdaki gibi tanımlanır. $<p_1,p_2,\ldots p_n>$ veri tabanındaki kayıtların olasılıklarını ifade ederse, tüm olasılıkların toplamı 1 olmalıdır. $\sum_{i=1}^{n} p_i = 1$, bu durumda entropi denklem.2.8' deki gibi olacaktır.

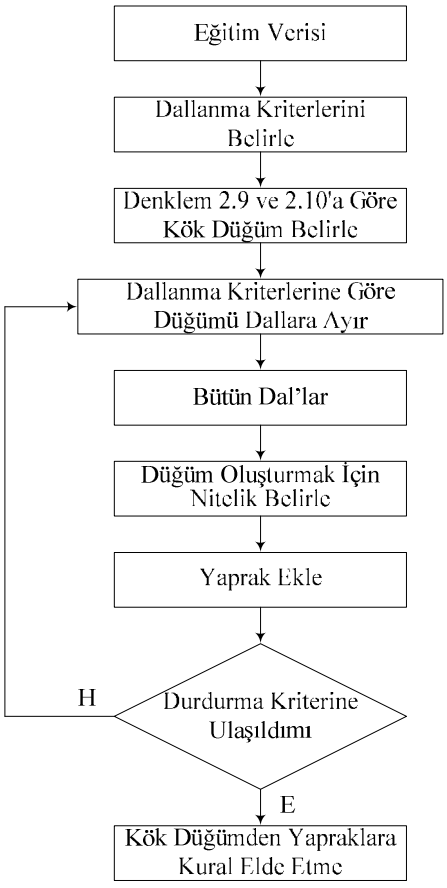
$$H(P) = \sum p_i \log(1/p_i) \tag{2.8}$$

Yukarıdaki denkleme göre, veritabanının tamamının entropisi hesaplanır. Veritabanı farklı bölümlere ayrılırsa her bir alt bölümün de entropisi hesaplanmalıdır. Buradaki H fonksiyonu veritabanının herhangi bir durumdaki durumunu temsil etmektedir. C4.5 algoritması kullanılarak ağaç elde edilirken, her bir alt ağaç yapraklara dönüştürülür [48]. Ağaç yapısı oluşturmak için, her bir alt ağacın yaprağa dönüşümü denklem 2.9 ve denklem 2.10'da gösterilen kazanım ve ayırma oranları ile gerçekleştirilir [49,50].

$$KO = K(D,S) / AO(D,S) \tag{2.9}$$

$$AO(D,S) = H(\frac{|D_1|}{|D|},...,\frac{|D_s|}{|D|}) \tag{2.10}$$

Şekil 2.6, tezde kullanılan eğitim verisinden karar ağacının elde edilmesi için genel akış şemasını göstermektedir.



Şekil 2.6 Karar ağacı algoritması akış şeması

Karar ağacı algoritmasına göre sınıflandırma kurallarının çıkartılması Şekil 2.6'daki algoritmaya göre yazılan program yardımıyla, klasik KDD CUP 99 veri seti sınıflandırılmış olup elde edilen sonuçlar Bölüm 2.4'de verilmiştir.

Aynı algoritmaya göre, güncel veriler kullanılarak çevrim dışı ve gerçek zamanda ağ paketlerin sınıflandırma uygulamaları tezin 7. bölümünde detaylı olarak açıklanmıştır.

2.2.2 Bayes Teoremi ile Sınıflandırma

Makine öğrenmesi tekniklerinden biri olan Bayes teoremi ile sınıflandırma istatistiksel temellidir. Bayes teoremi ile sınıflandırma olasılık hesabı yapılarak belirlenir. Bayes sınıflandırma, Naive Bayes olarakta bilinir. Naive Bayes sınıflandırma işleminde nitelikler birbirinden bağımsızdır. Bu temel kabul, şartlı bağımsız sınıf olarak adlandırılır.

X sınıfı bilinmeyen bir örnek veri olsun. H, belirli bir C sınıfına ait X örneği için hipotezdir. Genel olarak sınıflandırma problemi için P(H\X) olasılığı, H hipotezinde gözlemlenen X veri örneğini verir.

P(H\X) sonraki olasılık olarak ta adlandırılır. Örneğin bir ağdaki kullanıcıların davranışlarına, ilgi duydukları internet sitelerine veya hobilerine göre sınıflandırılabilir. Farz edelim ki X günlük gazeteleri ve spor sayfalarını okuyor bu durumda H hipotezi X'in bir fanatik olduğunu belirtir. O zaman P(H\X) hipotezin güvenliğini belirler. Bu sonuca zıt düşünce olarak P(H) önceki olasılıktır denilir. Yukarıda verilen örnek için P(H) verilen herhangi bir X kişisinin fanatik olma ihtimalini belirtir. Sonraki olasılık P(H\X), bilgi tabanlıdır ve önceki olasılık P(H), X' ten bağımsızdır.

Benzer şekilde P(X\H), H şartı ile X'in olasılığıdır. X günlük gazeteleri ve spor sayfalarını okuyor ise biz biliriz ki X fanatiktir. P(X), X in önceki olasılığıdır. Örneğimizi kullanarak ağ kullanıcılarından bir kişinin hem günlük gazete hemde spor sayfalarını okumasının olasılığını hesaplayabiliriz.

Burada P(X), P(H) ve P(X\H) verilen veri kümesinden elde edilen olasılık değerleridir. Böylelikle Bayes Teoremi denklem 2.11' deki gibi yazılabilir.

$$P(H \mid X) = \frac{P(X / H)P(H)}{P(X)} \tag{2.11}$$

Naive Bayesian sınıflandırıcı Bayes teoremi tabanlıdır ve aşağıdaki gibi tanımlanır.

1. Her bir veri örneği n boyutlu özellik vektörü ile temsil edilir $X=(x_1,x_2,...,x_n)$. Her bir örneğin nitelikleri $A_1,A_2,...A_n$ dir.
2. Veri kümesindeki verilerin ait olduğu sınıf sayısı m ise sınıflar $C_1,C_2,...C_m$ ' dir. Sınıfı belli olmayan X örneği en yüksek sonraki olasılığına sahip sınıfa ait olacaktır.

$$P(C_i \mid X) \succ P(C_j \mid X) \quad 1 \le j \le m, j \ne i$$

$P(C_i|X)$ maksimum sonraki hipotezi olarak adlandırılır. Böylece Bayes teoremi aşağıdaki gibi yazılır.

$$P(C_i \mid X) = \frac{P(X / C_i)P(C_i)}{P(X)} \tag{2.12}$$

3. P(X) bütün sınıflar için sabittir. Burada sadece $P(X|C_i)P(C_i)$ çarpımının maksimum olması yeterlidir. Burada $P(C_i)$ s_i/s oranı ile belirlenir. s toplam örnek sayısını belirtirken s_i, C_i sınıfına ait örneklerin sayısını belirtir.
4. Nitelikler arasındaki ilişki birbirinden bağımsız olduğu için, $P(x_k|C_i)$ olasılığı denklem.2.13' deki gibi yazılabilir.

$$P(x_k \mid C_i) = \prod_{k=1}^{n} P(x_n \mid C_i) \tag{2.13}$$

$P(x_1|C_i)$, $P(x_2|C_i)$,..., $P(x_n|C_i)$ olasılıkları eğitim verilerinden elde edilir. Burada,

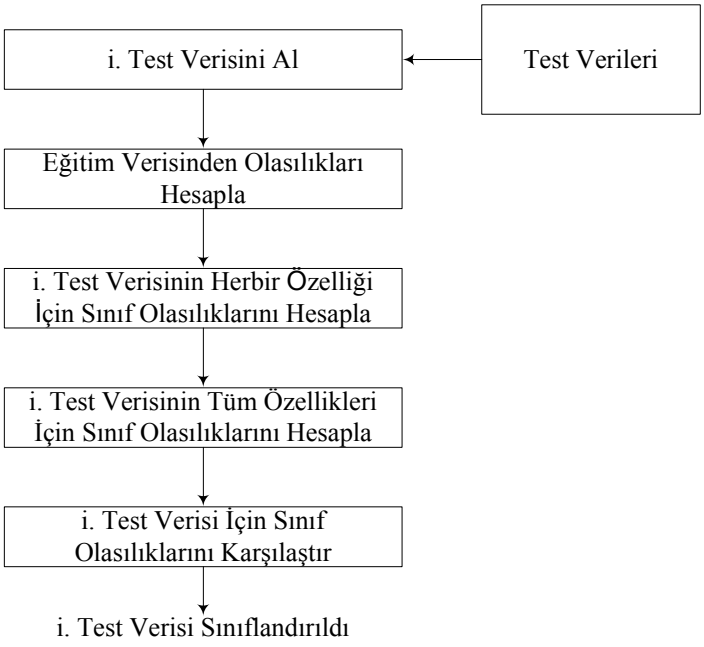
a) Eğer A_k, kategoriksel ise o zaman $P(x_k|C_i)=s_{ik}/s_i$ dir. Burada s_{ik} ,A_k için x_k değerine sahip C_i sınıfındaki örneklerin sayısıdır ve s_i, C_i sınıfına ait örneklerin sayısıdır.
b) Eğer A_k sürekli değerli ise o zaman nitelikler tipik olarak gaussian dağılımına sahiptir. Gaussian yoğunluk fonksiyonu denklem 2.14 teki gibi verilir.

$$P(x_k \mid C_i) = g(x_k, \mu_{Ci}, \sigma_{Ci}) = \frac{1}{\sqrt{2\pi}\sigma_{Ci}} e^{-\frac{(xk-\mu Ci)^2}{2\sigma_{Ci}^2}} \tag{2.14}$$

μ ve σ, C_i nin sırasıyla ortalama ve standart sapmasıdır.

5. Sınıfı bilinmeyen bir X örneğini sınıflandırmak için her bir C_i sınıfı için $P(X|C_i)P(C_i)$ çarpımı hesap edilir. Bu çarpımın en büyük değerine sahip sınıf etiketi X ' e atanır.

Naive Bayesian' a göre eğitim verilerinin sınıflandırılması için gerekli akış şeması Şekil 2.7' de verilmiştir. Algoritma eğitim verisinden hesaplanan nitelik ve sınıf olasılıklarına göre sınıflandırma yapmaktadır.



Şekil 2.7 Naive Bayesian sınıflandırma akış şeması

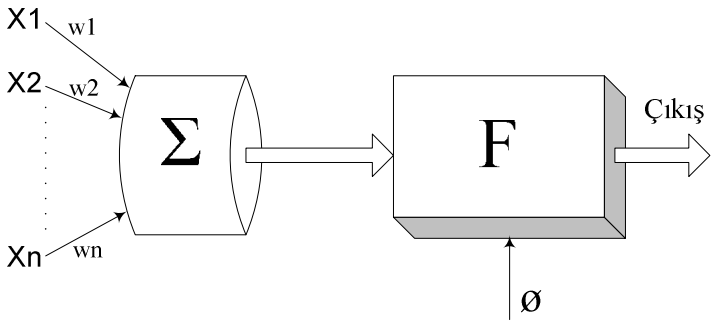
Naive Bayesian sınıflama algoritmasına göre yazılan programda KDD CUP veri seti kullanılarak elde edilen örnek sonuçlar Bölüm 2.4'de verilmiştir.

Aynı algoritmaya göre, güncel veriler kullanılarak çevrim dışı ve çevrim içi ağ paketlerin sınıflandırma uygulamaları tezin 7. bölümünde detaylı olarak açıklanmıştır.

2.2.3 Yapay Sinir Ağları ile Sınıflandırma

YSA, insan beyin yapısı ve özelliklerini kullanarak, herhangi bir yardım almadan öğrenme yoluyla yeni bilgiler üreten ve keşfedebilen yazılım ve donanım tabanlı sistemler olarak adlandırılabilir. Gerçekleştirilecek bir YSA ile öğrenme, ilişkilendirme, sınıflandırma ve optimizasyon gibi problemleri başarılı bir şekilde çözülebilmektedir.

İnsan sinir ağındaki sinir hücreleri gibi YSA'larda yapay sinir hücrelerini (process) kullanarak öğrenme gerçekleştirirler. Her bir sinir hücresinin beş temel elemanı vardır. Şekil 2.8 de bir yapay sinir hücresinin temel yapısı verilmektedir.



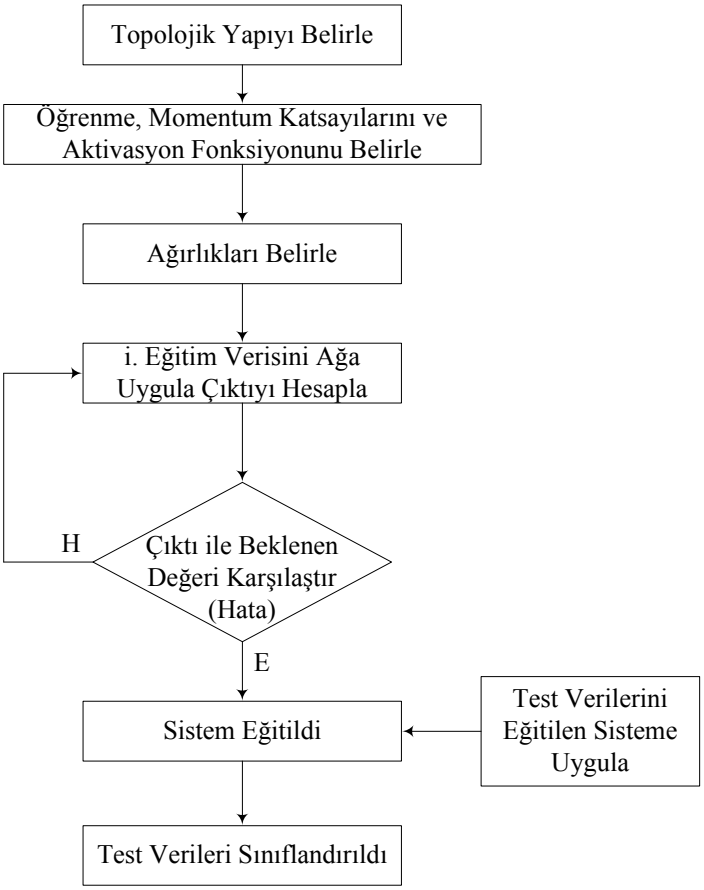
Şekil 2.8 Yapay sinir hücresi ve temel elemanları

Girdi, bir YSA hücresine dış dünyadan verilen bilgidir. Ağırlıklar, YSA hücresine gelen bilginin önemini belirtmektedir. Bu değerin negatif, sıfır veya pozitif olması ağırlığın önemli veya önemsiz olduğunu belirtmez. Toplama fonksiyonu, bir yapay sinir hücresine gelen net girdi değeridir. Literatürde en yaygın olarak kullanılan net fonksiyonu ağırlıklı toplamdır. Bu fonksiyonda yapay sinir hücresine gelen her bir girdi değerinin ağırlıklar ile çarpımının toplamı şeklindedir. Aktivasyon fonksiyonu, yapay sinir hücresine gelen net girdi değerini işleyerek yapay sinir hücresinin bu net değerine karşılık üreteceği çıktıyı belirler. Yapay sinir hücre çıktısı, Aktivasyon fonksiyonu tarafından hesaplanmış değerdir. Bu değer ya bir başka yapay sinir hücresinin verilir ya da sonuç olarak kabul edilir.

Birden fazla yapay sinir hücresinin bir araya getirilmesi ile YSA yapısı oluşturulur. YSA'lar genel olarak üç katmanlı yapıda bu yapay sinir hücrelerinin bir araya getirilmesi ile oluşturulur. Bu katmanlar sırasıyla giriş, ara ve sonuç katmanlarıdır.

Öğretmenli öğrenme yardımıyla sınıflandırmada, YSA eğitilirken sınıflandırılmış eğitim verileri ağa verilerek, çıktılar, eğitim verisindeki sınıflar ile karşılaştırılır. Eğitim aşaması istenen hata değerine ulaşılıncaya kadar devam ettirilir. Eğitim tamamlandıktan sonra test verileri sınıflandırılabilir.

Şekil 2.9 YSA ile sınıflandırmanın nasıl gerçekleştirileceğine dair akış şemasını göstermektedir. YSA' nın topolojik yapısı (katman sayısı gibi) belirlendikten sonra eğitim verileri ile sistemin eğitilmesi gereklidir. Bu amaç için sistemin öğrenme, ağırlık v.s parametreleri belirlenir. Sürekli olarak eğitim verisi sisteme uygulanıp, sistemin çıktısı beklenen değer ile karşılaştırılır ve sistem güncellenir. İstenen hata değerine ulaşılınca sistem eğitilmiş olur. Bu aşamadan sonra sistem test verilerini sınıflandırabilir.



Şekil 2.9 YSA eğitim ve test aşaması akış şeması

YSA ile sınıflandırma algoritmasına göre yazılan programda klasik veri seti (KDD CUP veri seti) kullanılarak elde edilen örnek sonuçlar Bölüm 2.4'de verilmiştir.

Aynı algoritmaya göre, güncel veriler kullanılarak çevrim dışı ve gerçek zamanda ağ paketlerin sınıflandırma uygulamaları tezin 7. bölümünde detaylı olarak açıklanmıştır.

2.3 Kullanılan Veri Seti

Bu tezdeki sınıflandırma algoritmalarının eğitimi ve testi için, KDD Cup 99 klasik veri seti kullanılmıştır. Bu veriler 5. Uluslararası Knowledge Discovery and Data Mining konferansı çerçevesinde düzenlenen bir yarışmada kullanıma sunulan ve neredeyse standart hale gelmiş veri setidir.

Sınıflandırma algoritmalarının paketleri sınıflandırabilmeleri için paketlerin öncelikle bir ön işlemden geçirilmesi gerekir. Bu ön işlemlerden sonra, önceden

belirlenmiş özelliklere göre her paket için bir özet veri elde edilir. Bu veriler ile sınıflandırma algoritmaları paketleri sınıflandırır.

Bu sette basit, içerik, zaman tabanlı trafik ve host tabanlı trafik gibi dört farklı özellik grubu altında belirlenmiş toplam 41 özelliğe göre değerlendirilmiş bulunan 311029 adet paket özeti bulunmaktadır. Veri setinden alınan üç farklı paket yapısına ait 41 özellik aşağıda görülmektedir.

Normal paket

5,tcp,smtp,SF,959,337,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00,144,192,0.70,0.02,0.01,0.01,0.00,0.00,0.00,0.00,normal.

Back saldırı paketi

0,tcp,http,SF,54540,8314,0,0,0,2,0,1,1,0,0,0,0,0,0,0,0,2,2,0.00,0.00,0.00,0.00,1.00,0.00,0.00,118,118,1.00,0.00,0.01,0.00,0.00,0.00,0.02,0.02,back.

Neptune saldırı paketi

0,tcp,http_443,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,114,2,1.00,1.00,0.00,0.00,0.02,0.06,0.00,255,2,0.01,0.07,0.00,0.00,1.00,1.00,0.00,0.00,neptune.

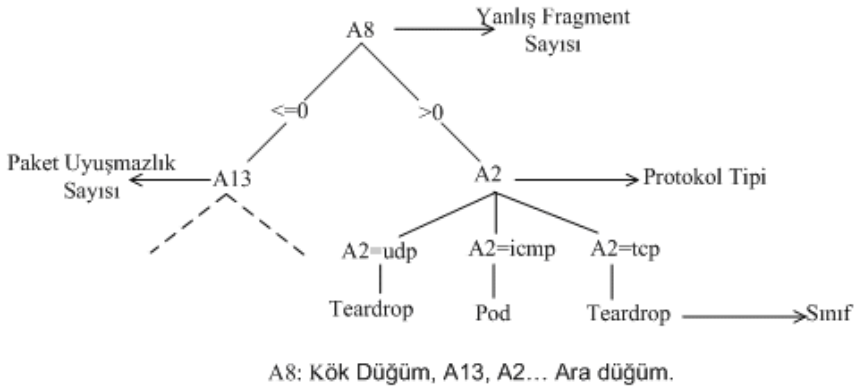
Her üç paket özelliğinde de 1.parametre paket bağlantı süresini, 2.parametre paketin protokol tipini, 6.parametre hedeften kaynağa gönderilen byte sayısını göstermektedir. En son parametre ise paketin hangi saldırı sınıfa ait olduğunu yada normal paket olduğunu göstermektedir. Bu paketlerden her biri toplam 23 adet farklı saldırı tipinden birinin veya birkaçının özelliklerine sahiptir. Bu paket özellikleri ve saldırı tipini gösteren bilgiler Ek-1' de verilmiştir.

Bu veri setini kullanılmasının sebebi paket ön işlemlerinden kaynaklanabilecek hatalardan kaçınmaktır.

2.4 Sınıflandırma Algoritmalarının Yazılımsal Olarak Gerçekleştirilmesi

Bu bölümde, karar ağacı, Naive Bayesian ve YSA sınıflandırma algoritmaları kullanılarak, KDD CUP 99 veri setinin sınıflandırılması gerçekleştirilmiştir. Elde edilen çevrim dışı sonuçlar açıklanmıştır.

Şekil 2.6'da verilen karar ağacı algoritmasını gerçekleştiren yazılımda, eğitim seti olarak KDD Cup 99 eğitim veri seti kullanılmıştır. Bu durumda elde edilen karar ağacının bir bölümü Şekil 2.10 da görülmektedir.



Şekil 2.10 Karar Ağacı algoritmasına göre elde edilmiş kural ağacının bir bölümü.

Eğitim setinde tanımlanmış 23 adet saldırı tipleri için, bu ağaç yapısına göre belirlenen kural örneklerinden biri;

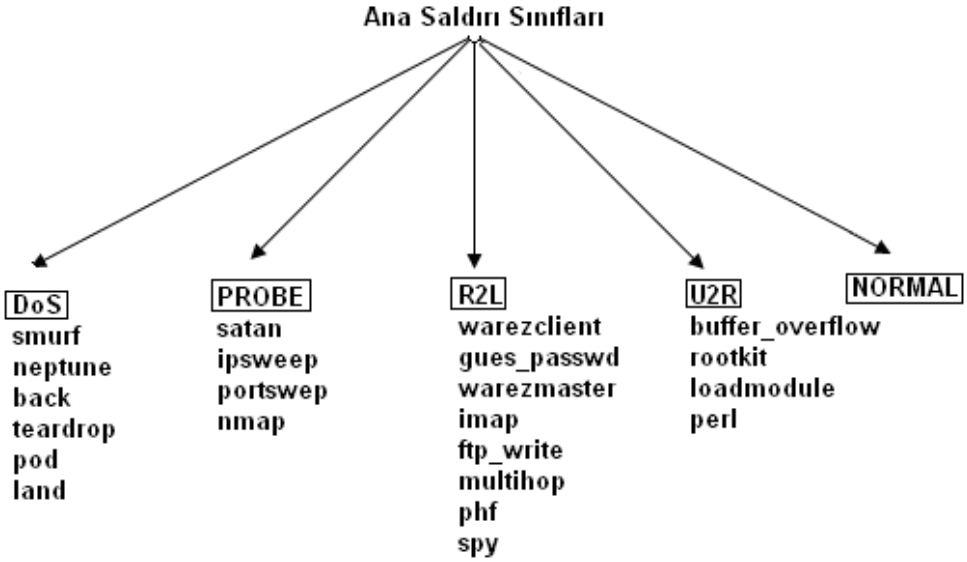
İf (A8 > 0 ve A2 = UDP) Then Sınıf = Teardrop

şeklinde yazılabilir. Elde edilen kurallar KDD CUP 99 test verilerine uygulanarak, paketlerin sınıflandırma işlemleri yapılır. Test örneklerinin en kısa sürede tespit edilebilmesi için thread (İş Parçacığı) yapıları kullanılabilir. Aşağıdaki yapı JAVA programlama dilinde bir iş parçaçığının nasıl tanımlandığı, çalıştırıldığı ve sonlandırıldığını göstermektedir.

```
class [thread adı] extends Thread{
 public void run(){
....
        }
}
public class [Sınıf adı] {
    public statc void main(argümanlar){
   [thread name] thread2=new [thread adı()];
   thread1=start();
}
}
```

Test verilerinin sınıflandırılması için ilk olarak eğitim verisindeki 23 farklı saldırı sınıfı kullanılmıştır. Her bir sınıfın kurallarını içeren 23 tane iş parçacığı yazılmış ve her bir verinin ne kadarlık bir sürede sınıflandırıldığı tespit edilmiştir.

İkinci olarak Şekil 2.11' deki şemaya uygun olarak eğitim setindeki veriler 5 ana sınıfta toplanmıştır. Bu sınıflama 23 adet saldırı tipinden, yapı itibariyle benzer özelliklerde olanların birlikte gruplandırılması ile oluşturulmuştur. Şekil 2.11' de gösterilen bu 5 sınıf için tekrar saldırı kuralları elde edilmiştir. Bu durum için test verilerinin sınıflandırılması, 5 iş parçacığı yazılarak yapılmış ve her bir test verisinin ne kadarlık bir sürede sınıflandırıldığı tespit edilmiştir.



Şekil 2.11 Ana saldırı sınıfları

Son olarak ise Servis Engelleme (Deniel of service, DoS)**,** Bilgi Tarama (Probe)**,** Yönetici hesabı ile yerel oturum açma (Remote to Local, r2l), Kullanıcı hesabının yönetici hesabına yükseltilmesi (User to Root, u2r), saldırıları tek bir saldırı sınıfı, normal paketler ise ikinci bir sınıf olarak değerlendirilmiştir. Bu iki sınıf için tekrar iş parçacığı yazılmış ve her bir test verisinin ne kadarlık bir sürede sınıflandırıldığı belirlenmiştir.

Testler PIV 3.0Ghz işlemcili 512 MB RAM'e sahip masaüstü bilgisayarda yapılmıştır. Şekil 2.6 daki algoritmaya göre JAVA programlama dilinde yazılan program ile 2, 5 ve 23 sınıf sınıflandırma zamanları ve FNR ve TPR başarımları elde edilmiştir. Tablo 2.2 23, 5, 2 sınıfa ait kural sayısı, FNR ve TPR başarımlarını göstermektedir. Sınıf sayılarının azaltılması ile saldırılar için elde edilen kurallar sayısı toplamda azaltılmıştır. Normal sınıfına ait kural sayısında artış gözlemlenmiştir

Tablo 2.2 23, 5, 2 sınıf için kural sayısı FNR ve TPR

a) 23 kural sınıfı için

Saldırı Sınıfı	Kural Sayısı	FNR	TPR
Smurf	1	7.10^{-6}	0,999
Neptune	7	10^{-6}	0.999
Back	6	0,004	0,995
Teardrop	1	0	1
Pod	1	0,018	0,981
Land	2	0	0,952
Satan	6	0,009	0,99
Ipsweep	9	0,002	0.997
Portsweep	6	0,025	0,974
Nmap	3	0,041	0,958
Warezclient	9	0,009	0,99
Guess_Passwd	3	0,036	0,963
Warezmaster	2	0,1	0,9
Imap	2	0,142	0,857
Ftp_write	3	0,384	0,615
Multihop	2	0,333	0,666
Phf	1	0	1
Spy	1	0,5	0,5
Buffer_overflow	4	0,071	0,928
Rootkit	3	0,474	0,526
Loadmodule	3	0,437	0,562
Perl	1	0	1
Normal	29	$1,5.10^{-4}$	0,999

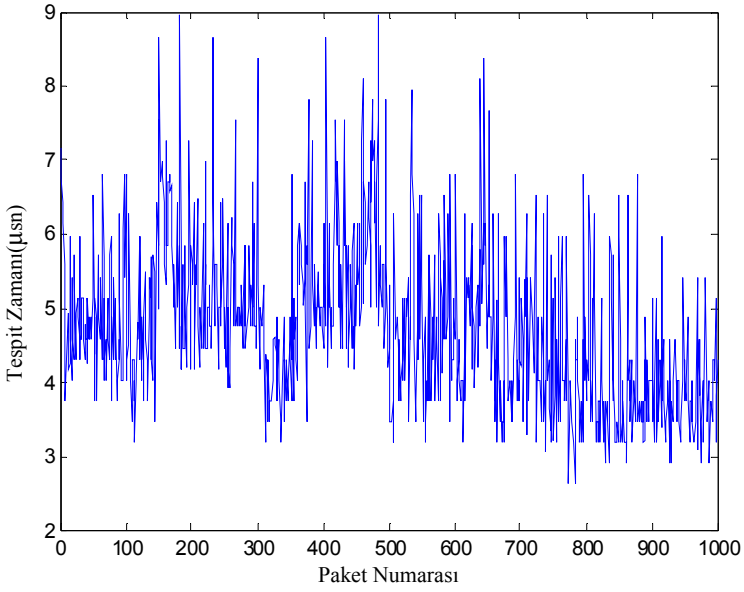
b) 5 ana kural sınıfı için

Saldırı Sınıfı	Kural Sayısı	FNR	TPR
Dos	18	$4,8.10^{-4}$	0,999
Probe	16	0,013	0,986
R2L	15	0,02	0,978
U2R	4	0,413	0,586
Normal	36	$1,6.10^{-4}$	0,999

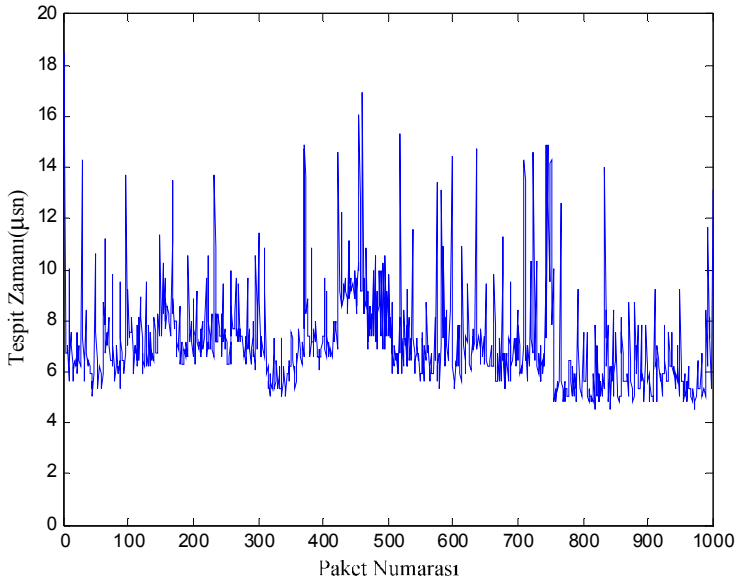
c) 2 Kural Sınıfı için

Saldırı Sınıfı	Kural Sayısı	FNR	TPR
Anormal	37	3.10^{-4}	0,999
Normal	39	10^{-4}	0,999

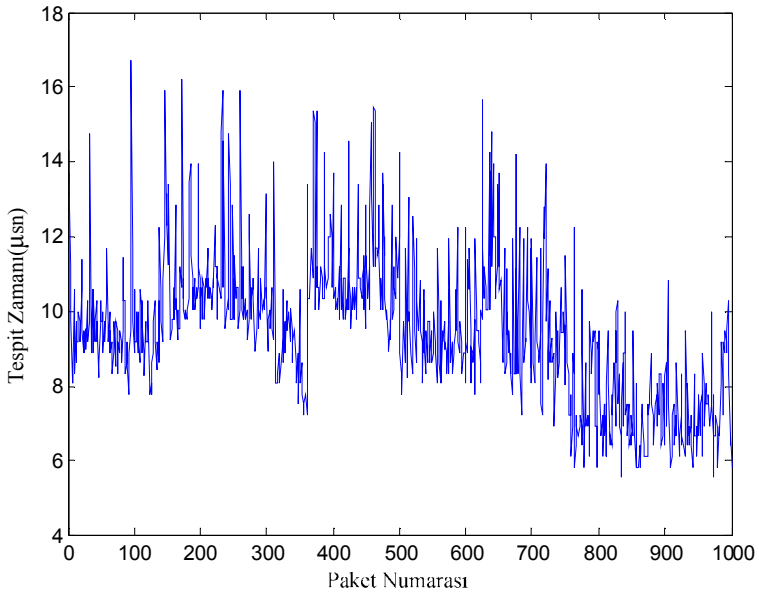
Şekil 2.12 2 sınıf için, Şekil 2.13 5 sınıf için, Şekil 2.14 23 sınıf için sınıflandırılmış paketlerin tespit zamanının değişimini göstermektedir.



Şekil 2.12 2 sınıf için paketlerin sınıflandırma tespit zaman değişimi

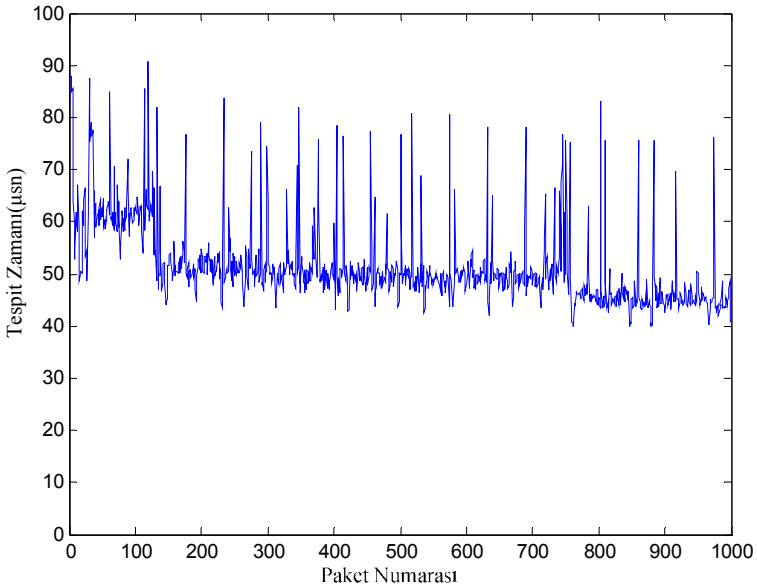


Şekil 2.13 5 sınıf için paketlerin sınıflandırma tespit zaman değişimi



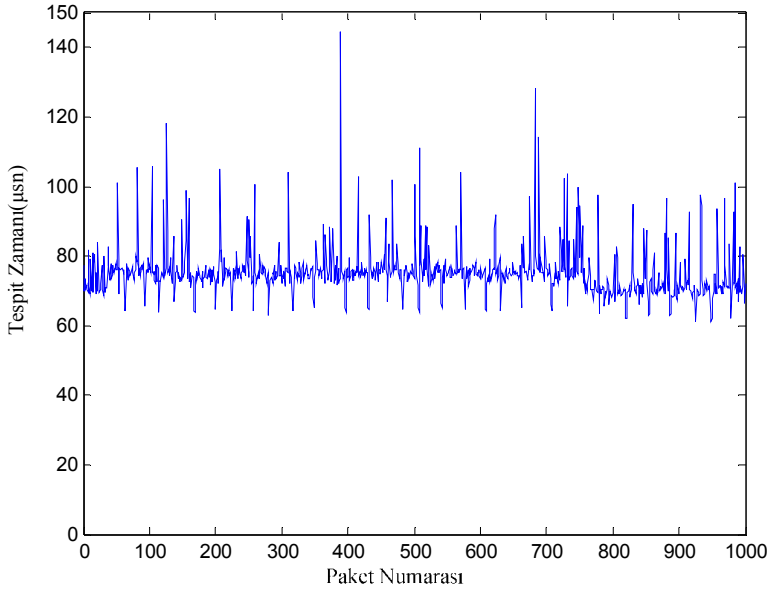
Şekil 2.14 23 sınıf için paketlerin sınıflandırma tespit zaman değişimi

YSA sınıflandırıcı kullanılarak 2 sınıf için, 1000 paketin tespit zaman değişimi Şekil 2.15'de verilmiştir. YSA'nın giriş katmanında, KDD CUP veri setinde tanımlı 41özellikten dolayı 41, ara katmanda 10 ve çıkış katmanında ise 1 sinir hücresi bulunmaktadır. YSA ağının eğitilmesi sonucunda, 1000 pakettin 996 adet paket doğru sınıflandırılmıştır. 4 paket ise yanlış sınıflandırılmıştır.



Şekil 2.15 2 sınıf için YSA ile elde edilen paket tespit zaman değişimi

Naive Bayesian sınıflandırıcı kullanılarak 2 sınıf için, 1000 paketin tespit zaman değişimi Şekil 2.16'da verilmiştir. Naive Bayesian eğitilmesi ve test aşamasında KDD CUP 99 veri seti kullanılmıştır. Naive Bayesian'ın eğitilmesi sonucunda, 1000 paketin 997 paket doğru 3 paket ise yanlış sınıflandırılmıştır.



Şekil 2.16 2 sınıf için Naive Bayesian ile elde edilen paket tespit zaman değişimi

3. FPGA'LAR ve FPGA'LARDA UYGULAMA GELİŞTİRME SÜRECİ

Sayısal sistemlerin gerçekleştirmesi için transistörleri bir kenara bırakılırsa, küçük (Small Scale Integration, SSI), orta (Medium Scale Integration, MSI) ve büyük ölçekli (Large Scale Integration, LSI) sayısal entegre devreleri kullanılabilir. Üreticisi tarafından belirli fonksiyonu gerçekleştirmek için üretilmiş olan bu entegre devrelerin pinleri, uygun şekilde birbirlerine bağlanarak tasarlanan devre donanımsal olarak gerçekleştirilmiş olur.

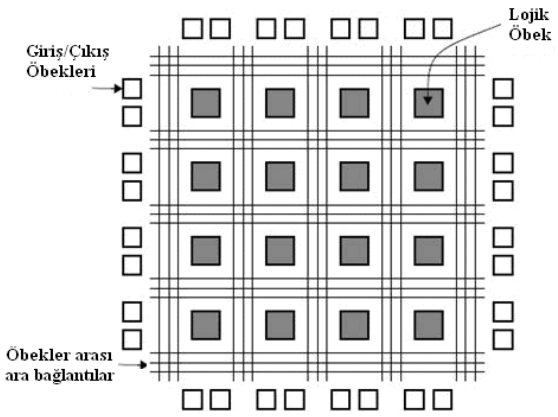
Bu tip gerçekleştirmeler küçük ve orta ölçekli sayısal devreler için uygundur. Esnek değildir. Daha kompleks devrelerin gerçekleştirilmesi için yazılımı da destekleyen donanım yapıları söz konusudur. Bunlar çok büyük ölçekli entegre devreler (Very Large Scale Integration, VLSI) olarak imal edilen SRAM, DRAM, Mikroişlemci entegre devreleri olabilir. Bu yapılar bir bütün olarak kompleks sayısal devreleri gerçekleştirebilirler, ancak birtakım fonksiyonlar yazılım ile gerçekleştirildiğinden, gerçek zamanlı uygulamalar için yeterince hızlı olmayabilirler.

ASIC'ler ise kompleks sayısal sistemleri gerçekleştirmek için üreticisi tarafından bir kereye mahsus olarak oluşturulan, donanımsal entegre devrelerdir. Oldukça hızlı çalışabilen bu devreler yeniden programlanamazlar, oluşturma süreçleri oldukça uzundur, maliyetlidir.

VLSI ve ASIC teknolojileri arasındaki boşluğu doldurabilecek yapılar ise, kullanıcının sayısal tasarımını, sayısal donanım devrelerine dönüştürebilme imkanı sağlayan Programlanabilir Lojik Devreler (Programmable Logic Design, PLD), Karmaşık Programlanabilir Lojik devreler (Complex Programmable Logic Design,CPLD) ve FPGA'lardır.

FPGA'lar, içlerinde programlanabilir lojik eleman öbekleri ve bu öbekler arasındaki ara bağlantıları barındıran, sayısal entegre devrelerdir. Alanda programlanabilir adının verilmesi, mantıksal eleman öbeklerinin ve ara bağlantıların imalat işleminden sonra defalarca programlanabilip yeni sayısal donanımsal devrelerin oluşturulabilmesindendir.

FPGA'lar iç yapı itibarı ile Şekil 3.1'de görüldüğü gibi üç kısımdan oluşmaktadır.



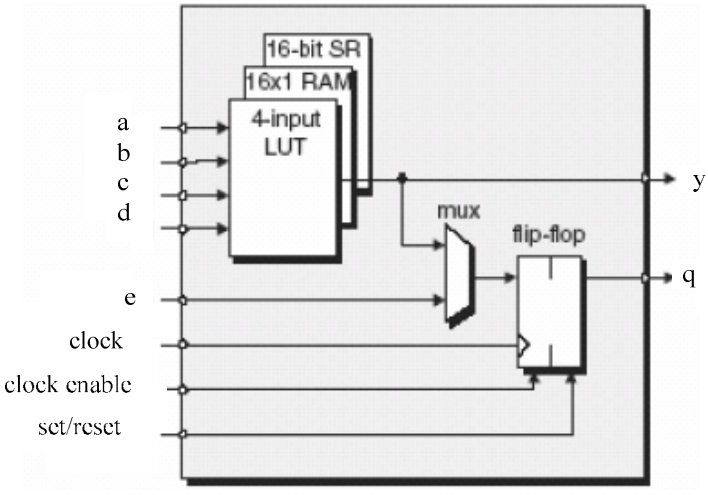
Şekil 3.1 FPGA yongasının iç yapısı

1.Lojik Devre Elemanlarından Oluşmuş Bloklar (Lojik Öbekler): Bu öbekler programlandığı zaman, anlamlı sayısal devre donanımı oluşturabilecek lojik devre elemanlarından meydana gelen birim hücrelerden oluşmuştur. Birim hücreler kendi aralarında bağlantılara sahiptirler.

2.Giriş Çıkış Öbekleri: Bu öbekler FPGA'nın dış dünyayla olan ilişkisini sağlayan, kullanıcının konfigüre edebileceği giriş/çıkış pinleridir.

3.Öbekler Arası Ara Bağlantılar: Bu iç bağlantılar, birden fazla öbek kullanılarak gerçekleştirilen sayısal donanımsal devre elemanlarının birbirleriyle bağlantısını sağlar.

FPGA'lar üretim teknolojileri açısından, Statik RAM temelli, Antisigorta temelli, E^2 PROM/FLASH veya hibrit olarak SRAM / FLASH temelli olarak üretilebilirler [51].



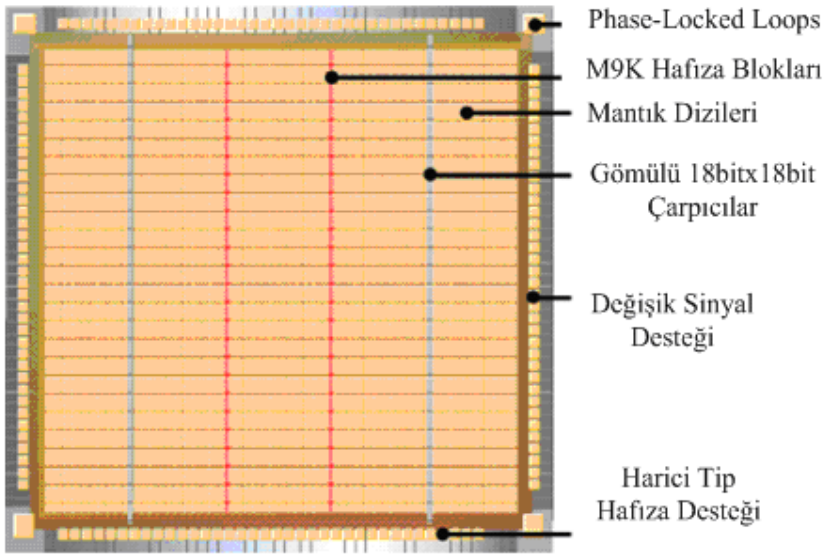
Şekil 3.2. Xilinx Lojik hücresinin basitleştirilmiş görünümü [51]

Lojik Öbekleri oluşturan birim hücrelere LC (Logic Cell) veya LE (Logic Element) denir. Birim hücre mimarisine göre Mux (Çoğullayıcı) tabanlı ve bakma tablosu (Look- Up Table, LUT) tabanlı FPGA'lar vardır. Karmaşık donanımları tasarlamak için LUT tabanlı mimariler daha uygundur. LUT mimarisinin gereği olarak bakma tablosunu oluşturmak için SRAM'lar kullanılır. Bu durum esnek hafıza kullanımına imkan verir. FPGA üretiminde söz sahibi firmalardan Xilinx ve Altera üretimlerinde LUT tabanlı mimarileri öne çıkarmışlardır [51].

Birden fazla LC veya LE'nin bir araya gelmesiyle Lojik öbekler oluşur. Bu lojik öbeklere Xilinx, Konfigüre Edilebilir Lojik Blok (Configurable Logic Block, CLB) ismi verirken, Altera ise Mantık Dizi Blokları (Logic Array Block, LAB) ismini verir.

3.1 Kullanılan FPGA Hakkında Genel Bilgi

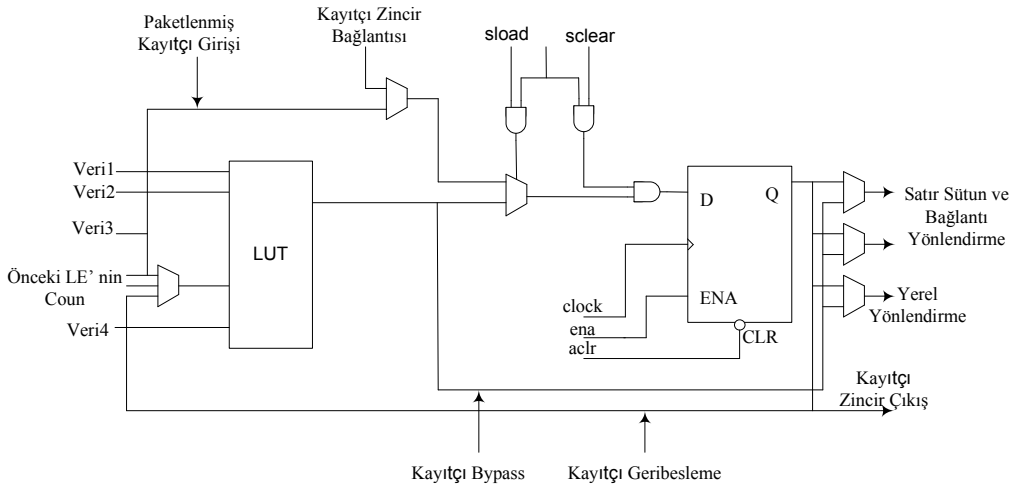
Bu tezde Alteranın Cyclone III tabanlı EPC3C40F484C7N FPGA entegresi kullanılmıştır. Bu entegrenin iç blok yapısı Şekil 3.3'te verilmiştir.



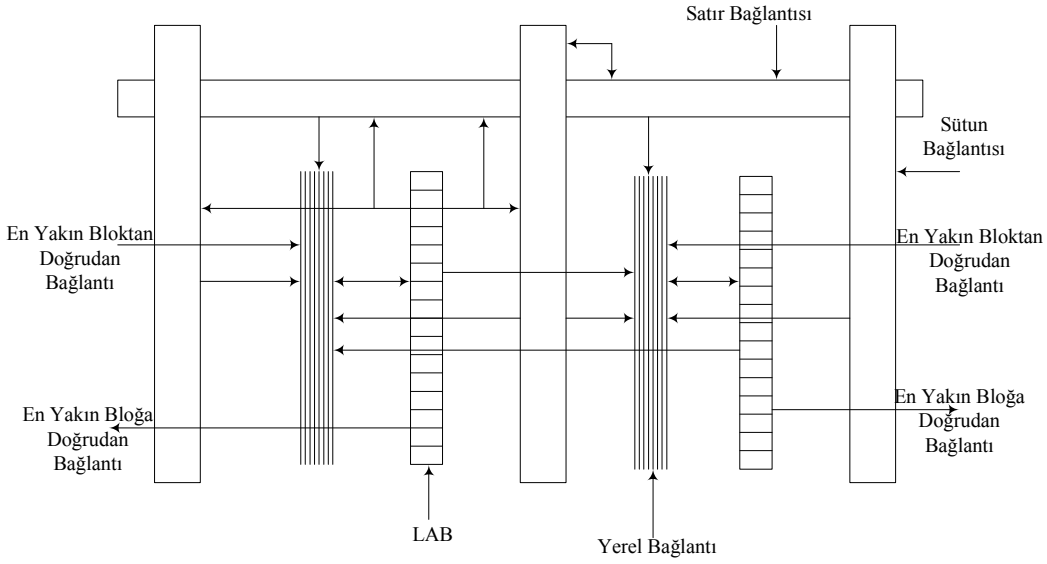
Şekil 3.3. Cyclone III FPGA' nın iç blok yapısı

Altera firması 65 nm teknolojisine sahip Cyclone III FPGA entegresi 5136 ile 119088 lojik elemanlı, düşük güç tüketimli gömülü hafıza elemanları, çarpıcılar, harici hafıza yığınları, farklı sinyal seviyelerine sahip, genel amaçlı bir FPGA'dır.

Bu FPGA'da kullanılan LE'lerin iç yapısı Şekil 3.4'de verilmiştir. LE'ler 4 girişli LUT' a sahip olup bu 4 giriş değişkeni ile mantıksal fonksiyonları ve aritmetik işlemleri gerçekleştirebilirler. Ayrıca lojik elemanlar bir adet kayıtçıya, birer adet taşıma zincir bağlantısına, kayıtçı zincir bağlantısına sahiptir.



Şekil 3.4 LE'nin iç yapısı



Şekil 3.5. LAB'ların birbirleriyle iç bağlantıları[52]

Bir LAB 16 adet LE, LAB kontrol sinyali, LE taşma zinciri, kayıtçı zinciri ve yerel bağlantılar içerir. Yerel bağlantılar aynı LAB içinde lojik elemanlar arasında sinyalleri

transfer eder. Şekil 3.5, Cyclone III tabanlı bir entegrede kullanılan LAB yapısını göstermektedir.

Cyclone III ailesi her bir M9K (9Kbit) hafıza bloklarının 315 Mhz hızında işletilmesine imkan verir. Gömülü 126 adet M9K hafıza blokları kullanıcı tarafından RAM, ROM veya FIFO olarak konfigüre edilebilir.

Cyclone III 288 gömülü çarpma bloklarına sahiptir. Her bir blok 18x18 bit çarpıcı veya 2 adet 9x9 bit çarpıcı olarak kullanılabilir. İstenirse işlemcilerin performanslarının yükseltilmesi için Quartus 8.0 ortamında bu çarpıcılar işlemcilere eklenebilir.

Clyclone III ailesi 20 adet global clock işaretine sahiptir. Bu global clock işaretleri, clock pinlerini, çift amaçlı – clock pinlerini, kullanıcı mantık elemanlarını ve PLL' leri sürebilir. Kullanıcı isterse clock yönetimi, harici sistem clock yönetimi ve giriş/çıkış arabirimleri için PLL kullanabilir[52].

Cyclone III 8 adet giriş/çıkış yığını içerir. Giriş/çıkış yığınları LVTTL, LVCMOS, SSTL, HSTL, BLVDS, PPDS standartlarını destekler.

Cyclone III harici DDR, DDR2, SDR SDRAM, QDRII SRAM tiplerini destekler. DDR2 SDRAM hafıza arabirimi Cyclone III ailesi için 400 Mbps hızında çalışır.

Cyclone III ailesinde kolay ve hızlı bir şekilde 32 bitlik soft işlemci çekirdeği oluşturulabilir. Bu işlemci kullanılarak SOPC tasarımı gerçekleştirilebilir. Cyclone III bir veya birden fazla Nios II gömülü işlemcisinin çevre birimleriyle beraber kullanımına imkan verir.

3.2. FPGA Uygulama Geliştirme Aşamaları

FPGA'ler ile uygulama geliştirmek; elde edilmek istenen devreye uygun olarak FPGA' ların içindeki bağlantıları ve lojik blokları, dahili mikroişlemciler ve ilişkili çevresel aygıtlarla birleştirip hazır bir forma getirme işlemidir.

Tasarımın ilk adımı olarak, istenen bir lojik devre fonksiyonu bilgisayar ortamında tasarlanır. Bu tasarım bir şematik çizim veya yüksek seviyeli bir donanım tasarlama dili şeklinde olabilir. Şematik tanımlamada birçok firma tarafından geliştirilmiş şematik editör programlarından faydalanılır. En yaygın donanım tasarım dilleri olarak (Hardware Description Language, HDL), VHDL (Very High Speed Integrated CircuitHardware Description Language) ve AHDL (Altera HDL) kullanılmaktadır. Ayrıca Verilog ve SystemC dilleri de kullanılmaktadır.

Derleme işlemi sonrasında tüm tanımlamalar standart bağlantı listesi (netlist) biçimine çevrilir. Yapılan tanımlamaların, istenilen fonksiyonları yerine getirip getirmediği fonksiyonel benzetim (functional simulation) yapılarak test edilir. Tasarlanan "lojik devre fonksiyonu" program tarafından bit sıralı bir dosyaya (binary) çevrilir (FPGA Üreticisinin yazılımı ile – Bu dosya FPGA'yı yeniden ayarlama amacıyla kullanılır). FPGA'yı programlamak için öngörülmüş bit sıralı dosya JTAG gibi bir iletişim ortamı kullanılarak FPGA´ya gönderilip, tasarlanan lojik devrenin FPGA içinde oluşturulması işlemi tamamlanmış olur[51].

Ayrıca, karmaşık sistemlerin FPGA' larda tasarlanmasını basitleştirmek için, önceden tanımlanmış hazır karmaşık fonksiyon ve süreçler test ve optimize edilmiş bir şekilde kullanıma hazır halde bulunabilmektedir. Böylece bunları kullanarak karmaşık tasarım süreci hızlandırılabilmektedir. Önceden tanımlı sürecler (karmaşık fonksiyonlar) genelde IP Cores (Intellectual Property) olarak adlandırılır. Bazı IP Core'lara örnek olarak, sayısal filtre modülleri, bazı haberleşme protokol modülleri v.b. verilebilir. "IP Cores" lara FPGA üreticileri ve üçüncü parti IP satıcılarından (çok azı parasız, genelde ticari lisans ile satılır) ulaşılabilir.

3.3. VHDL Donanım Tanımlama Dili

FPGA'larda donanımsal bir sistem oluşturmak için en yaygın donanım tanımlama dili VHDL' dir. Sonraki bölümlerde gerçekleştirilecek programlanabilir sistem için, Altera Quartus Web Edition 8.0 ortamı bu dili kullanmaya imkan vermektedir.

VHDL dili, tam bir uygulama dizisinden ve genel olarak bir donanımsal (sayısal) modellemeden ibarettir. VHDL dilinin kendi kendine dokümantasyon özelliğinin olması ile VHDL modeli, sistem dokümanının standart bir biçimde hazırlanmasını sağlar. Donanım tanımlama dilinin büyük avantajlarından biride, donanımı gerçekleştirecek kodun adım adım çalıştırılabilmesine imkan vermesidir.

Mevcut sentez araçları, donanım tanımlama dilindeki spesifik elemanları tümleşik devrelerdeki standart elemanlarla eşleştirmektedir. VHDL dilinin yapısal elemanları aşağıdaki gibidir[53].

1. **Varlık (Entity) :** Arayüz (Interface)
2. **Mimari (Architecture):** Gerçekleştirme(Implementation), davranış(behaviour)

3. **Konfigürasyon (Configuration):** Model zincirleme (model chaining), yapı (structure), hiyerarşi (hierarchy)
4. **İşlem (Process) :** Uyumluluk(Concurrency), olay denetimi(event controlled)
5. **Paket (Package) :** Modüler tasarım, standart çözüm, veri tipleri, sabitler
6. **Kütüphane (Library)**: Derleme(Compilation), obje(nesne) kodu

VHDL dilinde *Entity* ile bir arayüz oluşturulabilir, port sinyallari birbirine bağlanabilir. *Entity,* olmazsa olmaz komutlarından biridir. *Entity* bir isim olmalıdır ve bu isim belirlenen herhangi bir isim olabilir. Mutlaka *end* ile sonlandırılması gerekir. Bir *Entity*, *end* ile sonlandırıldıktan sonra mimari içerisinde kullanılabilir. *Entity* de davranışsal tanımlama yoktur. Ancak giriş ve çıkış sinyalleri ve onların tipleri port ifadelerinde *port*' anahtar sözcüğü ile *Entity* içerisinde tanımlanır.

Entity; modülün dış dünya ile olan bağlantılarını içerir. *PORT( );* ifadesi giriş çıkışların yönünü sayısını ve türünü belirtmekte kullanılır. Böylece modüle olan giriş çıkışlar belirtilir.

entitiy port modları 4 farklı yapıdadır. Bu port tanımlamaları aşağıdaki gibidir.

1. IN (Giriş) : Sinyal değerleri yalnızca okunabilir.
2. OUT (Çıkış): Sinyal değerleri yalnızca yazılabilir. Çoklu sürücü söz konusudur.
3. BUFFER (Arabellek): Çıkış için karşılaştırılabilinir, sinyal değeri okunabilinir ve tek sürücü söz konusudur.
4. INOUT (Giriş-Çıkış) : Çift yönlü portlardır.

Bir 4 bitlik paralel toplayıcı donanımının gerçekleştirilmesi için VHDL'de yazılmış port tanımlamaları aşağıdaki gibidir.

```
entity toplama is
port(   A:      in std_logic_vector(3 downto 0);
        B:      in std_logic_vector(3 downto 0);
        carry:  out std_logic;
        sum:    out std_logic_vector(3 downto 0));
end toplama;
```

architecture, bir davranışsal tanımlamayı, bir yapısal netlist'i veya bir birden fazla varlık tanımını gerçekleştirir. Bir *architecture* kesinlikle belirli bir *entity*'e bağlanmalıdır.

Bir varlık, içerisinde birden fazla *architecture* barındırabilir. İsim ' *architecture* ' anahtar sözcüğünden sonra gelmelidir. Bunu takiben *of*' anahtar sözcüğü kullanılır ve varlığın ismi arayüz olarak kullanılır. *Architecture*' de tıpkı *entitiy* gibi *end* ile sonlandırılması gerekir. *End* ile sonlanmış *architecture* daha sonra istenirse tekrar kullanılabilir. Kısaca *architecture*; *entity*' de belirtilen port davranışlarının belirlendiği yerdir.

4 bitlik bir paralel toplayıcı donanımının gerçekleştirilmesi için VHDL'de yazılmış *architecture* yapısı aşağıdaki gibidir.

```
architecture govde of toplama is
signal result: std_logic_vector(4 downto 0);
begin
   result <= ('0' & A)+('0' & B);
    sum <= result(3 downto 0);
    carry <= result(4);
end govde;
```

VHDL'de *Configuration* işlemi ile bileşenlerin tam bir tasarımını yapmak için, entitiy/architecture çiftine bağlantı oluşturulur. *Configuration* end sözcüğü ile bitirilmelidir. Bitirildikten sonra tekrar kullanılabilir.

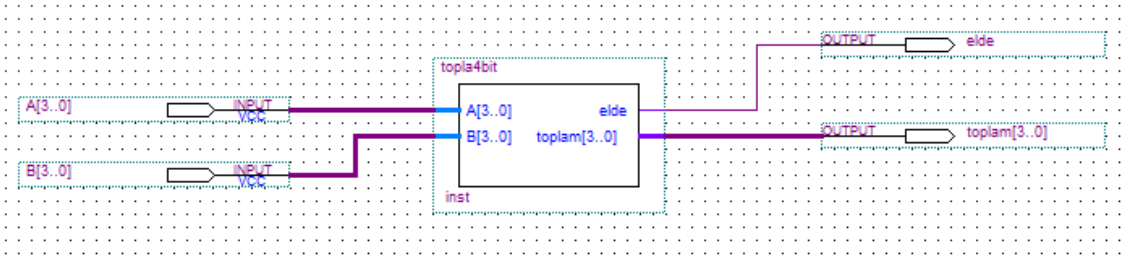
Process, duyarlılık listesi, ifadeler ve işlemlerden oluşur. Duyarlılık listesindeki elemanlarda herhangi bir değişiklik olmadıkça içerisindeki işlemler gerçekleştirilmez. *Process* ifadelerinde işlemler diğer programlama dillerinde olduğu gibi sıralı (sequential) işlenir. *Architecture* bloğunun içine birden fazla *process* konulursa bu *process*' ler kendi içlerinde seri çalışırken dışarıda parallel çalışırlar.

Package, veri tiplerinin, altprogramların, sabitlerin, vb tanımlamalarından oluşur. Paketler tasarımda kullanılacak modüllerinin bir araya getirilip VHDL modelinin oluşturulmasını kolaylaştırmaktadır. Paketler başlık (header) ve gövde (body) bölümlerinden oluşur.

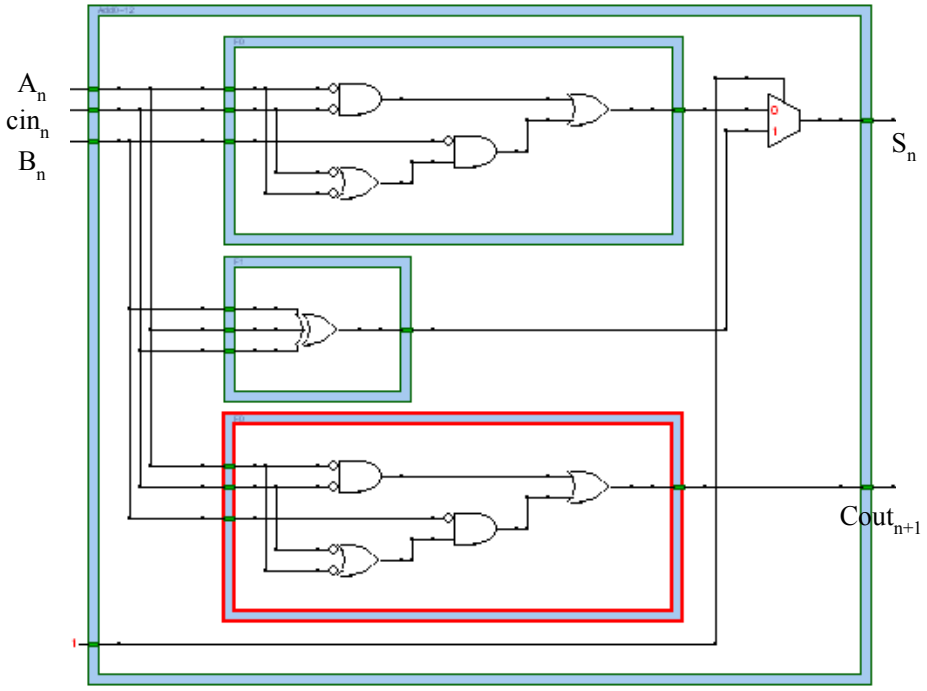
Library, paket gövdeleri, varlıklar, mimariler ve konfigürasyonları içerisinde barındırır.

Yukarıdaki 4 bitlik toplayıcı örneğini sonuçlandırmak için, yani bu devreyi donanımsal olarak gerçekleştirmek için, VHDL kodlarını destekleyen Altera'nın Quartus 8.0 ortamında VHDL kodunun derlenmesiyle Şekil 3.6'daki netlist oluşturulur. Şekilde görüldüğü gibi toplayıcı 4'er bitlik A ve B giriş portlarına, sonuç için 4 bitlik çıkış portuna,

1 bitlik elde çıkış portuna sahiptir. Şekil 3.6 daki 4 bitlik toplayıcı donanımını oluşturan birer bitlik iki sayıyı toplayan tam toplayıcının lojik devresi Şekil.3.7 de verilmiştir. Bu devre VHDL dilinde yazılan 4 bitlik toplayıcı devrenin bir kısmının sembolik şemasıdır.

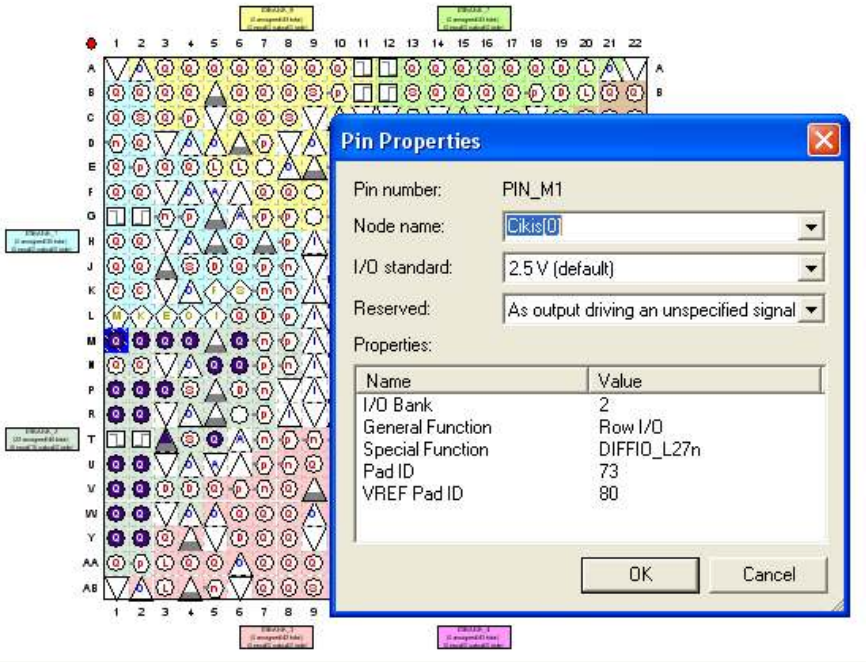


Şekil 3.6 Toplayıcı blok diyagramı

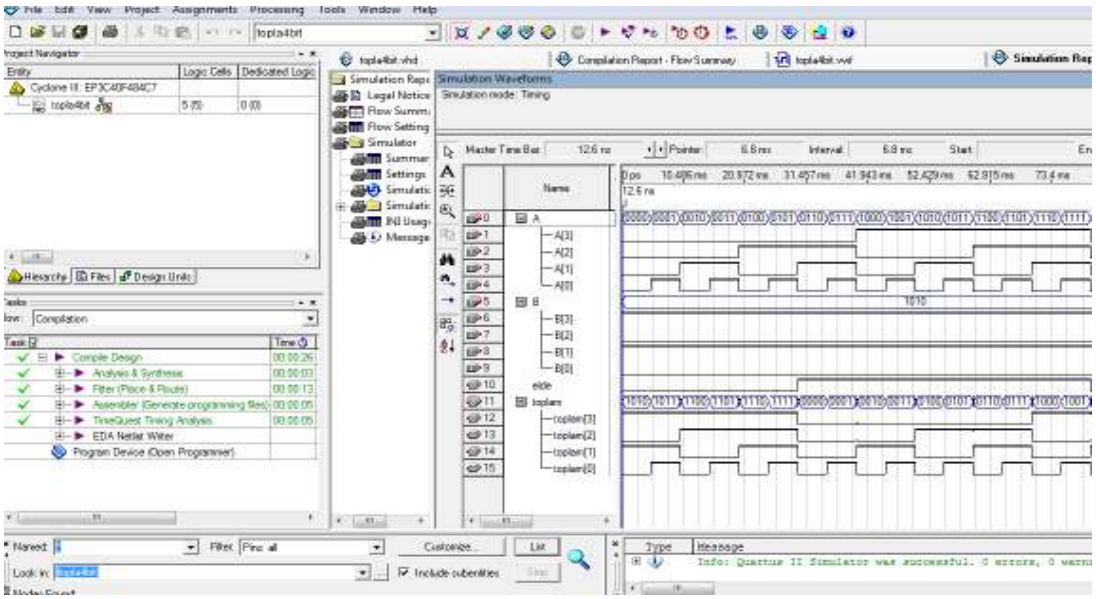


Şekil 3.7 Bir bitlik tam toplayıcı

Toplayıcı devrede kullanılan giriş ve çıkış pinlerinin, gerçek giriş çıkış pinlerine dönüştürülmesi için Şekil 3.8' de görüldüğü gibi pin atama işlemleri yapılmıştır.



Şekil 3.8 Pin atamaları



Şekil 3.9 4 bit toplama işlem simülasyonu

Pin atamalarından sonra toplayıcı devrenin fonksiyonel benzetimi Şekil 3.9 da gösterilgiği gibi gerçekleştirilir. Fonksiyonel benzetim sonuçlarının doğruluğu ispat edildikten sonra, tasarlanan toplayıcı donanımı FPGA içerisine gömülmüştür. Bu şekilde

FPGA içerisinde gerçekleştirilen 4 bitlik toplayıcı donanımının giriş pinlerine dışarıdan 4' er bitlik işaretler uygulanarak çıkışta elde edilecek 4 bitlik sayının binary kodlarını gösteren elektriksel işaretler elde edileblir.

Flow Status	Successful - Sat Jan 30 14:01:01 2010
Quartus II Version	8.0 Build 215 05/29/2008 SJ Web Edition
Revision Name	toplama
Top-level Entity Name	toplama
Family	Cyclone III
Device	EP3C40F484C7
Timing Models	Final
Met timing requirements	N/A
Total logic elements	5 / 39,600 (< 1 %)
Total combinational functions	5 / 39,600 (< 1 %)
Dedicated logic registers	0 / 39,600 (0 %)
Total registers	0
Total pins	13 / 332 (4 %)
Total virtual pins	0
Total memory bits	0 / 1,161,216 (0 %)
Embedded Multiplier 9-bit elements	0 / 252 (0 %)
Total PLLs	0 / 4 (0 %)

Şekil 3.10 Quartus'ta 4 bitlik toplayıcının derlenmesi ve FPGA'ya gömülmesi sonuç raporu

Şekil 3.10'daki rapordan da görüldüğü gibi, 4 bitlik toplayıcı donanımının FPGA içerisinde oluşturulması için 5 adet LE kullanılmıştır.

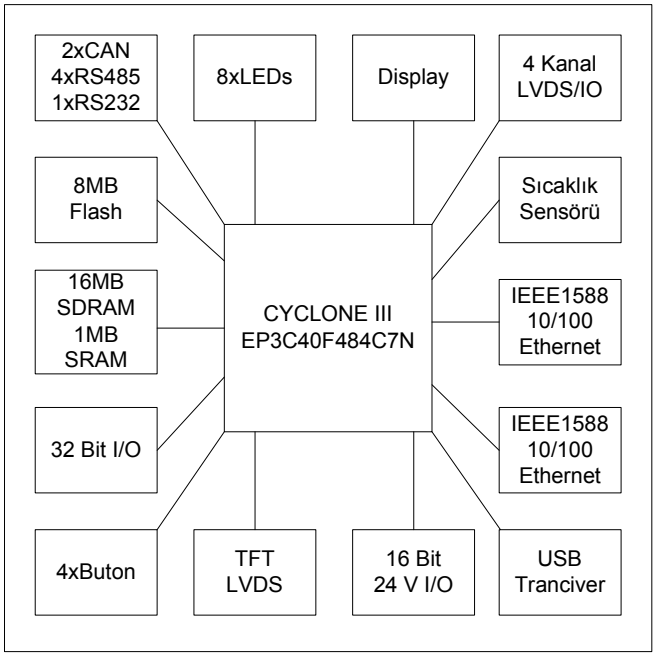
Verilen bu örnek donanımsal olarak gerçekleştirilmesi istenen bir lojik devrenin, VHDL dilinde kodlanarak bir arayüz vasıtasıyla sembolik forma dönüştürülmesi ve giriş çıkış pinlerinin atanmasından sonra, FPGA içerisine gömülüp donanımsal bir devrenin oluşturma aşamalarını gösteren basit bir uygulamadır.

4. GERÇEK ZAMANLI STS İÇİN KULLANILAN GEREÇLER

Bu bölümde gerçek zamanlı STS'nin tasarımı için kullanılacak olan FPGA temelli platformun tanıtımı yapılarak, tasarım için gerekli olan donanımsal ve yazılımsal araçlar hakkında bilgi verilecektir.

FPGA'ların defalarca programlanabilme imkanına sahip olmaları en büyük avantajlarıdır. Bu yüzden en basit devrelerden, en karmaşık devrelere kadar donanımsal gerçekleştirmelerde önemli bir platform olarak kullanılırlar. Bu projede oluşturulan gerçek zamanlı devreler VHDL dili ile yazılmış olup, Altera Cyclone III tabanlı DBC3C40 uygulama geliştirme kartı üzerinde gerçekleştirilmiştir. Kart üzerindeki donanımsal birimler ve EP3C40F484C7N FPGA entegresinin yerleşim planı Şekil 4.1 de gösterilmektedir. Şekilden de görüldüğü gibi kart üzerinde FPGA entegresi ile haberleşebilen birtakım çevre birimleri vardır. Bu çalışmada bizim için önemli olan ise IEEE 1588 10/100 fiziksel Ethernet ara birimidir.

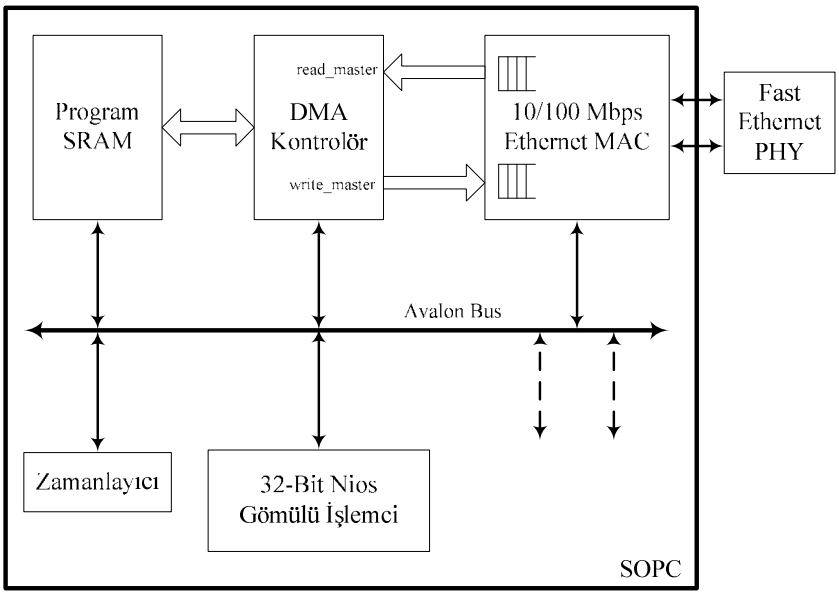
Bu geliştirme kartı kullanılarak gerçekleştirilmiş olan STS'nin donanımsal blok şeması Şekil 4.2'de verilmiştir.



Şekil 4.1 DBC3C40 kartının genel görünüşü

Bilgisayar ağı ile STS, fiziksel ethernet arabirimi üzerinden birbirine bağlıdır. Ağdan gelen veya giden paketlerin, OSI modelinin veri bağı katmanında kullanılacak biçime dönüşümü kart üzerindeki DP83640 fiziksel arabirim entegresi (Fast Ehernet PHY) ile gerçekleştirilir.

Veri bağı katmanının MAC (Media Access Control) alt katmanındaki tüm görevleri gerçekleştirmek için, Şekil 4.2' deki MAC modülü kullanılır. Gönderilen veya alınan veriler burada çerçeve formuna çevrilir. Ağdan gelen çerçeveler kullanıcı tarafından erişilebilen bir hafıza bölgesine kaydedilir. Aynı zamanda kullanıcı tarafından hafıza arabiriminde oluşturulan veriler, çerçeve formuna dönüştürülerek fiziksel arabirime gönderilir.

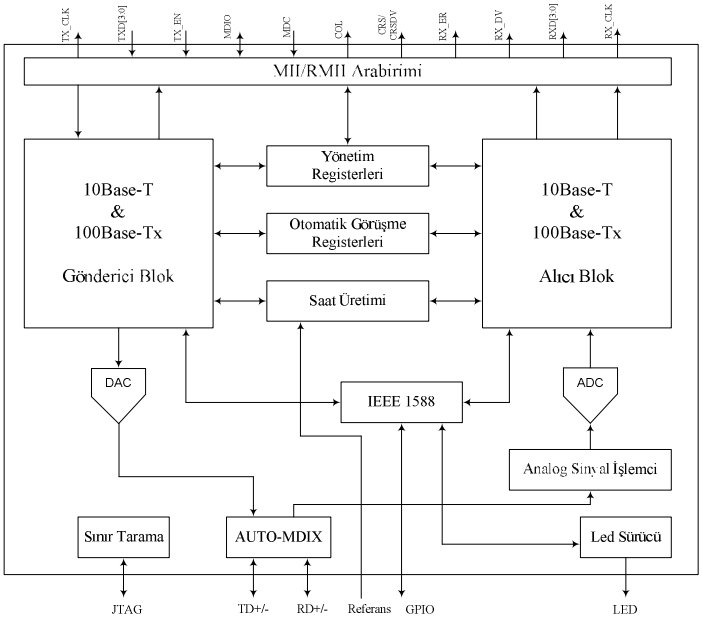


Şekil 4.2. Gerçekleştirilen STS'nin blok yapısı [40]

Bilgisayar ağından gelen paketlerin alınması, işlenmesi ve gönderilmesi için tasarlanan STS'nin donanımsal yapısı, fiziksel Ethernet arabirimi hariç FPGA içerisinde programlanarak gerçekleştirilmiştir. FPGA'da programlanarak gerçekleştirilen gömülü işlemci donanımı, MAC modülünü de yöneterek paketlerin alınması, işlenmesi ve gönderilmesi için kullanıcı tarafından dışarıdan programlanır. Gömülü işlemci, MAC modülü ve hafıza birimi Doğrudan Hafızaya Erişim (Direct Memory Access, DMA) ile kontrol edilen bir veri yoluyla (Avalon Bus) haberleşir. Bu birimlerin açıklanması konunun anlaşılması açısından önemlidir.

4.1. Fiziksel Arabirim

Verilerin seri bitler düzeninde iletildiği Fiziksel katman, OSI modelinin ilk katmanıdır. Bu katmanda ağ kablosu ile ağ elemanları arasında iletişim kurulur. Fiziksel iletimle ilgili olarak IEEE 802.3, 802.4, 802.5 standartları kullanılır. Fiziksel katman kablolama yapısını ve ağ kartına bağlanmayı sağlayan birimleri içerir. Ayrıca fiziksel katman iletim ortamı işaretlerini kontrol eder. Bu katmanda ağ kablosu ile ağ elemanları arasında iletişim kurulur. Fiziksel katmanda verilerin sayısal olarak iletimi, koaksiyel kablo, UTP yada fiber optik kablo üzerinden yapılır. Bu projede kullanılan DBC3C40 FPGA geliştirme kartı, OSI katmanlı yapının fiziksel katman görevlerini yerine getiren arabirimlere sahiptir. RJ45 konnektörü ile ağa fiziksel olarak bağlantısının sağlandığı bu arabirim, National firmasının DP83640 kodlu entegresidir. Şekil 4.3 DP83640 entegresinin iç yapısını göstermektedir.

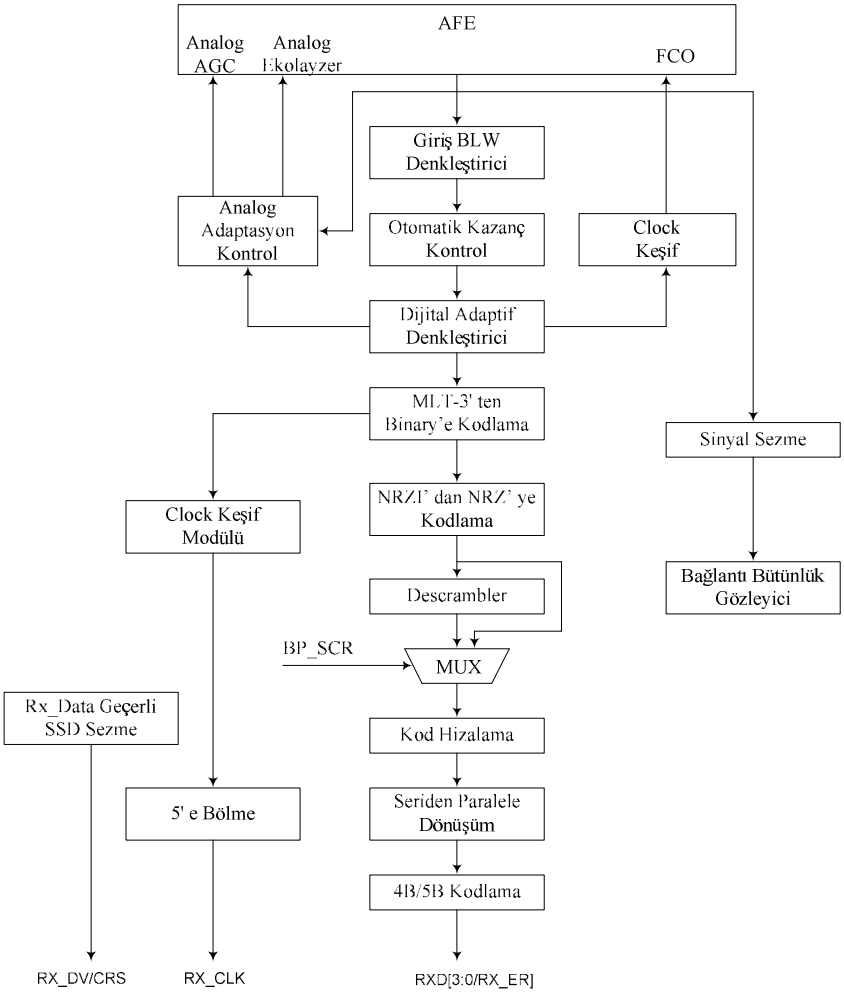


Şekil 4.3 DP83640 entegresi iç yapısı

DP83640 entegresi, ağ ortamından alınan işaretleri gerekli işlemlerden geçirdikten sonra veri bağı katmanına iletilir. Ağ ortamına gönderilecek verileri ise veri bağı katmanından alarak gerekli işlemlerden geçirdikten sonra fiziksel ortama iletir.

RJ45 konnektörü ile doğrudan bağlı TD+/- ve RD+/- pinlerinden gelen veya giden bit düzeyindeki veriler, analog/dijital dönüştürme ile ya alıcı bloğa yada gönderici bloktan

alınan bit düzeyindeki veriler dijital/analolog dönüştürücüler ile fiziksel ortama iletilir. Şekil 4.4 ortamdan bağımsız (Media Independent Interface/ Reduce Media Independent Interface, MII/RMII) arabirime yada veri bağı katmanına gönderilecek verinin alıcı bloğunda yapılması gereken işlemleri göstermektedir.

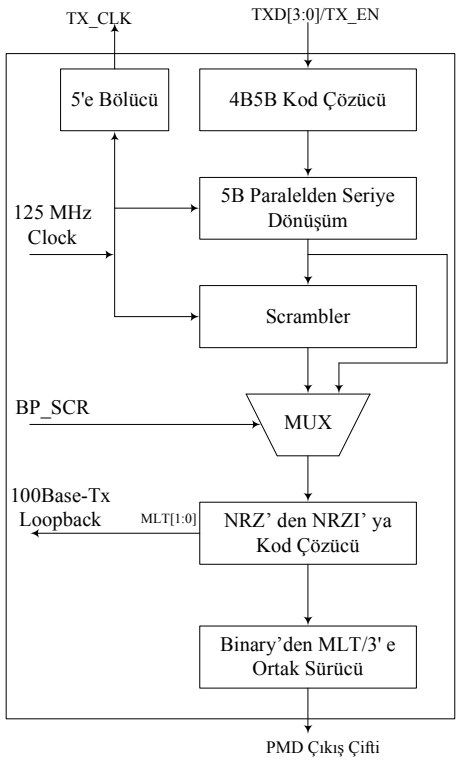


Şekil 4.4 Alıcı Blok

Alıcı blok bir clock çevriminde fiziksel ortamdan alınan ve ADC ile dönüşümü yapılmış bit düzeyindeki verileri NRZI (Non Return to Zero Inverted) formatından NRZ (Non Return to Zero) formatına çevirir. Bu dönüşümden sonra seri olarak gelmiş olan veriler, seriden paralele çevrilir. Paralele çevrilmiş veri, 4B/5B kodlaması ile MII/RMII arabirimine gönderilir.

Gönderici blok ise MII/RMII arabiriminden 1 clock çevrim zamanda aldığı işaretleri veya veriyi öncelikle 4B/5B dekodlaması yaparak, paralel veriyi seriye çevirir. Bu

aşamadan sonra elde edilen veri NRZ' den NRZI' ya çevrilerek fiziksel ortama iletilmek üzere RJ45 konnektörünün sahip olduğu TD+/- pinlerine gönderir. Şekil 4.5 gönderici blok yapısını göstermektedir.



Şekil 4.5 Gönderici blok

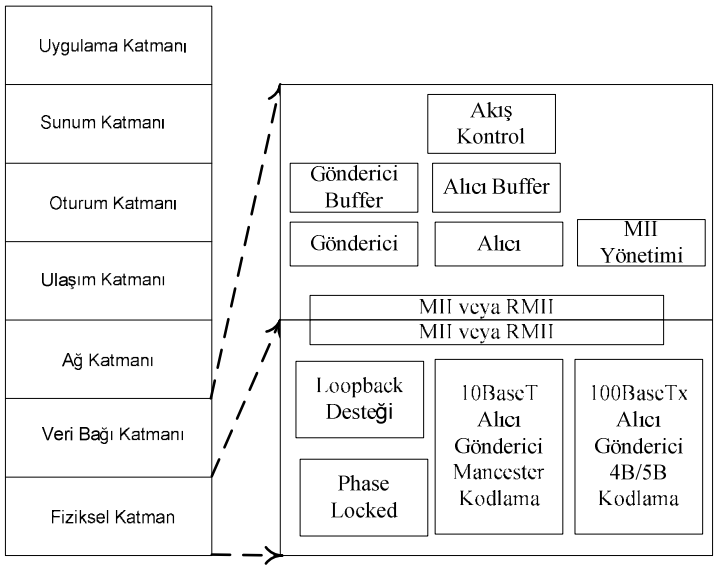
4.2. Veri Bağı Katmanı ve 10/100 Ethernet IP MAC Core

Fiziksel katmanın doğrudan iletişim halinde bulunduğu OSI katmanı, veri bağı katmanıdır. Ethernet çerçeve formatı bu katmanda tanımlıdır. Ethernet, veri haberleşmesi ağları içinde, en başarılı ve en çok kullanılan LAN teknolojsidir. 3 Mbit/sn paylaşımlı şebeke yapılarından, bugün en son gelinen durum itibariyle gigabit veri iletimi hızları yakalanabilmiştir. Ağ kullanımının kolay olması, teknik olarak basit ve kurulu sistemlerle uyumlu olması, Ethernet'i gözde bir teknoloji yapmıştır. IEEE 802.3 olarak bilinen Ethernet standartları, yerel alan ağlarının kurulması, kullanılabilecek fiziksel ortam tipleri ve bunların bir arada nasıl kullanılabileceğini tanımlar.

Ethernet teknolojisi 3 temel unsur üzerine kurulmuştur. Bunlardan ilki haberleşme cihazları arasındaki Ethernet işaretlerini taşımaktan sorumlu fiziksel ortam, ikincisi fiziksel ortamın kullanılması kurallarının düzenlendiği erişim standartları, üçüncüsü ise haberleşme

kanalları üzerinden gönderilecek bit dizilerinin biçimini tanımlayan Ethernet çerçeve formatıdır. Ethernet teknolojisinin temelini Ethernet çerçeve yapısı oluşturur ve tüm Ethernet türevleri için bu yapı değişmez. Bu da ağ üzerinde, veri paketlerinin herhangi bir değişikliğe uğrama ihtiyacını ortadan kaldırarak çok büyük avantaj sağlar.

Yeni geliştirilen veya değişiklik yapılan bir takım standartlar ile Ethernet ağları üzerinde, tutarlı akış kontrolleri teknikleri, yüksek hızlarda ve güvenli bir şekilde veri aktarımı, ses ve görüntü işaretlerinin taşınabilmesi ve ayrıca birtakım servis kalitesi parametreleri tanımlanabilir hale gelmiştir. Şekil 4.6 veri bağı katmanı ve fiziksel katman yapısını göstermektedir.



Şekil 4.6. OSI başvuru modeline göre IEEE LAN standartları

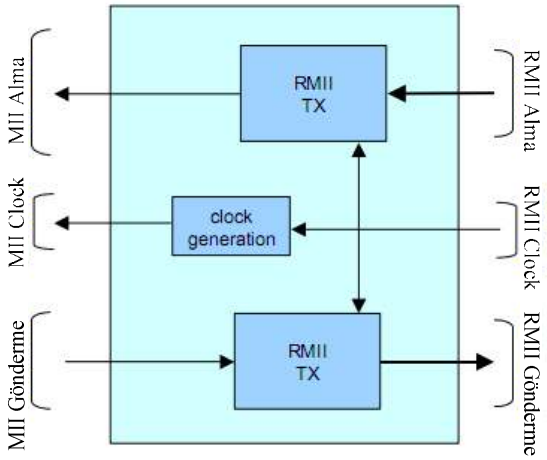
Veri bağı katmanı düğümler arasında çerçevelerin güvenli olarak gönderilmesini ve alınmasını sağlar. Bu katman 2 alt katmandan, LLC (Logical Link Control) ve MAC'dan (Media Access Control) oluşur. MAC alt katmanı, çerçevenin yola erişimini, çerçeve hata kontrolünü yapar. Çerçeveleri fiziksel katmana aktarır. Alıcı tarafta da bu işlemlerin tersini gerçekleştirerek veriyi, bir üst katmana yani LLC'ye aktarır.

LLC alt katmanı, ağ katmanına geçiş görevi görür. Protokole özel mantıksal portlar oluşturur (Source Service Access Points - SSAPs ve Destination Service Access Points - DSAPs v.b). Böylece kaynak makine de ve hedef makine de aynı protokollerin iletişiminin yapılmasına olanak sağlar. LLC ayrıca veri paketlerinden bozuk gidenlerin tekrar gönderilmesinden ve çerçevelerin sıralanmasından, bağlantıların yönetilmesinden sorumludur. Akış kontrolü de LLC'nin görevidir[54].

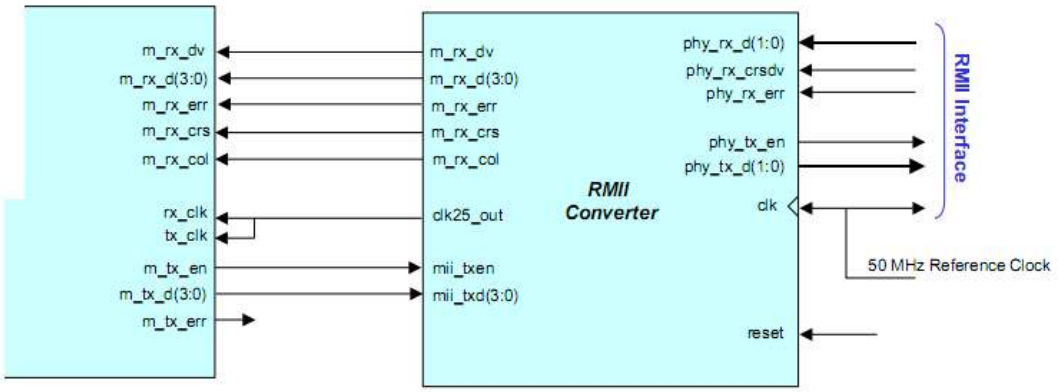
MAC katmanı ile fiziksel katman arasındaki iletişim, fiziksel ortamdan bağımsız ara bağdaştırma(MII) katmanı ile gerçekleştirilir. Her clock çevriminde 4 bitlik veri MII'ya taşınır. MII işaret pinleri, verici ve alıcı bitleri, saatler, veri hata kontrolü işaretlerinden ibarettir. Ayrıca apın aktif olup olmadığını, çatışmaların sezilmesini ve yönetim arayüz işaretlerini de destekler. Veri bağı katmanının PHY arabirime bağlanması ve işlem yapması için aşırı pin sayısını azaltmak gerekebilir. Bu durumda azaltılmış MII yada RMII, kullanılabilir. Bu özelliğin gelişmesi, yalnızca 802.3'ün değil aynı zamanda 802.3z Gigabit Ethernet'in de çok hızlı bir şekilde gelişmesinin sonucudur. RMII, 10 Mbit/sn ve 100 Mbit/sn veri hızlarını tam dupleks işlemleri destekler. RMII, MII'nın aksine eşzamanlı olarak işlem yapması, tekrarlayıcı ve diğer cihazların aynı veri ve alıcı saatlerinde çalışmasını sağlar. Ayrıca RMII, MII'nın iki katı saat hızında 50 MHz'de çalışır. Bundan başka veri ve kontrol yollarının kullanımını azaltır. IEEE 802.3u standardına bir alternatif olarak gerçekleştirilen bu arabirimin amacı, azaltılmış pin sayısı ile düşük maliyetli olmasıdır. RMII aşağıdaki karakteristiklere sahiptir.

1. 10 Mbps ve 100 Mbps veri hızını destekler.
2. MAC' dan PHY' ye bir tek saat darbesi ile veri aktarımı sağlar.
3. Birbirinden bağımsız 2 bitlik gönderme ve alma veri yoluna sahiptir.
4. TTL ve CMOS ile uyumlu çalışabilir.

Aşağıdaki Şekil 4.7 RMII-MII dönüştürücü blok diyagramını Şekil 4.8 ise MII arabirimi ile RMII arabirimi arasındaki dönüşüm için gerekli pinleri ve Tablo 4.1 ise pin tanımlamalarını göstermektedir.



Şekil 4.7 MII-RMII dönüşümü



Şekil 4.8 MII-RMII Dönüşüm pinleri

Tablo 4.1 RMII ve MII pin tanımları

Sinyal adı	Tip	Tanım
Reset	Giriş	Reset sinyali, aktif yüksek
Clk	Giriş	50 Mhz RMII saat sinyali
ena_10	Giriş	100 Mbps için lojik 0, 10 Mbps için lojik1 seviyesi
clk25_out	Çıkış	MAC alma ve gönderme işlemlerinde 100 ve 10Mbps hızları için 25 veya 2.5 Mhz saat işareti
RMII Arabirimi		
phy_tx_en	Çıkış	RMII iletim veri kontrol pini
clk	Giriş	50 Mhz RMII Referans saat işareti
phy_tx_d(1:0)	Çıkış	RMII gönderme veri yolu (PHY' ye)
phy_rx_crsdv	Giriş	RMII alma veri kontrolü ve carrier sense
phy_rx_d(1:0)	Giriş	RMII alma veri yolu (PHY'den)
phy_rx_err	Giriş	Hata gösterim biti
MII Arabirimi		
m_rx_d(3:0)	Çıkış	MII alma veri yolu
mrx_dv	Çıkış	Eğer bu bit 1 ise Alma veri yolu aktiftir.
m_rx_err	Çıkış	Hata gösterim biti
m_rx_crs	Çıkış	Sadece Half Duplex mod için Carrier Sense gösterim biti
m_rx_col	Çıkış	Sadece Half Duplex modu için çarpışma sezme biti
m_tx_d(3:0)	Giriş	MII gönderme veri yolu
m_tx_en	Giriş	Eğer bu bit 1 ise veri yolunda geçerli veri olduğunu gösterir

OSI modelinin veri bağı katmanı, fiziksel ortamdan aldığı verileri hatalardan arındırılmış bir iletişim kurulması işlemini yapar. Bu katmanda alınan/gönderilen veriler üzerinde çerçeveleme, eş zamanlama, akış denetimi, hata denetimi, adresleme ve bağlantı yönetimi v.b gibi işlemler gerçekleştirilir.

Bu işlemleri gerçekleştirmek için kullanım lisansı ile alınmış olan 10/100 Ethernet IP MAC Core modülü bu bölümde tanıtılacaktır. MAC modülü VHDL diliyle yazılmış ve veri bağı katmanındaki görevleri yapması için hazırlanmış, konfigüre edilmesi gereken bir programdır. Bu yazılım uygun ortamlarda derlenip, FPGA içerisine gömülerek donanımsal bir modül elde edilir. İlerleyen bölümlerde bu işlemler detaylı olarak açıklanacaktır. Bahsedilen MAC modülünün veri bağı katmanında gerçekleştirdiği fonksiyonlar aşağıda özetlenmiştir [55].

1. 802.3 standardında belirtilen çerçeve yapısını alma/gönderme için; premable/SFD üretimi, çerçeve üretimi, çerçeve doldurma bitleri üretimi, CRC üretme ve doğrulama.
2. Dinamik olarak 10 Mbps ve 100 Mbps'de çalışabilme.
3. 10/100 Mbps hızı için hem full duplex hemde half duplex modunda çalışabilme.
4. MII arabirimi ile 25 Mhz hızında Fast Ethernet yapısında çalışabilme.
5. Diğer ortamdan bağımsız yapıları desteklemek (RMII v.b gibi).
6. Avalon sistem arabirimi ile uyumlu çalışabilme özelliği. Veri ve kontrol için ayrı portlar.
7. Herhangi bir Ethernet çerçeve yapısının desteklenmesi (SNAP/LLC gibi).
8. CRC üretimi ve çerçevedeki ilgili alana ekleme CRC doğrulaması.
9. Full duplex modunda işletildiği zaman otomatik olarak Durdurma çerçevesi oluşturabilme.
10. Trafik akışını kontrol edebilmek için, kullanıcı tarafından geliştirilen uygulamalar ile durdurma çerçevesi oluşturabilmek.
11. Durdurma çerçevesi alabilme.
12. Half duplex modunda işletildiği zaman çarpışma sezebilme ve çerçevenin yeniden iletiminin gerçekleştirilmesi.
13. IEEE 802.1Q' ya göre VLAN çerçeve formatının desteklenmesi.
14. Unicast MAC çerçeve desteği.

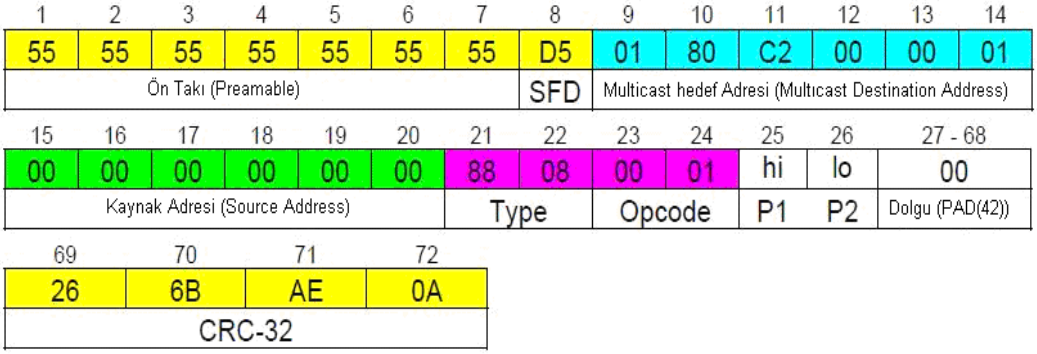
15. Multicast MAC çerçevelerinin alınmasını sağlamak için 64'lük Hash tablosu oluşturabilme.
16. CPLD veya ASIC gerçekleştirmelerinin desteklenmesi.
17. Altera SOPC builder ile bütünleşik çalışabilme.
18. Hem C dili yazılım sürücülerini hem de TCP/IP ağ kütüphanesini destekleme.

Modülün alacağı veya göndereceği çerçeve formatı IEEE standartlarında olup, Tablo 4.2 'deki gibi tanımlanır.

Tablo 4.2. MAC çerçeve format özeti

7 octets (bayt)	Preamble (Öntakı)
1 Octets (bayt)	Start-Frame Delimiter SFD (Çerçeve başlangıç belirteci)
6 Octets (bayt)	Destination Addres (Hedef Adres)
6 Octets (bayt)	Source Address (Kaynak Adres)
2 Octets (bayt)	Lenght/Type (Uzunluk/Tip)
0..1500/9000 Octets (bayt)	Payload data (faydalı yük verisi)
0..46 Octets (bayt)	PAD (Dolgu)
4 Octets (bayt)	Frame Check Sequence (Çerçeve kontrol dizisi)

Bir Ethernet çerçevesinin uzunluğu 64 İle 1518 Byte arasında olup örnek çerçeve yapısı Şekil 4.9'de verilmiştir.

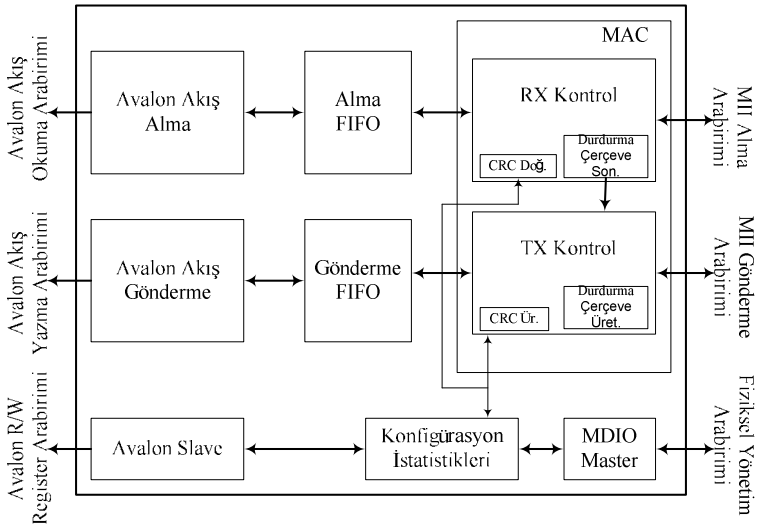


Şekil 4.9. Çerçeve örneği

Her bir çerçeve 7 baytlık Öntakı bitleri ile başlar. Öntakılar, fiziksel katman ile senkronize olması için kullanılır. Çerçevenin bir sonraki alanı SFD(Start-Frame Delimiter) bir bayt olup, bu alan çerçevenin başladığını göstermek için kullanılır. Çerçevenin

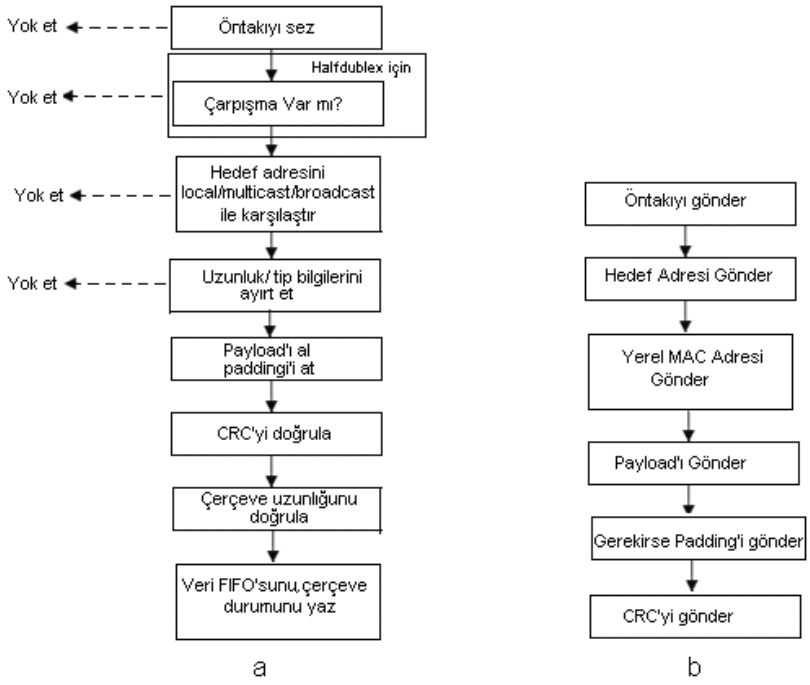
gönderileceği ve alınacağı düğümlerin fiziksel adresleri, 6' şar baytlık MAC adres alanını oluşturur. Gönderilen veya alınan paketin uzunluk ve tip alanı 2 bayt uzunluğunda olup, çerçevenin bu alanı LLC katmanındaki verinin sayısını belirtir Ayrıca bu alan ethernet protokollerinde tip olarak ta kullanılır. Çerçevenin başlık yapısı oluşturulduktan sonra çerçeve içeriğinin, veri alanının belirtilmesi gereklidir. Eğer çerçeve 64 bayttan küçükse çerçevenin 64' bayta tamamlanması için doldurma bitleri kullanılır. Doldurma bitlerinin sayısı padding alanı ile belirtilir. Bu bitler çerçevenin alındığı tarafta çerçeveden çıkartılarak dikkate alınmaz. Ayrıca 4 baytlık FCS alanı da çerçevenin doğru alındığını veya gönderildiğini göstermek için çerçevenin en sonuna eklenir. Bu alan hatalı paketlerin sezilmesinde kullanılır.

Şekil 4.10 MAC modülünün blok yapısını göstermektedir. Kullanıcıların ağdan gelen/giden çerçeveler üzerinde işlem yapabilmesi için modül, FPGA içerisine gömülebilen bir soft işlemci ve hafıza birimleriyle, yazma/okuma arayüzleri vasıtasıyla Avalon bus üzerinden iletişim halinde olmalıdır. Aynı zamanda fiziksel arabirim ile de MII alma/gönderme ara yüzüyle bağlantılıdır.



Şekil 4.10. MAC Core modülünün blok yapısı [51]

Çerçevelerin hafıza modülüne (SRAM, On-Chip Memory v.b) alınması veya hafızadaki çerçevelerin fiziksel katmana iletilmesi süreçleri, Şekil 4.10' da görüldüğü gibi Avalon Streaming Receive/Transmit, Receive/Transmit FIFO ve Rx/Tx Kontrol blokları üzerinden olur. Rx kontrol ve Tx kontrol ünitelerinin fonksiyonları Şekil 4.11' deki akış şemalarındaki gibidir.



Şekil 4.11 a) RX kontrol, b) TX kontrol fonksiyonlarının akış diyagramı [55]

Çerçeve alma/gönderme işlemi sürecinde, gerek gömülü işlemci tarafından gerekse dışarıdan kontrol edilebilmesi için MAC modülünde, konfigüre edilebilen 256 adet 32 bitlik gömülebilir kayıtçı vardır. Bu kayıtçıların haritası ve tanımlamaları Tablo 4.3'de özetlenmiştir. Tablo 4.3'de görüldüğü gibi 0x000 ile 0x05C adresleri arasındaki 18 adet kayıtçı, MAC modülünün konfigürasyonu için kullanılır.

Tablo 4.3 MAC modül kayıtçılarının haritası

Adres	Bölüm	Tanımlama
0x000-0x05C	MAC Konfigürasyon	MAC modülünü konfigüre etmek için kayıtçılar
0x060-0x0DC	İstatistik Kayıtçıları	Trafik istatistiklerinin toplandığı sayıcılar
0x0E0-0x0FC	İşlem Kontrolleri	Rx/Tx FIFO ve kontrol FIFO Oku/Yaz durumları
0x100-0x1FC	Çoklu Hash Tablosu	64 lük Hash tablo girişi
0x200-0x27C	MDIO-0	İlk fiziksel cihazın MDIO adresi
0x280-0x2FC	MDIO-1	İkinci fiziksel cihazın MDIO adresi
0x300-0x31C	MAC Adresleri	Unicast MAC adresleri
0300-03FC	Saklı Tutulmuş	Kullanılmıyor.

32 bitlik bu kayıtçıların adresleri ve tanım özetleri Tablo 4.4' de verilmiştir. İstatistik kayıtçılarının görevi, alınan, gönderilen, hatalı alınan veya gönderilen çerçevelerin sayısını veya bayt olarak ne kadar verinin alındığını, çerçeve boyutlarına göre istatistik tutma v.b genel bilgileri tutar.

Tablo 4.5' de özetleri verilen işlem kontrol kayıtçıları, çerçeve alma ve gönderme işlemlerinde kullanılır. Çoklu Hash tablosu, multicast adresler için kullanılan kayıtçılara sahiptir. Ayrıca fiziksel arabirime ait adresler ise MDIO-0/-1 kayıtçılarında tutulur. 0x300-0x31C arasındaki kayıtçılar Unicast adresler için kullanılırken, diğer kayıtçılar ise saklı tutulmuştur. MAC modülünün VHDL kodlarının derlenip FPGA'ya gömülmesi esnasında bu kayıtçılar da modül içerisinde oluşturulmuş olur. İlerleyen bölümlerde bu kayıtçıların nasıl konfigüre edileceği açıklanacaktır.

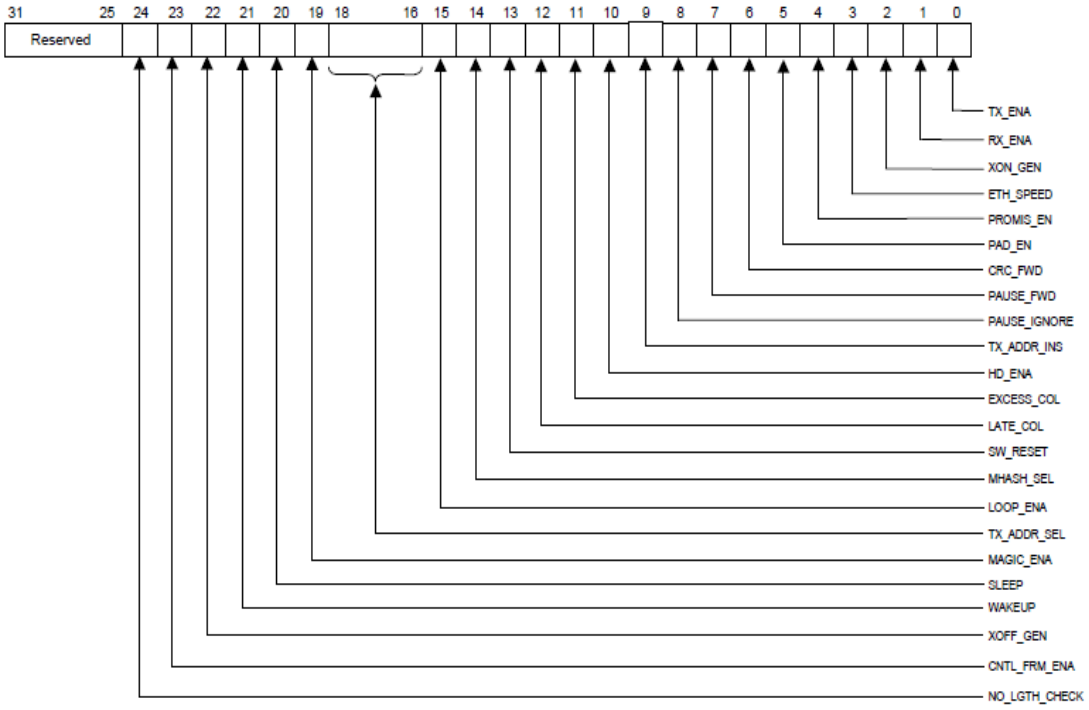
Tablo 4.4. MAC modülünün konfigürasyon kayıtçıları

Adres	Kayıtçı	Tanımlama
0x000	REV	32 Bitlik MoreThanIP ve müşteri kodu
0x004	SCRATCH	Hafıza yerleşimini test etmek için. R/W
0x008	COMMAND_CONFIG	Çekirdeği kontrol ve konfigüre etmek için. RW
0x00C	MAC_0	R/W MAC adresinin en anlamlı 32 biti
0x010	MAC_1	R/W MAC adresinin en anlamsız 16 biti
0x014	FRM_LENGHT	R/W İlk 14 bit maksimum çerçeve uzunluğu.
0x018	PAUSE_QUANT	Durdurma çerçevesi oluşturmak için. R/W
0x01C	RX_SECTION_EMPTY	Alma FIFO' nun eşik değerini ayarlar. R/W
0x020	RX_SECTION_FULL	Store&Forward modunda kullanılır.
0x024	TX_SECTION_EMPTY	Gönderme FIFO' nun eşik değerini ayarlar. R/W
0x028	TX_SECTION_FULL	Gönderme FIFO' nun eşik değerini ayarlar. R/W
0x02C	RX_ALMOST_EMPTY	Alma FIFO' nun eşik değerini ayarlar. R/W
0x030	RX_ALMOST_FULL	Alma FIFO' nun eşik değerini ayarlar. R/W
0x034	TX_ALMOST_EMPTY	Gönderme FIFO' nun eşik değerini ayarlar. R/W
0x038	TX_ALMOST_FULL	Gönderme FIFO' nun eşik değerini ayarlar. R/W
0x03C	MDIO_ADDR0	0. fiziksel cihazın MDIO Adresi
0x040	MDIO_ADDR1	1. fiziksel cihazın MDIO Adresi
Diğer	Saklı Tutulmuş	

Tablo 4.5. İşlem kontrol kayıtçıları

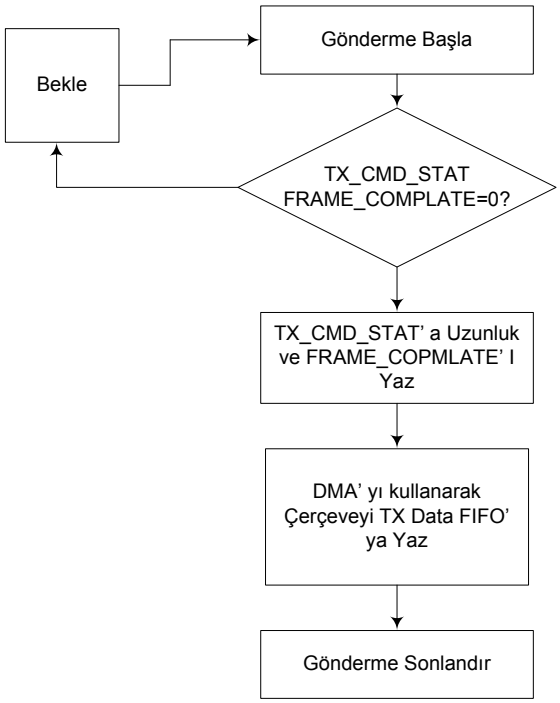
Adres	Kayıtçı	Tanımlama
0x0E0	AVL_STATUS	FIFO durum göstergeleri
0x0E4	IRQ_CONFIG	Kesme konfigürasyonu
0x0E8	TX_CMD_STAT	Çerçeve gönderme kontrolu için çerçeve komut ve durum bilgisi
0x0EC	RX_CMD_STAT	Çerçeve alma kontrolu için alama komut ve durum bilgisi

Tablo 4.4' te verilen 0x008 Adresli **Command_Config** kayıtçısının yapısı Şekil 4.12' de verilmiştir. Ethernet MAC modülünün çerçeve alma/gönderme için, bu kayıtçının konfigüre edilmesi gereklidir.



Şekil 4.12. COMMAND_CONFIG kayıtçı yapısı

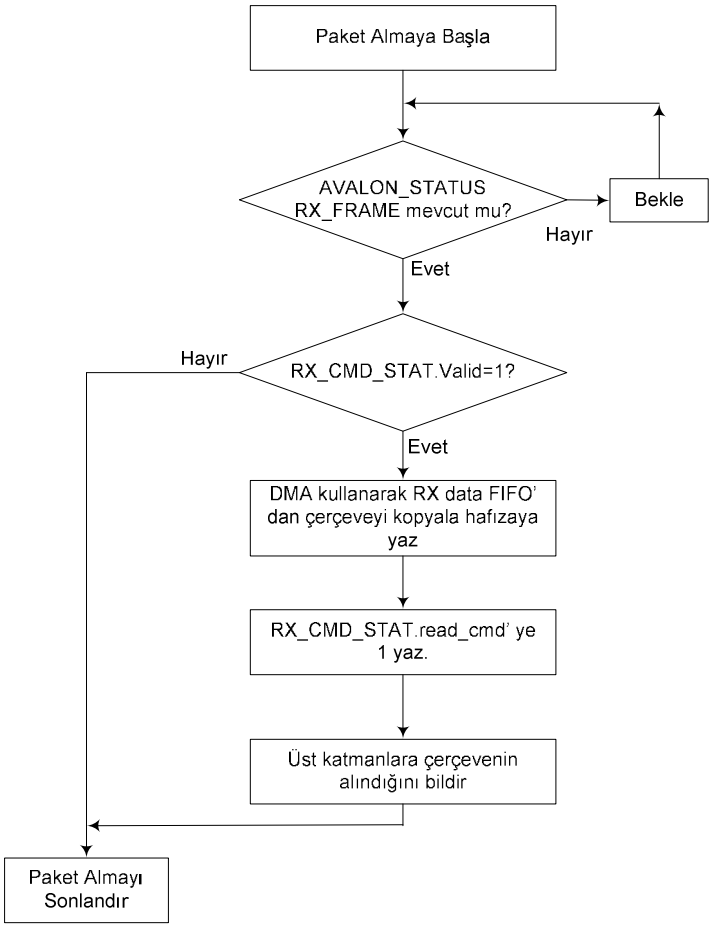
Bu açıklamalardan sonra MAC modülünün, çerçeve gönderme ve alma prosedürlerini nasıl gerçekleştirdiği, Şekil 4.13 ve 4.14 yardımıyla özetlenebilir.



Şekil 4.13. Çerçeve gönderme akış diyagramı

Çerçeve gönderme işlemi iki adımda gerçekleştirilir. Her çerçeve gönderme işleminde bu iki adım gerçekleştirilmek zorundadır. Bu adımlardan ilki Tablo 4.5'de verilen TX_CMD_STAT kayıtçısının, çerçeve uzunluğunu gösteren bitler (15 bit) ve çerçeve tamamlama (FRAME_COMPLATE) biti (1 bit) uygun değerlerle yüklenir. İkinci adımda ise DMA kontrolör yardımıyla hafıza uzayında (SRAM, On Chip Memory v.b) oluşturulmuş çerçeve, MAC modülünün içerisindeki Transmit FIFO'ya kopyalanır. Bu süreç Şekil 4.13'de verilmiştir. Bundan sonraki süreçte, çerçeveye, Tx kontrol ünitesinde gerekli kontroller ve eklemeler (CRC bitleri v.b) yapılarak çerçeve fiziksel arabirime gönderilir.

MAC modülündeki çerçeve alma işlemi üç adımda gerçekleştirilir. Her çerçeve alma işleminde bu üç adım gerçekleştirilmek zorundadır. Bu adımlardan ilkinde Tablo 3.5' deki AVALON_STATUS kayıtçısının RX_FRAME_AVAILABLE biti kontrol edilir. Çerçeve mevcutsa ikinci adıma geçilir. Bu adımda, Rx kontrol ünitesinden alınan çerçeve Receive FIFO'ya yazılır. DMA kullanılarak Receive FIFO'daki çerçeve, Avalon veriyolu üzerinden hafızaya (SRAM, On-Chip Memory v.b) yazılır. Son adımda ise çerçevenin alındığını gösteren ilgili kayıtçılar (çerçevenin alınma işlemi bitene kadar), yeni bir çerçevenin alınmasını önlemek için uygun değerlerle yüklenir. Bu süreç Şekil 4.14'de verilmiştir.



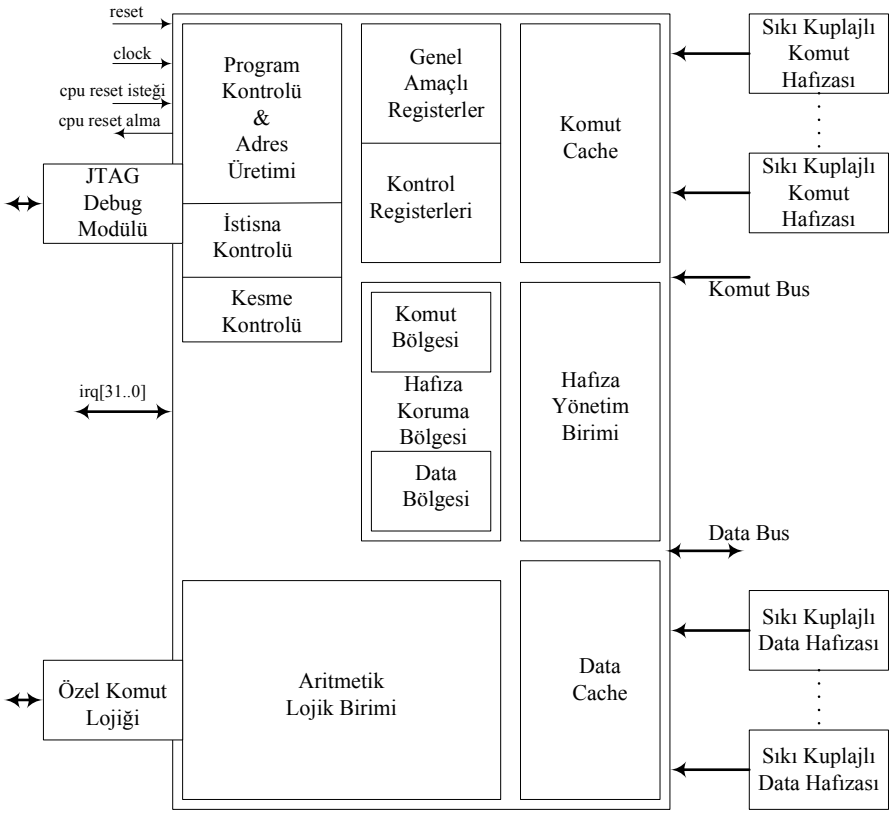
Şekil 4.14 Çerçeve alma akış diyagramı

4.3. Gömülü İşlemci

Bir soft işlemci, genellikle VHDL, Verilog gibi donanım tanımlama dili ile yazılmış ve mikroişlemcilerin tüm özelliklerini yerine getiren bir yazılımdır. Bu yazılım, FPGA gibi programlanabilir donanımlara gömülerek gerçek (donanımsal) mikroişlemci elde edilir. Soft işlemcilerin, gerçek işlemcilere karşı üstünlükleri; son kullanıcının uygulamaya özel basit veya karmaşık mimariye sahip işlemci gerçekleştirebilmesi, işlemcinin güncellenebilme imkanı, tasarımının, güncellenmesinin ve değiştirilmesinin basitliği, FPGA içerisindeki diğer birimler ile düşük gecikmeli iletişim gerçekleştirebilmesidir. Soft işlemcilerin hard işlemcilere (Tek bir entegre devre olarak oluşturulan işlemci) göre daha düşük saat frekansı ile çalışmaları, onların performanslarının düşük olacağı anlamına gelmez. Herhangi bir komutu tek bir komut çevriminde işleyebildiği için hard işlemcilere göre performansı daha iyidir. Örneğin, 32 bitlik çarpma işlemini çarpma komutuyla tek bir

komut çevriminde yapabilir. Soft işlemcilerin dezavantajları ise herhangi bir işe atanmış ASIC donanım çözümüne göre daha pahalıdır. Eğer FPGA içerisindeki diğer birimlerle iletişimi iyi optimize edilemezse yavaş çalışabilir. FPGA'lar için hazırlanmış soft işlemi yazılımlarından en çok kullanılanları, Xilinx firmasının Microblaze, Picoblaze, 8051'i ve Altera firmasının Nios işlemcileridir. Bu projede Nios soft işlemcisi kullanılmıştır [56].

Nios 3.0 işlemci, RISC mimarisi tabanlı bir mikroişlemci özelliğinde olup 16 ve 32 bit desteklidir. Tasarımda kullanılan işlemci mimarisi Şekil 4.15'de verilmiştir. Bu işlemcinin temel özellikleri aşağıda özetlenmiştir. İşlemci Von-Neuman mimarisinde olduğu gibi veri ve komut hafızalarına (Data cache, Instruction cache) erişmek için ayrı ayrı veriyolları (Data Bus, Instruction Bus) kullanır.



Şekil 4.15 NIOS işlemcinin iç yapısı

NIOS CPU genel amaçlı 512 dahili kayıtçı (General Purpose Registers, Control Registers) içerir. 16 bit genişliğinde komutlar kullanır. Komut seti, yükleme ve depolama komutlarını içerir. Kullanıcılar NIOS aritmetik lojik birimi (Arithmetic Logic Unit) ile doğrudan mantıksal ve matematiksel işlemleri kullanabilir. Nios işlemci beş durumlu

pipeline özelliğine sahiptir. Quartus 8.0 yazılımıyla bütünleşik SOPC ortamı kullanıcıya her iki hafıza birimini ayrı ayrı veya beraberce kullanıma izin verilir. Ayrıca işlemci kaydırma komutlarını gerçekleştirebildiği gibi çarpma işlemlerini de gerçekleştirebilen ayrı bir çarpma donanımına sahiptir. İşlemcinin verilen göreve dışarıdan programlanabilmesi için bir JTAG Debug modül aracılığı ile bir bilgisayarla haberleşebilir. Ayrıca işlemcinin komut ve data hafızası olarak kullanılması için dışarıda sıkı bağlantılı (Tightly Coupled) hafızalara da bağlantısı vardır. Ayrıca işlemcinin dışarıyla bağlantılı reset, clok girişi v.b pinleri de mevcuttur.

NIOS 3.0 işlemci, Tablo 4.6 da görüldüğü gibi modül olarak 3 farklı tipte gerçeklenebilir. Kullanıcı tasarımını yaparken bunlardan herhangi birisini seçebilir.

Tablo 4.6 NIOS işlemci versiyonlarının karşılaştırılması

NIOS II	Nios II/f (Fast)	Nios II/s (Standart)	Nios II/e (Economy)
Pipeline	6-Stage	5-Stage	None
Max. Frequency1	200MHz	180Mhz	210Mhz
Max. D-MIPS1	255	130	30
Size(4-Input LUTs)	1800	1200	600
Branch Prediction	Dinamik	Statik	No
I-Cache	Up to 64K	Up to 64K	No
D-Cache	Up to 64K	No	No

4.4 Avalon Bus

Avalon bus (Veriyolu) Nios gömülü işlemciler tarafından kullanılan Altera'ya özgü Bus arabirimidir. Wizard tabanlı avalon SOPC builder geliştirme tool (aracı)'u otomatik olarak Avalon bus yapısını üretir. Üretilen anahtar çatı mantığı chip seçme sinyallerini (data path multiplexing, adres decoding, wait-state generation , interrupt priority assignment, dynamic bus sizing, bus control signal) içerir. Tablo 4.7 Avalon Bus mimarisinin temel özelliklerini göstermektedir.

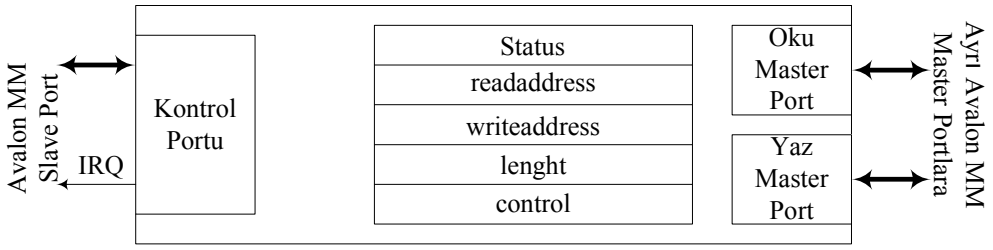
Tablo.4.7 Avalon Bus mimarisinin özellikleri

	AVALON BUS
Data Bus Genişliği	8,16,32 Bit
Adres Bus Genişliği	32 Bit
Mimari	Çoklu-Master/Çoklu-Slave
Özellikler	Kesme öncelik ataması, Wait-state üretimi, Oku/yaz transferi
Data Bus Protokol	Bir veya daha fazla bus çevrimi, gruplandırma transferi, bayt , half word veya word transferi (8,16,32 bit)
Zamanlama	Bütün sinyaller Avalon saat frekansı ile senkronizelidir.
Desteklenen bağlantılar	Ayrı adres, data ve kontrol hatları, bridge ile tri-state sinyalleri
Teknoloji	Altera Avalon SOPC builder kullanan altera cihazlarında gerçekleştirilebilir.

Avalon Bus, On chip işlemcileri ve çevre birimlerinin iletişimi için tasarımlanmış bus mimarisidir. Master(Usta) ve Slave(Köle) komponentler arasındaki port bağlantılarını ve komponent iletişimi için zamanlamayı belirler. FPGA'da komponentlerin kolayca birbirine bağlanabilmesi için geliştirilmiş basit bir arabirimdir. Avalon ailesi hem yüksek hızlı akış hem de hafıza haritalama uygulama kullanımları için geliştirilmiştir.

4.5. Doğrudan Hafızaya Erişim (DMA) Modülü

DMA modülü, Altera SOPC Builder kütüphane komponentidir. DMA, CPU meşgul edilmeden hafızadan hafızaya ya da çevre birimleri arasında veri transferine izin verir. DMA'nın Şekil 4.16' da gösterildiği gibi iki adet Avalon master portu vardır. Bunlardan ilki oku (Read Master) diğeri yaz (Write Master) portudur. Ayrıca bir adette kontrol işlemleri için Slave (Avalon Slave Port) porta sahiptir.



Şekil 4.16 DMA modülü Master ve Slave portları

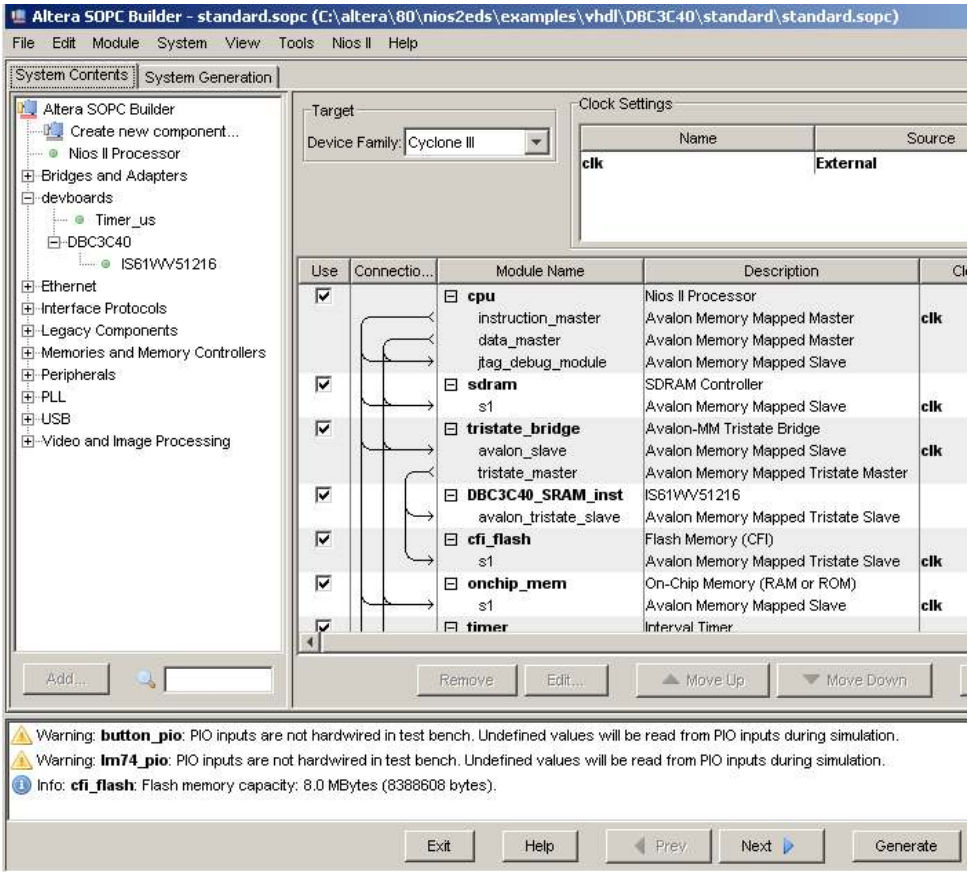
DMA' nın çevre birimleri ile veri transferi yapabilmesi için gerekli transfer süreci aşağıdaki gibidir.

1. CPU tarafından, veri transferi gerçekleştirmek için kontrol portu vasıtasıyla DMA konfigüre edilir.
2. DMA ilgili çevre birimlerine erişir. CPU nun herhangi bir etkisi olmaksızın veri transferi başlatılır.
3. Oku portu okuma adresinden itibaren hafıza veya çevrebirimlerinden veri okur, yazma portu yazma adresinden itibaren hafıza veya çevre birimlerine verinin yazılmasını sağlar. Okuma ve yazma işlemleri esnasında veri, her iki portun direkt olarak erişebileceği FIFO kuralına göre çalışan hafıza birimlerinde saklanır.
4. Transferin sonunda "end of packet" EOP üretilir ve DMA transferin tamamlandığını göstermek üzere kesme üretir.
5. Veri transferi boyunca veya sonrasında DMA durumu Status kayıtçısı vasıtasıyla belirlenir.

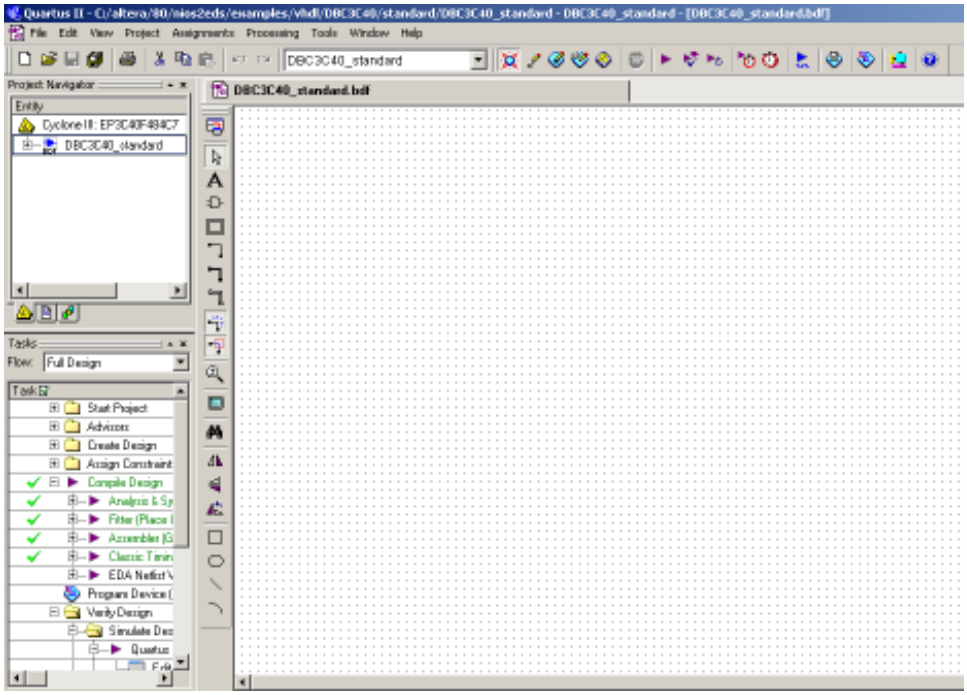
4.6.Yazılım Araçları

Bu projenin uygulama ve geliştirme süreçlerinde Altera firmasının Quartus 8.0 ve Nios IDE 8.0 yazılım araçları kullanılmıştır. Quartus 8.0; kullanıcılara SOC ve SOPC tasarımı gerçekleştirmesine olanak sağlayan Altera firması tarafından ücretsiz olarak kullanıma sunulan bir araçtır. Kullanıcılara hem VHDL hem de Verilog donanım tanımlama dili ile sistem geliştirme imkanı sunmasının yanında, geliştirilecek sistemin şematik olarak oluşturulmasına da izin verir. Quartus 8.0 yazılımı, programlanabilir gömülü sistem geliştirme için SOPC Builder yazılım aracına da sahiptir. Bu araçlar bir sistemin FPGA içerisine gömülmesi süreci için gerekli olan yazılımlardır.

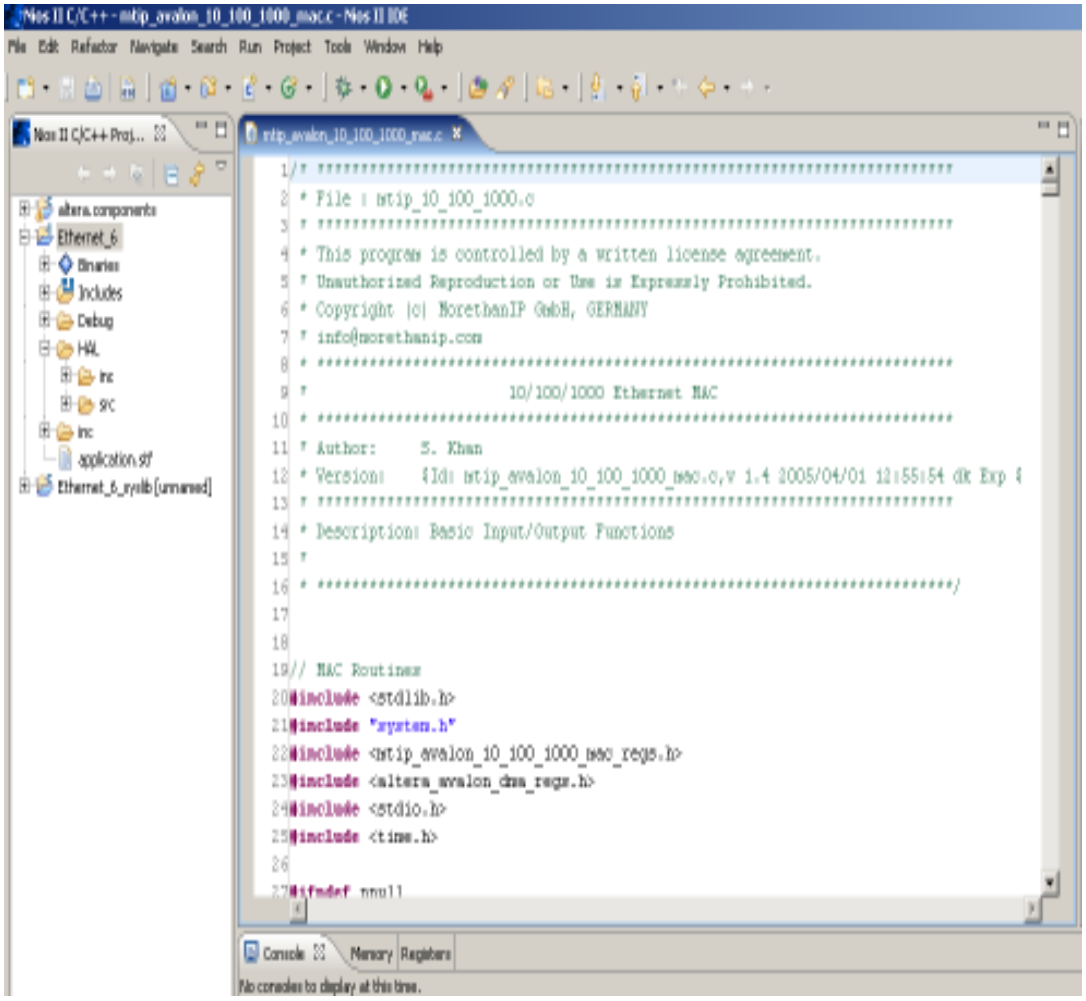
NIOS IDE 8.0 yazılımı ise Altera tarafından ücretsiz olarak kullanıma sunulan kullanıcıların tasarladığı SOPC sistemlerinin bir göreve programlanması için kullanılan C/C++ tabanlı bir araçtır. C/C++ ile yazılmış bir program parçası, bu araçla assemler kodlarına çevrilerek işlemcinin program hafızasına yüklenmesini sağlar. SOPC sistemdeki her bir donanım için özel sürücüleri de içerir. Şekil 4.17 SOPC Builder arayüzünü, Şekil 4.18 Quartus 8.0 arayüzünü ve Şekil 4.19 Nios II IDE arayüzünü göstermektedir.



Şekil 4.17 SOPC Builder arayüzü



Şekil 4.18 Quartus 8.0 arayüzü

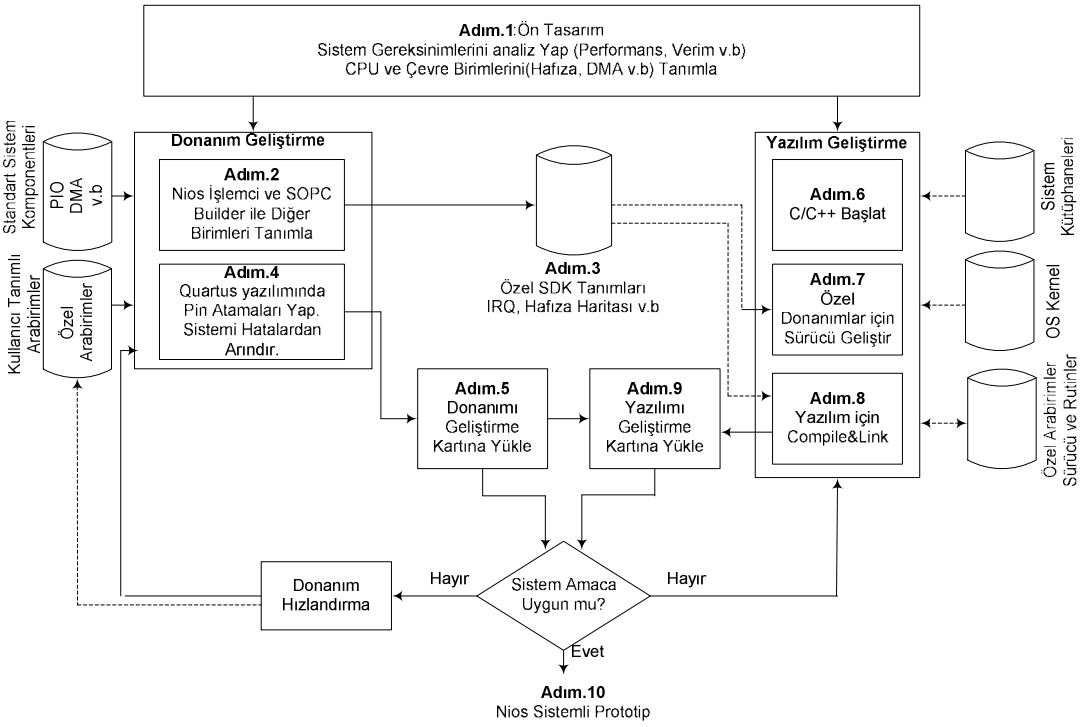


Şekil 4.19 Nios II IDE arayüzü

4.7 Programlanabilir Gömülü Sistem Tasarım Aşamaları

FPGA içinde oluşturulacak lojik devrelerin, soft işlemcilerin ve çevre birimlerinin tasarlanacak sistemi oluşturacak şekilde birleştirilmesi, SOC veya SOPC olarak tanımlanır.

Bir SOPC sisteminin FPGA içerisine gömülmesi ve çalışabilir duruma gelmesi için gerekli donanım ve yazılım adımları Şekil 4.20' de akış şeması şeklinde verilmiştir. Akış şemasındaki donanım ve yazılım geliştirme süreçleri aşağıdaki bölümlerde açıklanmıştır.



Şekil 4.20 Gömülü sistem geliştirme süreci

4.7.1 Donanım Geliştirme Süreci

Adım 1: Tasarımı yapılacak SOPC 'un performans, verim v.b kriterlerini sağlayacak donanımsal üniteler kağıt üzerinde belirlenir. Bu donanımları tarif eden yazılımlar SOPC Builder yazılım aracının kütüphanesinde hazır olarak bulunabileceği gibi, kullanıcı tarafından da donanım tanımlama dili kullanılarak oluşturulabilir. Önceden tanımlı bu üniteler (karmaşık fonksiyonlar) genelde "IP Cores " olarak adlandırılır.

Adım 2: Bu adımda tasarlanacak sistemin tüm donanım birimleri (Komponentler-modüller), SOPC Builder yazılım aracı ile seçilir ve ilgili bağlantılar yapılır.

Adım 3: Sistem geliştirme kiti (System Development Kit, SDK) vasıtasıyla sistemin kullanacağı kesmeler, hafıza haritası v.b belirlenir. Elde edilen bu sistem derlenir ve hatalardan arındırılarak SOPC olarak elde edilir.

Otomatik olarak üretilmiş yazılım geliştirme kiti, C programlama dili için özel komutlara erişim ve assembler dili programlamalarını içerir. Bu adımda kullanılan kesme ve adresler, sistem için geliştirilecek yazılım ile ilişkili olduğundan yazılım geliştirme sürecinin 3. adımı ile ilişkilidir.

Adım 4: SOPC Builder ile oluşturulmuş gömülü modül, Quartus 8.0 ortamına aktarılarak, varsa kullanıcı tanımlı çevre birimleriyle bağlantısı gerçekleştirilir. Son olarak gömülü sistemin Pin bağlantıları yapılır. Sistem derlenerek hatalar giderilip FPGA'ya gömülecek duruma getirilmiş olunur.

Adım 5: 4. Adımda elde edilen sistem karta yüklenerek donanımsal olarak gömülü sistem elde edilmiş olur.

4.7.2 Donanımı Kontrol Eden Yazılım Geliştirme Süreci

Şekil 4.20'deki akış şemasını devam ettirerek, programlanabilir donanımı kontrol eden yazılımların geliştirme sürecini aşağıdaki gibi açıklayabiliriz.

Adım 6: Gömülü sistemin istenen görevi yerine getirmesi için C/C++ programlama ortamında programlanması yapılmalıdır. Bu adımda donanımın çalıştıracağı yazılım için gerekli sistem kütüphaneleri tanımlanır.

Adım 7: Nios IDE 8.0 tarafından tanımlanabilen donanımlar (CPU, DMA v.b gibi) için gerekli sürücü tanımlamaları otomatik olarak HAL (Hardware Abstraction Layer) tarafından gerçekleştirilir. Sistem geliştirildikten sonra herhangi bir donanımdaki değişiklik, HAL donanım sürücü konfigürasyonlarına yansıtılır. HAL ile C Standart kütüphanesi bütünleşik olduğundan dolayı, C programı ile HAL tarafından tanımlı donanımlara erişim desteklenir. Bu adımda HAL tarafından desteklenmeyen donanımlar için sürücüler geliştirilir.

Adım 8: Bu adımda istenen yazılımın derlenmesi işlemi gerçekleştirilir.

Adım 9: Karta yüklenmiş donanım için geliştirilen yazılım, JTAG ara birimi kullanılarak, bilgisayardan, kartta oluşturulmuş soft işlemcinin program hafızasına yüklenir.

Adım 10: Donanım ve yazılımın birlikte çalıştırılması ile gömülü sistemin testleri yapılır. Testler ile gömülü sistemden beklenen performans kontrol edilir. Performansın iyi olmaması durumunda tamamlanmış bu gömülü sistemin hızlandırılması için ek bileşenler eklenerek donanım geliştirme sürecinin 4. adımına geri dönülerek 4,5,7,8,9 ve 10. adım tekrarlanır.

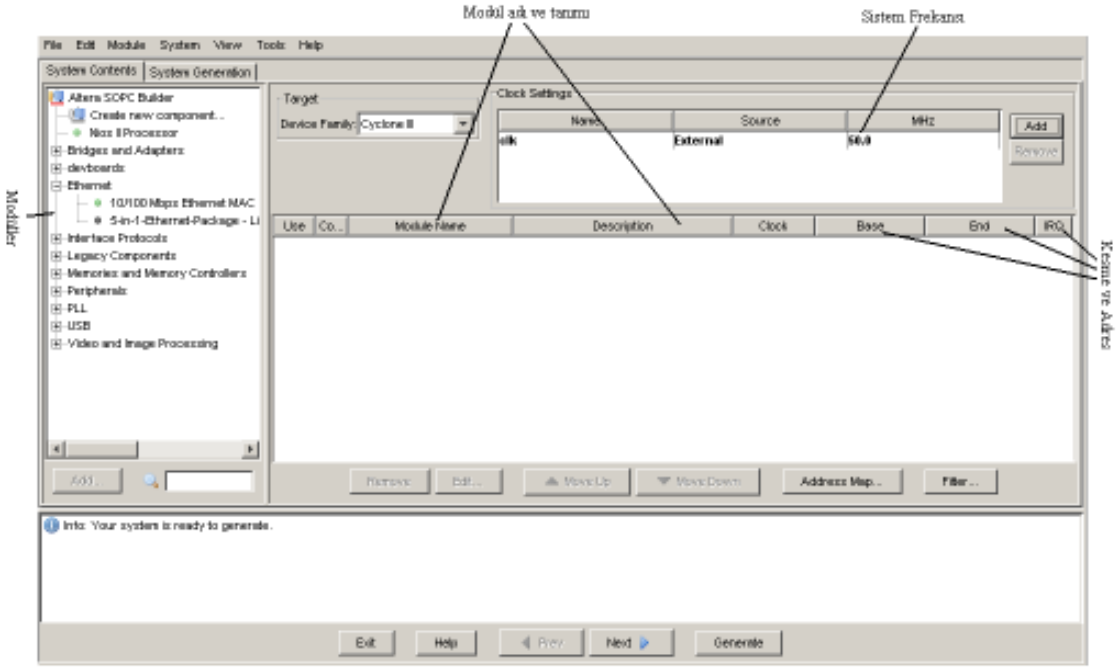
Bu bölümde açıklanan "programlanabilir gömülü bir sistemin geliştirme aşamaları" Tezin 5,6 ve 7. bölümünde gerçekleştirilecek gerçek zamanlı bir STS'nin tasarlanması sürecinde yapılanların ifade edilebilmesi için alt yapı oluşturmaktadır.

5. STS DONANIMININ FPGA ORTAMINDA GERÇEKLEŞTİRİLMESİ

Geliştirilmiş olan STS, FPGA içerisinde donanımın oluşturulması ve bunun içindeki Nios işlemcisinin programlanması aşamalarından oluşur. Bu bölümde programlanabilir gömülü STS'nin donanımsal gerçekleştirilmesi üzerinde durulacaktır.

STS'nin donanım kısmının geliştirilmesi aşamaları, 4.bölümde açıklanan donanım geliştirme süreci baz alınarak açıklanabilir. Tasarlanacak sistemin performansı için öncelikle sistemde olması gereken donanım birimleri seçilmiştir. Bu donanım birimleri NIOS İşlemci, Hafıza, DMA, JTAG, Timer (zamanlayıcı), SysID ve 10/100 Ethernet IP MAC Core' dur. Paket alma/gönderme işlemi için önceki bölümlerde bahsedilen donanımların gerekliliği açıklanmıştı. Alınan/gönderilen paketlerin işlenebilmesi için, yani paketlerin saldırı olup olmadığının tespitinin yapılabilmesi için Nios işlemcinin programlanmasının da ayrıca yapılması gerekir.

Sistem tasarımına Quartus 8.0 yazılımından, gömülü sistemin geliştirileceği FPGA donanımı (Cyclone EP3C40F484C7N) seçilerek başlanır. Seçilen bu donanım için gömülü sistem, SOPC Builder tarafından oluşturulabilir. Şekil 5.1 SOPC Builder'in genel görünümünü göstermektedir.

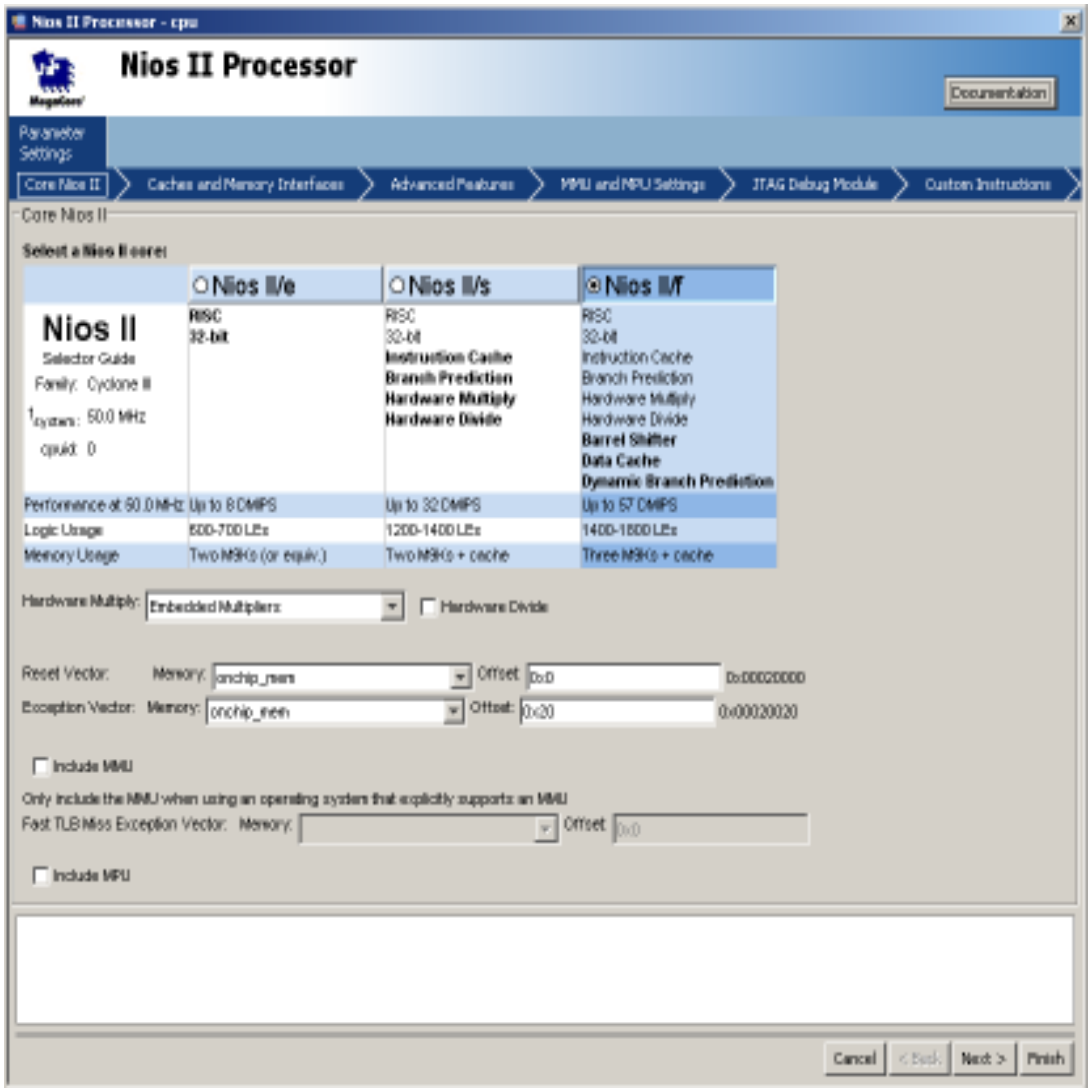


Şekil 5.1 SOPC Builder ortam tanıtımı

Burada seçilen her bir donanım için kesmeler, adres, bağlantı yolları, sistemin kullanacağı frekans ve modüller belirlenir. Sistem için bu arayüzden seçilen modüller açıklanmıştır.

5.1 İşlemci Seçim Özellikleri

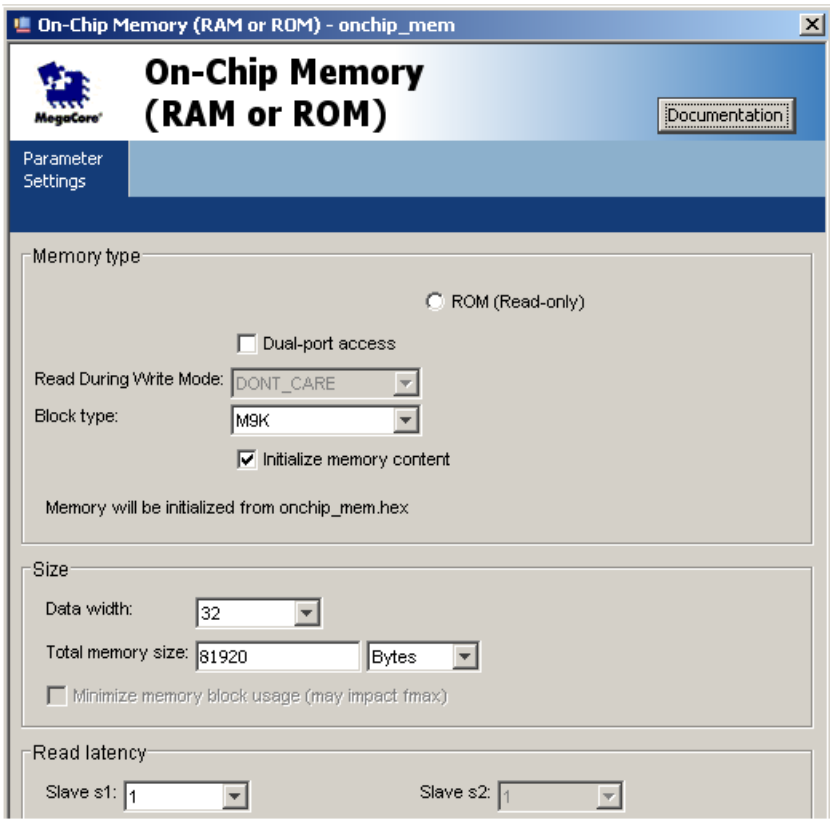
İlk olarak gömülü sistemin görevini yerine getirecek olan işlemci (Nios II Processor) sisteme eklenir. Burada önemli olan işlemcinin türünün, *Reset* ve *exception* vektörlerinin belirlenmesidir. Şekil 5.2 de görüldüğü gibi Nios II/f (hızlı) işlemci, Cyclone III donanımı için seçilmiştir. İşlemci clock frekansı 50 khz' dir.



Şekil 5.2 Nios II işlemci

5.2 Hafıza Seçim Özellikleri

Sisteme işlemci eklendikten sonra eklenmesi gereken bir diğer modül hafıza'dır. Hafızaya hem paketlerin yazılması hem de okunması gerektiğinden hafıza birimi olarak RAM modülü seçilmiştir. Şekil 5.3 eklenen hafıza modülüne ait özellikleri göstermektedir.

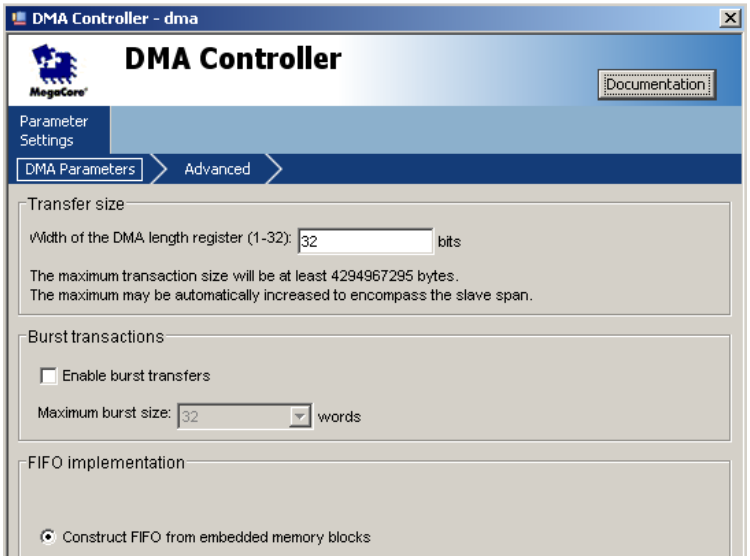


Şekil 5.3 Hafıza modül özellikleri

Bu aşamada kullanılacak hafızanın blok tipi, veri genişliği ve boyutu belirlenmelidir. Şekil 5.3' te görüleceği üzere blok tipi MK9, veri genişliği 32 bit ve boyutu 80 KBayt seçilmiştir.

5.3 DMA Seçim Özellikleri

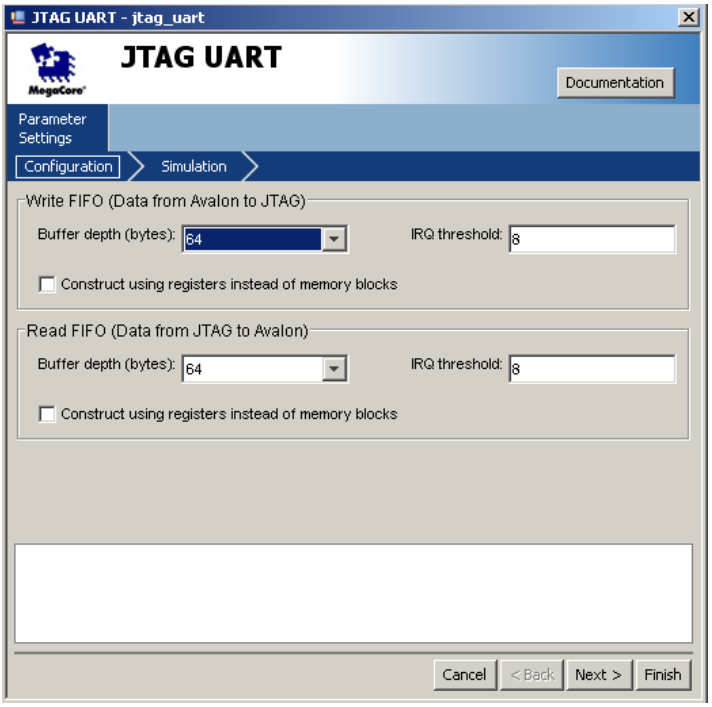
DMA seçiminde önemli olan parametreler, kullanılacak kayıtçı uzunluğunun belirlenmesi ve DMA'nın hafızadan hafızaya veri transferi yapılırken kullanacağı FIFO hafızalarının nerede oluşturulacağıdır. Şekil 5.4' te görüleceği gibi kayıtçı uzunluğu 32 bit FIFO hafızaları ise yukarıda belirlenen 80 KBayt uzunluktaki hafıza uzayında oluşturulmuştur.



Şekil 5.4 DMA özellikleri

5.4 JTAG Seçim Özellikleri

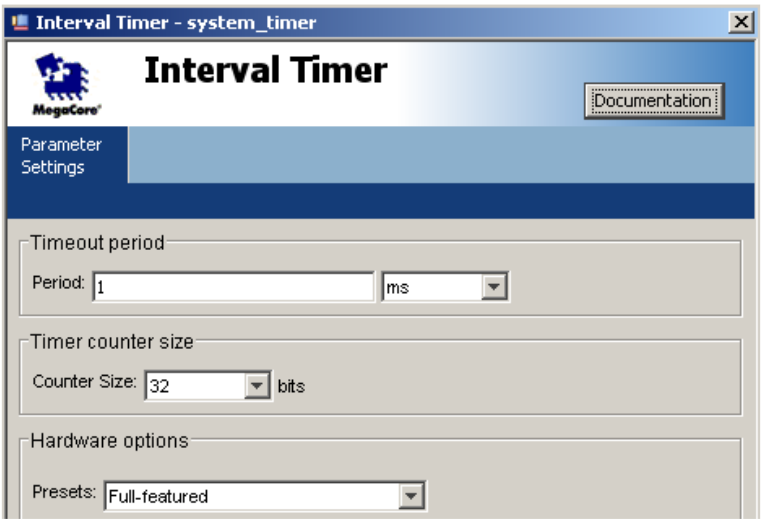
Bu donanım için kesme ve Avalon bus ile FIFO'lara yazılacak ve okunacak veri derinliği belirlenir. JTAG için Şekil 5.5' te gösterildiği gibi kesme 8, FIFO' lar için veri derinliği 64 Bayt olarak seçilmiştir.



Şekil 5.5 JTAG özellikleri

5.5 Timer (Zamanlayıcı) Seçim Özellikleri

Nios işlemci için ön ayarlamalar yapmak için kullanılır. LE' ler arasında konfigürasyon yapılabilmesi için Şekil 5.6'da temel ayarlar olarak, full-featured (tam özellik) seçilmiştir.



Şekil 5.6. Timer Özellikleri

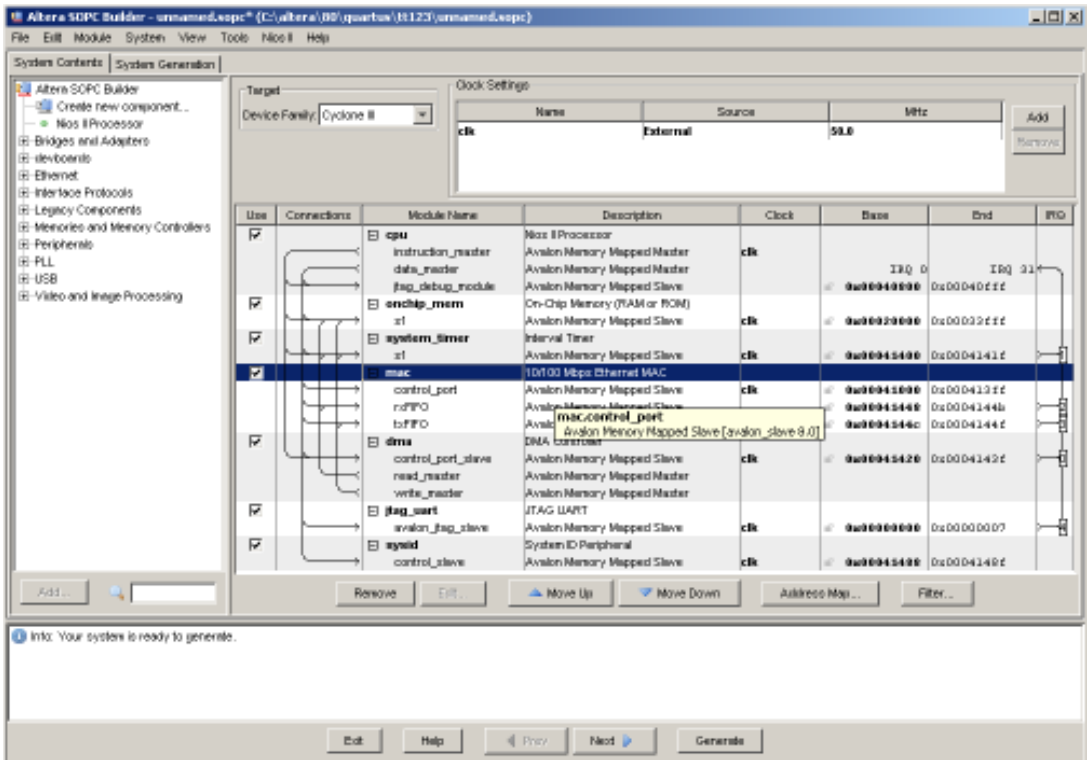
5.6 SysID Seçim Özellikleri

Gömülü sistem üretildiği zaman, sistem tarafından bir tek zaman damgası ve sistem ID eklemek için kullanılır. Seçimlik herhangi bir parametresi yoktur.

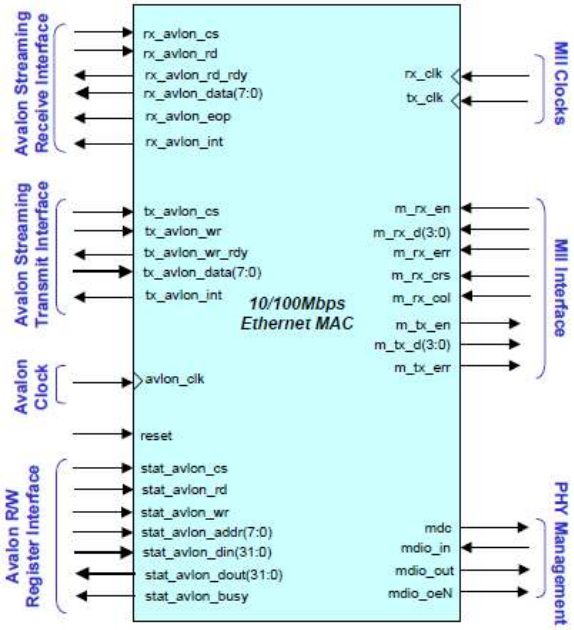
5.7 10/100 Ethernet IP MAC Core Seçim Özellikleri

Lisanslı olarak IP core şeklinde satın alınmış bu modül SOPC Builder'ın modüller menüsüne eklenmiştir. Eklenen bu modülün sistem için kullanılmasında herhangi bir seçimlik bilgi yoktur.

Bu 7 donanım SOPC builder'a eklenmesiyle otomatik olarak kesmeler, başlangıç ve bitiş adresleri, modüller arasındaki bağlantılar oluşturulur. Bu bağlantıları tasarlayıcı isterse değiştirebilir. Şekil 5.7'de modüllerin SOPC sisteme eklenmiş hali görülmektedir [57].



Şekil 5.7 SOPC tasarımı ve modüller arası bağlantı



Şekil 5.8 10/100 Ethernet MAC modülünün blok gösterimi

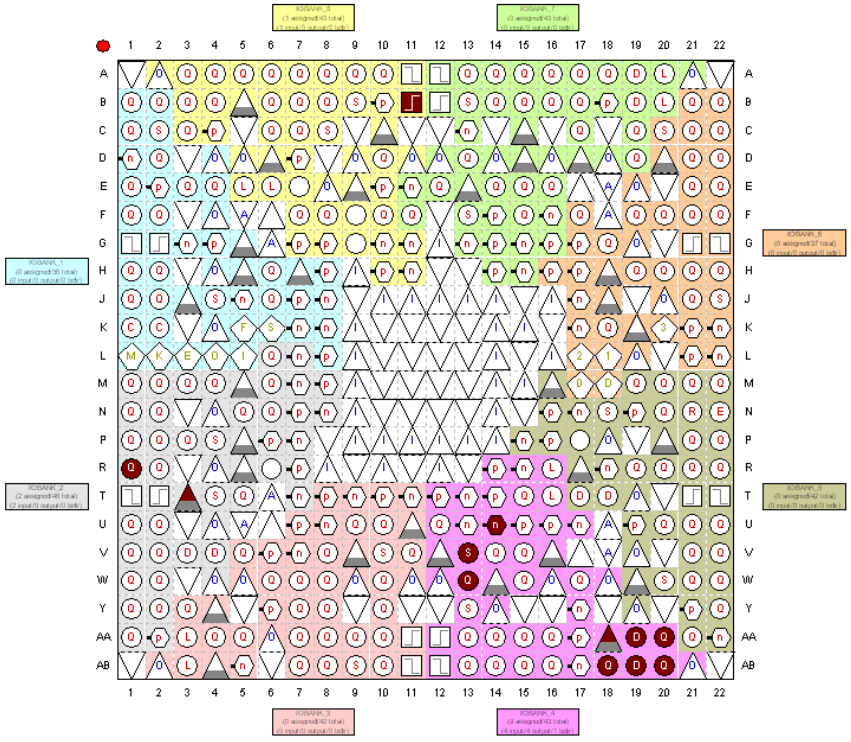
Blok gösterimi Şekil 5.8'te verilen gömülü sisteme eklenmiş 10/100 Ethernet MAC Modülü MII arabirimini desteklemektedir. Ancak bu sistemin üzerinde kurulduğu kart

(DBC3C40) RMII (azaltılmış MII) arabirimine sahip olduğundan dolayı sisteme bir MII/RMII dönüştürücü modülü de eklenmiştir.

Bu aşamalardan sonra seçilen modülleri içeren bir SOPC oluşturulması ve Quartus ortamına aktarılması gereklidir. Quartus ortamına eklenen SOPC yukarıda bahsedilen donanımları içermektedir. Bu SOPC ve varsa yardımcı diğer modüller ve elektronik elemanlar (sistemin dışarıdan başlatılabilmesi veya resetlenmesi için gerekli donanımlar) Quartus ortamında eklenir. Her biri arasındaki bağlantılar Şekil 5.10' da gösterildiği gibi gerçekleştirilir. Modüllerin kullanılmayan pinleri toprağa bağlanır. Fiziki ortama bağlanacak pinler birer uçları DP83640 entegresine bağlanır.

Sistemin başlatılması için özel olarak bir reset sistemi hazırlanmıştır. Nios işlemci ve 10/100 Ethernet MAC core'un sistemin her başlatılması durumunda resetlenmesi gereklidir. Nios işlemcinin resetlenmesi için aktif düşük (Lojik 0), Ethernet modülünün ise aktif yüksek (Lojik 1) işarete sahip olması gereklidir. Bu nedenle D türü filip flop ile bir resetleme devresi VHDL ortamında oluşturularak sisteme eklenmiştir.

Donanımsal sistemde dikkat edilmesi gereken bir başka konu pinlere, doğru giriş/çıkış veya hem giriş hem çıkış atama işlemlerinin yapılmasıdır. Örneğin Ethernet MAC Core'un Rx uçları paket alınacağı için giriş, Tx uçları paket gönderileceği için çıkış, MDIO adreslerinin ise hem yazılacağı hem de okunacağından dolayı giriş çıkış seçilmiştir.



Şekil 5.9 Quartus ortamında pin atamaları

Şekil 5.9 FPGA'da gerçekleştirilen STS donanımının dış dünya ile olan fiziksel pin atamalarını göstermektedir. STS uygulamasına özel olarak atanmış pin tanımlamaları ise Tablo 5.1'de verilmiştir.

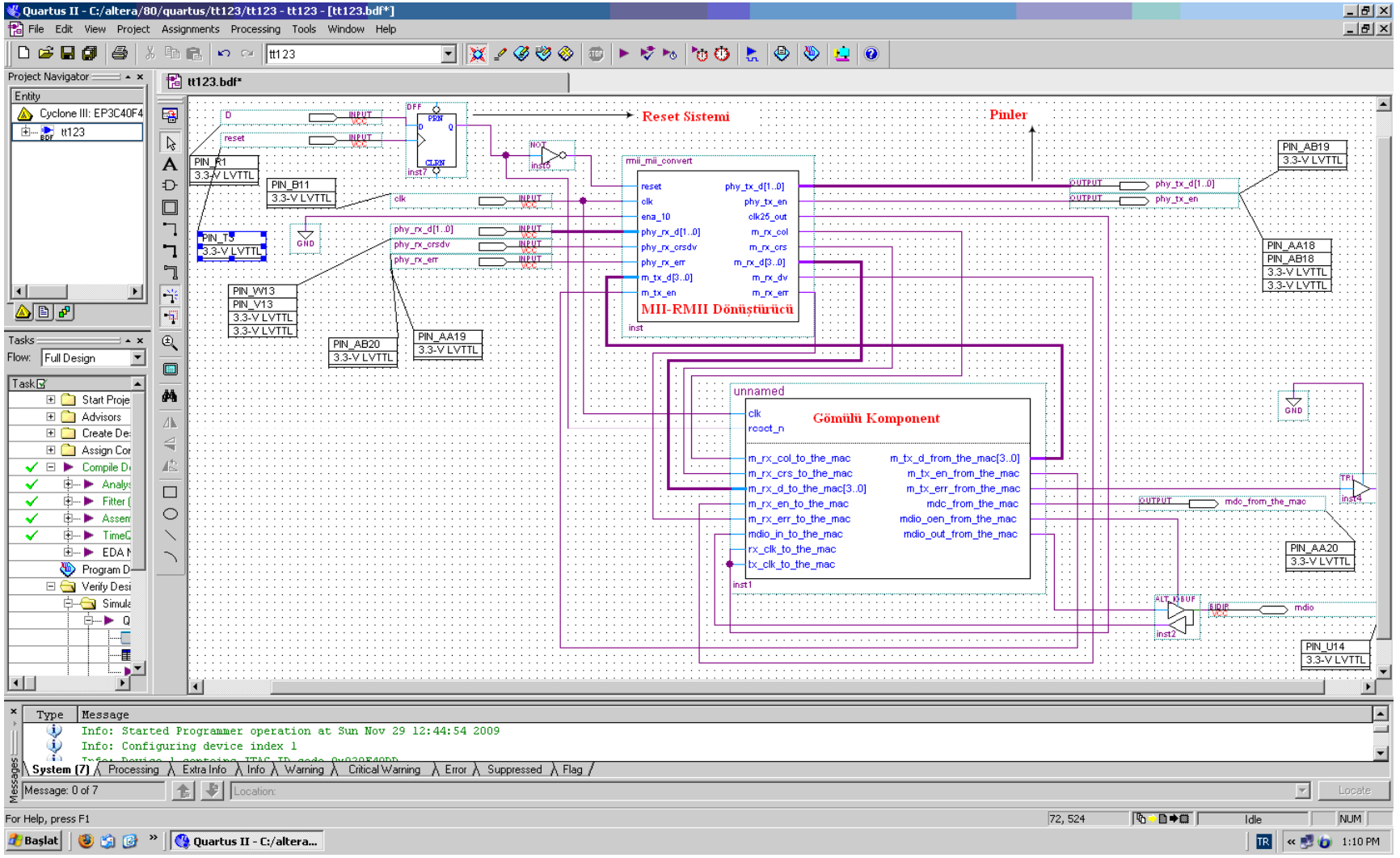
Tablo 5.1 STS donanımının pinleri

Pin No	Pin Adı	Tipi
B11	clk	Giriş
R1	D (Flip-Flop)	Giriş
T3	reset	Giriş
U14	MDIO	G/Ç
V13	phy_rx_d[1]	Giriş
W13	phy_rx_d[0]	Giriş
AA18	phy_tx_d[1]	Çıkış
AB18	phy_tx_d[0]	Çıkış
AA19	phy_rx_err	Giriş
AB19	phy_rx_en	Çıkış
AA20	mdc_from_the	Çıkış
AB20	phy_rx_crsdv	Giriş

Tüm bağlantılar ve pin atamaları tamamlandıktan sonra sistem derlenerek varsa hatalar giderilir. Herhangi bir hatanın olmaması durumunda sistem, FPGA ortamına yüklenecek duruma gelmiş olur. Şekil 5.10 FPGA'ya gömülmeye hazır STS donanımının şemasını göstermektedir.

Bundan sonra sistem Quartus ortamından JTAG vasıtasıyla FPGA içerisine yüklenerek, gömülü donanımsal sistem elde edilmiş olunur.

Şekil 5.11'deki rapordan da görüldüğü gibi, STS donanımının FPGA içerisinde oluşturulması için 7.875 adet LE, 4893 adet kayıtçı, 793.204 bit hafıza alanı, 4 adet çarpıcı devre kullanılmıştır.



Şekil 5. 10 Gömülü STS'yi gerçekleştirecek blok yapısı.

Flow Status	Successful - Sat Jan 30 14:04:32 2010
Quartus II Version	8.0 Build 215 05/29/2008 SJ Web Edition
Revision Name	tt123
Top-level Entity Name	tt123
Family	Cyclone III
Device	EP3C40F484C7
Timing Models	Final
Met timing requirements	N/A
Total logic elements	7,875 / 39,600 (20 %)
Total combinational functions	6,711 / 39,600 (17 %)
Dedicated logic registers	4,893 / 39,600 (12 %)
Total registers	4893
Total pins	17 / 332 (5 %)
Total virtual pins	0
Total memory bits	793,204 / 1,161,216 (68 %)
Embedded Multiplier 9-bit elements	4 / 252 (2 %)
Total PLLs	0 / 4 (0 %)

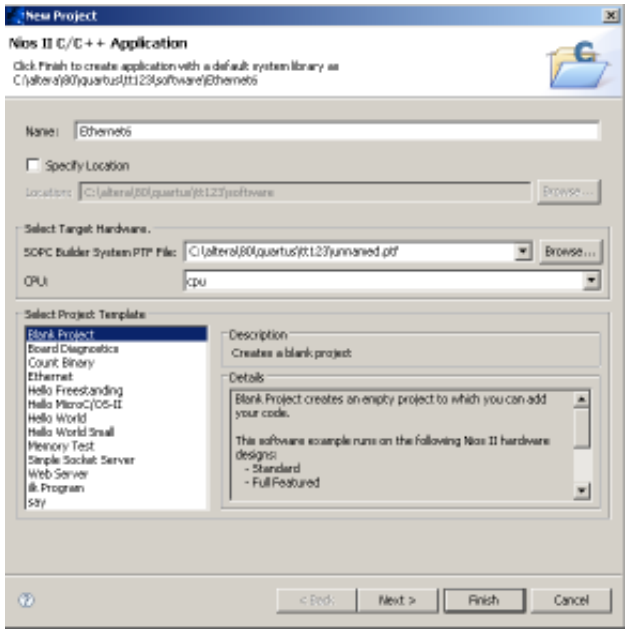
Şekil 5.11. Quartus'ta STS'nin derlenmesi ve FPGA'ya gömülmesi sonuç raporu

Bu şekilde oluşturulan donanımsal STS *(Unnamed SOPC)* artık programlanabilecek duruma gelmiştir. MAC modülünün konfigürayonu ve STS'nin değişik sınıflandırma algoritmalarına göre gerçek zamanlı çalıştırılabilmesi için gerekli olan programlamalar bir sonraki bölümde açıklanacaktır.

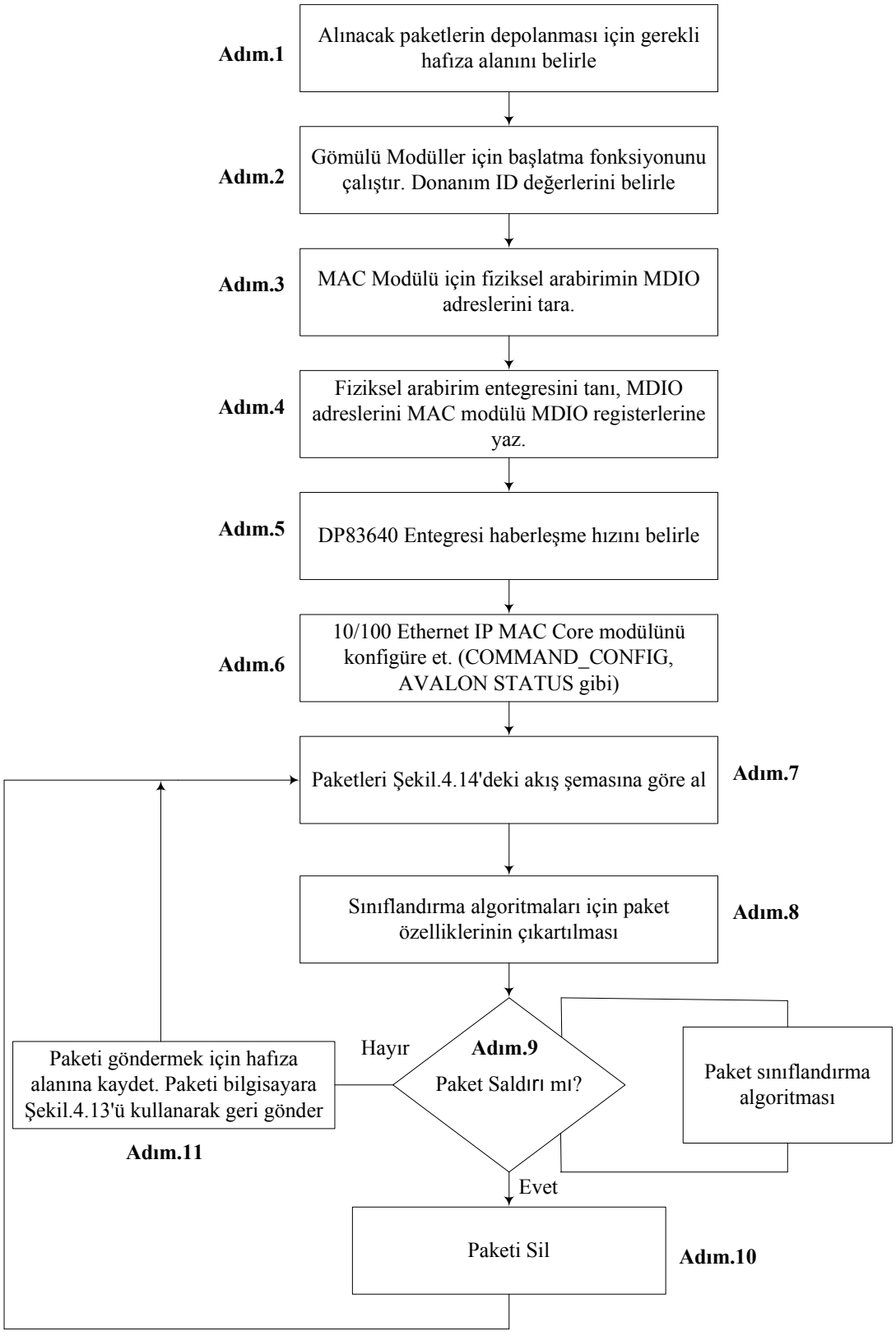
6. GÖMÜLÜ STS' NİN PROGRAMLANMASI

Programlanabilir gömülü sistem oluşturulduktan sonraki adım, bu gömülü sistemin işlevini belirlemektir. Bunun için Nios II IDE 8.0 ortamında C/C++ ile bir uygulama projesinin oluşturulması gereklidir. Şekil 6.1 oluşturulan projeyi göstermektedir. Oluşturulan proje ismi ***Ethernet6*** olarak seçilmiştir. Yazılacak programın hangi gömülü sistem için kullanılacağını belirlemek için donanım seçilmelidir. Biz SOPC sistemde oluşturduğumuz gömülü sistem için *unnamed.ptf* donanımını seçtik. Bu seçimden sonra bu donanımda kullanılan işlemci otomatik olarak Şekil 6.1 de görüldüğü gibi CPU alanına eklenir. Eğer önceden yazılmış ve geliştirilmiş bir yazılım varsa *Select Project Template* bölümünden seçilerek proje tamamlanır. Eğer yeniden bir proje hazırlanacak ise boş bir proje seçilerek yeni bir program yazılır. Böylece Nios işlemcinin programlanması için gerekli adımlar tamamlanmış olunur.

Bu tezde tasarlanan STS donanımının konfigürasyonu ve paket alma/gönderme işlemi için C/C++ ortamında yazılmış olan program, ***Ethernet*** projesi içerisine kaydedilmiştir. Bu programın akış şeması Şekil 6.2' de gösterilmektedir. Şekil 6.2 ayrıca;, ağdan yakalanacak paketlerin sınıflandırılması işlemi için gerekli olan yazılımın akış şemasını da içermektedir.



Şekil 6.1 Yeni bir proje oluşturma



Şekil 6.2 STS gömülü sisteminin programlanması adımları

Adım 1: STS bir sınıflandırma işlemi gerçekleştireceğinden dolayı paketler öncelikle, adresleri tanımlanmış bir hafıza alanına alınmalıdır. Bu adreslenmiş hafıza alanı işlenecek veya gönderilecek paketlerin tutulduğu yerdir. Bu tezde 8 KByte'lık bir hafıza alanı bu işlem için belirlenmiştir.

Adım 2: Sisteme yüklenecek yazılım, t=0 anında çalışmaya başlayacağından dolayı işlemci ve diğer birimlerin başlatılması gereklidir. Bu amaç için bir başlatma fonksiyonu yazılmıştır. Bu fonksiyon SOPC tarafından her bir donanım birimine verilen başlangıç adres alanını ID numarası olarak belirler. Bu ID numaraları kullanılarak donanım birimleri arasında veri aktarımı yapılır. Örneğin ağdan alınmış bir paket 10/100 Ethernet MAC modülünün FIFO hafızasında saklanır. FIFO' dan DMA yoluyla paket program hafıza alanına alınır. DMA sistem fonksiyonlarını IORD (adres, kayıtçı) veya IOWR (adres, uzunluk, kayıtçı) kullanır.

Adım 3: MAC modülünün fiziksel arabirim ile iletişim kurabilmesi için aktif fiziksel arabirim aranır.

Adım 4: Fiziksel katman görevlerini yerine getiren DP83640 entegresi tanınır. İlgili MDIO adresleri 10/100 Ethernet MAC Core' un MDIO_0 ve MDIO_1 kayıtçılarına yazılır.

Adım 5: DP83640 entegresinin hızı belirlenir.

Adım 6: 10/100 Ethernet MAC Core' un paket alma ve gönderme yapabilmesi için ilgili kayıtçılar konfigüre edilir.(COMMAND_CONFIG v.b.)

Adım 7: Bilgisayardan (veya ağdan) gönderilen paketler Şekil 4.14' deki akış şemasına göre alınır. (10/100 Ethernet MAC Core' un alma FIFO'sundan program hafızasına)

Adım 8: Paketin ilgili alanları belirlenen değişkenlere atanır. Bu değişken 8 elemanlı bir dizi olabileceği gibi farklı değişkenlerde olabilir.

Adım 9: Alınmış paketin 8 özelliği kullanılarak, Karar ağacı, Naive Bayes ve Yapay sinir ağları yardımıyla gerçekleştirilen sınıflandırma algoritmasına göre paketin sınıfı belirlenir.

Adım 10: Eğer paketin sınıfı saldırı olarak belirlenmiş ise paket hafızadan silinerek 7. Adım'a dönülerek yeni paket beklenir veya alınır.

Adım 11: Eğer paket normal sınıfına ait ise program hafızasındaki paket Şekil 4.13' deki akış şemasına göre bilgisayara (veya ağa) tekrar geri gönderilerek, 7. adıma dönülür ve yeni paket beklenir varsa alınır.

Bu akış şemasındaki 8., 9. ve 10. adımlar 7. bölümde daha detaylı olarak açıklanacaktır. Akış şemasına göre yazılmış program ise Ek 2,3,4' de verilmiştir.

6.1 STS Donanımının Konfigürasyonu ve Paket Alma Gönderme Fonksiyonları

Paketlerin alınması ve gönderilmesi işlemlerini gerçekleştirmek için öncelikle MAC modülünün konfigüre edilmesi gereklidir. Bu işlemler Şekil 6.2'deki akış şemasının ilk altı adımında gerçekleştirilir. MAC modülünün konfigüre edilmesi için gereken ön hazırlıklar aşağıda maddeler halinde anlatılmıştır.

1. Fiziksel arabirimin MDIO adresleri taranır. Aktif olan entegre seçilir.(RJ45 portuna bağlı DP83640 Entegre MDIO adresleri)
2. Seçilen entegrenin tipi belirlenir (National Firmasının DP83640 entegresi gibi).
3. Entegrenin haberleşme hızı belirlenir (10 Mbps veya 100 Mbps gibi).

Aşağıda 10/100 Ethernet MAC Core'un konfigürasyonu için yazılan C programının bir bölümü görülmektedir.

```
// MAC Kayıtçılarının içeriklerinin tanımlanması
mi.mac->IRQ_CONFIG = 0;
mi.mac->MAC_0 = ((int)(myAddr[5]) & 0xff) | (((int)(myAddr[4]) & 0xff) << 8) |
        (((int)(myAddr[3]) & 0xff)<< 16) |  (((int)(myAddr[2]) & 0xff)<< 24);
mi.mac->MAC_1 = ((int)(myAddr[1])  & 0xff) | (((int)(myAddr[0]) & 0xff) << 8);
mi.mac->FRM_LENGTH = 15100;
mi.mac->RX_ALMOST_EMPTY = 8;
mi.mac->RX_ALMOST_FULL = 14;
mi.mac->TX_ALMOST_EMPTY = 8;
mi.mac->TX_ALMOST_FULL = 20;
mi.mac->TX_SECTION_EMPTY = 0;
mi.mac->TX_SECTION_FULL  = 0;
mi.mac->RX_SECTION_EMPTY = 0;
mi.mac->RX_SECTION_FULL = 0;
mi.mac->TX_CMD_STAT = 0x00000000;
mi.mac->RX_CMD_STAT = 0x00000000;
mi.mac->AVL_STATUS=0x00000000;
mi.mac->COMMAND_CONFIG  |=  mmac_cc_TX_ENA_mask      |
mmac_cc_RX_ENA_mask | mmac_cc_PROMIS_EN_mask | mmac_cc_PAD_EN_mask|
mmac_cc_CRC_FWD_mask | mmac_cc_TX_ADDR_INS_mask;
```

Yukarıdaki kayıtçılardan 4. bölümde bahsedilmişti. Yukarıdaki program parçası ile MAC modülüne MAC adresinin atanması, modülün fiziksel arabirim ile çalışacağı hız, alma ve gönderme kayıtçılarına v.b. atanacak değerler belirlenmiştir. Burada en önemli kayıtçı, 32 bitlik COMMAND_CONFIG kayıtçısıdır. Tablo 6.1 COMMAND_CONFIG kayıtçısına atanan değeri bitler halinde göstermektedir [57].

Tablo 6.1 COMMAND_CONFIG kayıtçı içeriği

31..25	**24**	**23**	**22**	**21**	**20**	**19**	**18..16**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	1	0	0	1	1

Tablo 6.1' de görüleceği üzere 0,1,4,5,6, ve 9. bit yani TX_ENA, RX_ENA, PROMIS_EN, PAD_EN, CRC_FWD ve TX_ADDR_ING bitlerine 1 değeri atanmıştır. Diğer bitlere ise 0 değeri atanmıştır. Örneğin 0. bite 1 değerinin atanması modülün çerçeve alabileceğini gösterir iken 3. bitin 0 olarak atanması ile modülün 100 Mbit hızında çalışacağı belirtilmektedir[55]. COMMAND_CONFIG kayıtçısının bu şekilde konfigürasyonu ile paket alma ve gönderme işlemleri gerçekleştirilebilir. 10/100 Ethernet IP MAC modülü için en önemli bitlerden bazıları 16., 17., 18. bitlerdir. Bu bitlerin tamamının sıfır olması durumunda, MAC core'un MAC_0 ve MAC_1 kayıtçılarına yüklenmiş olan MAC adresleri çerçeve gönderme ve alma işlemlerinde kullanılır [58].

Yukarıdaki gibi bir konfigürasyondan sonra MAC modülünün paket alma/gönderme yapabilmesi için gerekli olan fonksiyonlar yazılmalıdır.

4. Paket gönderme için gerekli fonksiyon aşağıda verilmiştir Bu fonksiyon Avalon sistem fonksiyonlarını kullanarak Şekil 4.13' de verilen akış şemasını gerçekleştirmektedir [59].

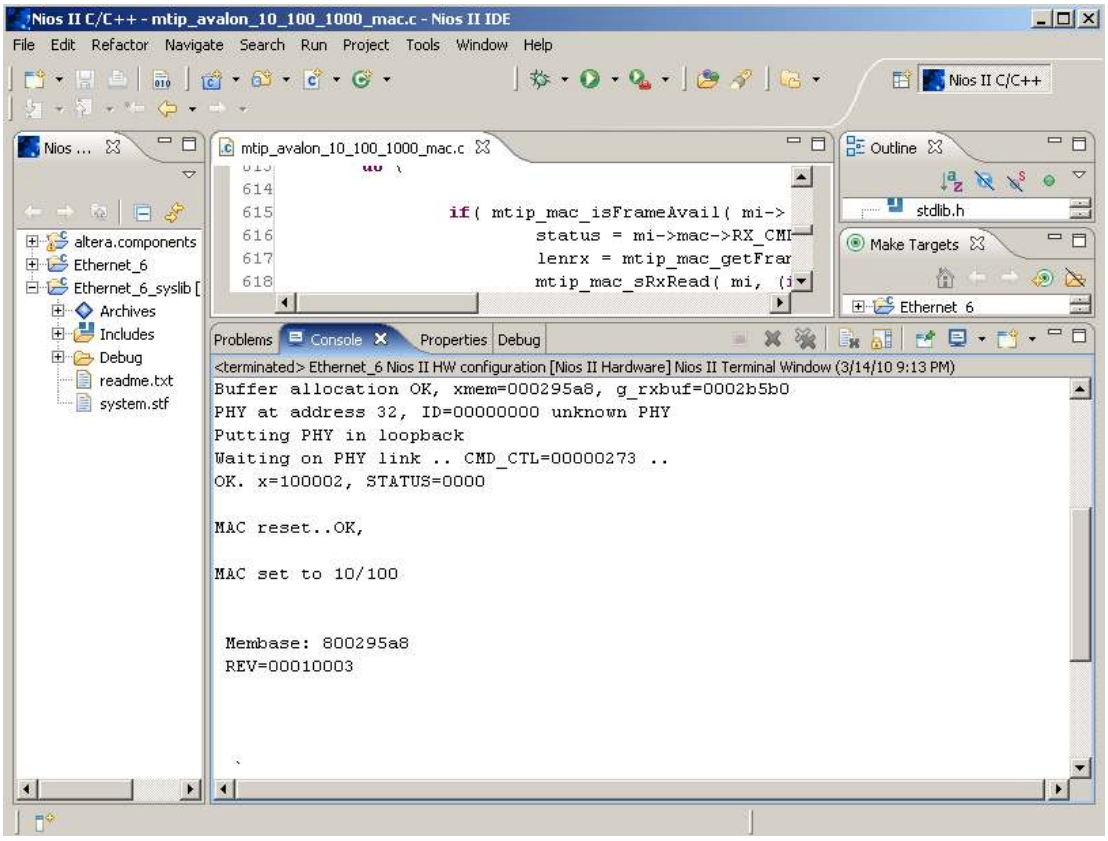
Her bir donanım için kullanılacak Avalon sistem fonksiyonları IORD ( ) ve IOWR ( )'dir. Her iki fonksiyon tasarlanmış sistemdeki tüm donanımlar için kullanılabilir. Örneğin IORD_ALTERA_AVALON_DMA_STATUS () fonksiyonu DMA' nın status kayıtçısına veri yazmak için kullanılırken, IOWR_ALTERA_AVALON_DMA_LENGTH() fonksiyonu DMA' nın ne kadarlık bir veriyi donanımlar arasında transfer edeceğine dair bayt cinsinde değeri *length* kayıtçısına yazar.

```
//Paket Gönderme Fonksiyonu
int  mtip_mac_sTxWrite( mtip_mac_trans_info *mi,  int *src, int length) {
int timeout=1,  base = mi->dma;
        while ( (IORD_ALTERA_AVALON_DMA_STATUS (base) &
ALTERA_AVALON_DMA_STATUS_BUSY_MSK) ||
            (IORD(&mi->mac->TX_CMD_STAT,0)&
mmac_tcs_FRAME_COMPLETE_mask)) {
        if(timeout++ == 1000000) return -2;
      IOWR_ALTERA_AVALON_DMA_CONTROL (base, 0);
      IOWR_ALTERA_AVALON_DMA_STATUS (base, 0);
      IOWR_ALTERA_AVALON_DMA_WADDRESS (base, (alt_u32) mi->txFIFO);
      if( ((int)src & 0x02) != 0 ) {
      IOWR_ALTERA_AVALON_DMA_RADDRESS (base, ((alt_u32)src) & 0xfffffffc);
      IOWR_ALTERA_AVALON_DMA_LENGTH (base, (length+5) & 0xfffc );
      IOWR(&mi->mac->TX_CMD_STAT,0, (length  |
mmac_tcs_FRAME_COMPLETE_mask |mmac_tcs_SHIFT16_mask ));
      } else {
      IOWR_ALTERA_AVALON_DMA_RADDRESS (base, (alt_u32) src);
      IOWR_ALTERA_AVALON_DMA_LENGTH (base, (length+3) & 0xfffc );
      IOWR(&mi->mac->TX_CMD_STAT,0, (length  |
mmac_tcs_FRAME_COMPLETE_mask));       }
      IOWR_ALTERA_AVALON_DMA_CONTROL (base,
              ALTERA_AVALON_DMA_CONTROL_WORD_MSK |
              ALTERA_AVALON_DMA_CONTROL_GO_MSK   |
              ALTERA_AVALON_DMA_CONTROL_WCON_MSK |
              ALTERA_AVALON_DMA_CONTROL_LEEN_MSK);
      timeout=0;
      while(IORD_ALTERA_AVALON_DMA_STATUS (base) &
                    ALTERA_AVALON_DMA_STATUS_BUSY_MSK) {
        if(timeout++ == 1000000) return -1; }
      IOWR_ALTERA_AVALON_DMA_CONTROL (base, 0);
      IOWR_ALTERA_AVALON_DMA_STATUS (base, 0);
      return 0;}
```

5. Paket alma için gerekli olan fonksiyon, C programlama dilinde yazılmış şekliyle aşağıda verilmiştir. Bu fonksiyon Avalon sistem fonksiyonlarını kullanarak Şekil 4.14' de verilen akış şemasını gerçekleştirmektedir.

```
//Paket Alma Fonksiyonu
int  mtip_mac_sRxRead( mtip_mac_trans_info *mi, int *dest,int length) {
int timeout=0, cmd;
int base = mi->dma;
     while ( (IORD_ALTERA_AVALON_DMA_STATUS (base) &
ALTERA_AVALON_DMA_STATUS_BUSY_MSK))       {
            if(timeout++ == 1000000) return -1;         }
      IOWR_ALTERA_AVALON_DMA_CONTROL (base, 0);
      IOWR_ALTERA_AVALON_DMA_STATUS (base, 0);
      IOWR_ALTERA_AVALON_DMA_RADDRESS (base, (alt_u32) mi->rxFIFO);
      IOWR_ALTERA_AVALON_DMA_WADDRESS (base, (alt_u32) dest);
      IOWR_ALTERA_AVALON_DMA_LENGTH (base, (length+3) & 0xfffc );
      IOWR_ALTERA_AVALON_DMA_CONTROL (base,
ALTERA_AVALON_DMA_CONTROL_WORD_MSK |
ALTERA_AVALON_DMA_CONTROL_GO_MSK   |
ALTERA_AVALON_DMA_CONTROL_RCON_MSK |
ALTERA_AVALON_DMA_CONTROL_LEEN_MSK);
      cmd = IORD(&mi->mac->RX_CMD_STAT,0) |
          mmac_rcs_READ_CMD_mask;
      IOWR(&mi->mac->RX_CMD_STAT,0, cmd );
      while (IORD_ALTERA_AVALON_DMA_STATUS (base) &
ALTERA_AVALON_DMA_STATUS_BUSY_MSK);
      IOWR_ALTERA_AVALON_DMA_CONTROL (base, 0);
      IOWR_ALTERA_AVALON_DMA_STATUS (base, 0);
      printf("RX----------\n");
      printf("Gönderilen Paket Sayısı: %d",IORD(&mi->mac->aFramesTransmittedOK,0));
      printf("\nAlınan Paket Sayısı: %d",IORD(&mi->mac->aFramesReceivedOK,0));
      printf("\nToplam Alınan Paket Sayısı: %d",IORD(&mi->mac->etherStatsPkts,0));
      return 0; }
```

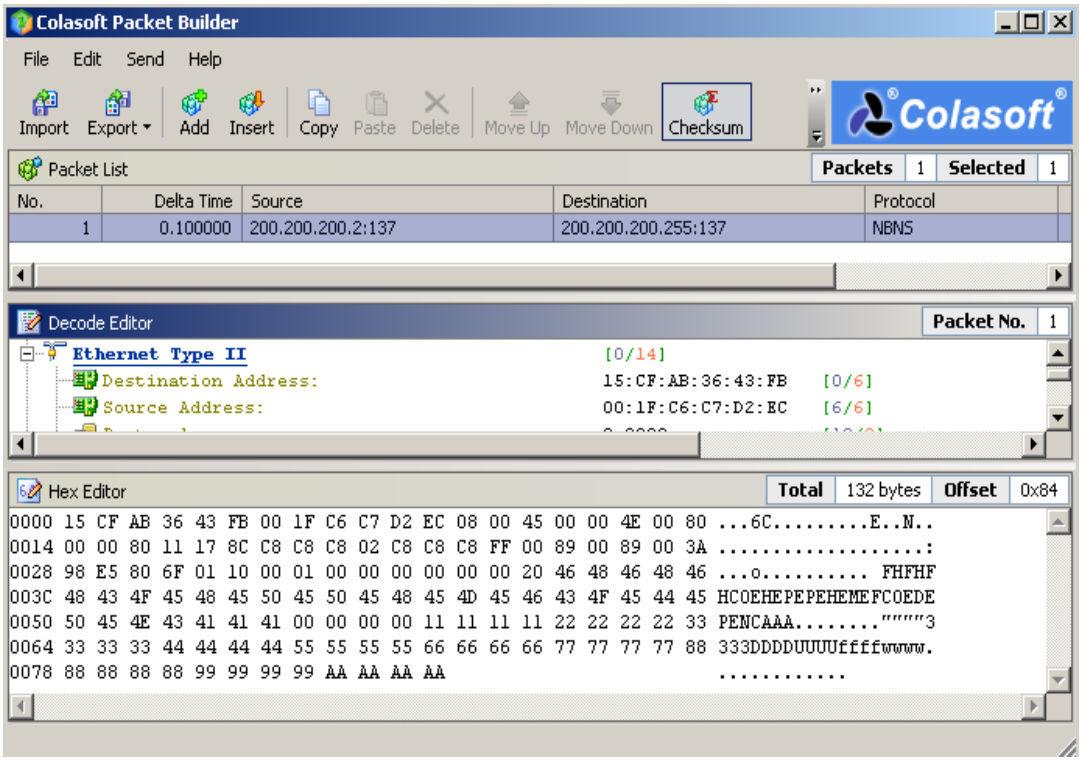
Paket alma/gönderme için yazılan program derlendikten sonra karta yüklü donanımın program hafızası üzerine yazılır. Sistem artık bilgisayar ağından paket alma/gönderme işlemi için hazır hale gelmiştir. Bu durum Şekil 6.3'de görülmektedir.



Şekil 6.3 Paket alma gönderme için sistemin bekleme konumu

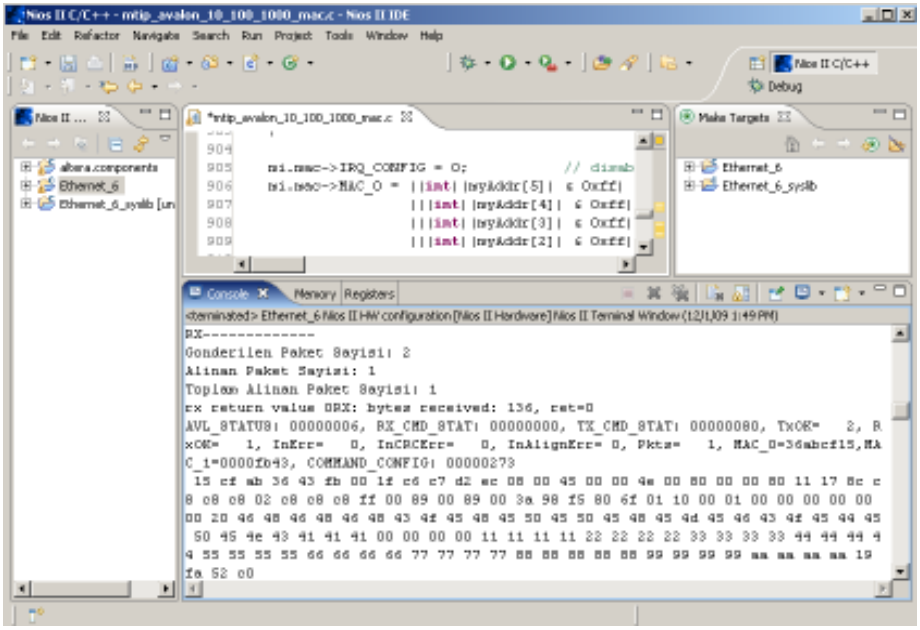
6.2 Paket Gönderme/ Alma

Tasarlanan gömülü sisteme gönderilecek paketler, ColaSoft Packet Builder paket oluşturma yazılımı ile oluşturulmuştur. Oluşturulan paketlerin hedef MAC adresleri gömülü sistemdeki 10/100 Ethernet MAC modülünün MAC adresini (15 CF AB 36 43 FB), kaynak MAC adresleri ise kullanılan bilgisayarın MAC adresini (00 1F C6 C7 D2 EC) göstermektedir. Şekil 6.4'de ColaSoft Packet Builder ile oluşturulmuş bir UDP paketi gösterilmektedir. Oluşturulan paketin başlık yapısından MAC adresleri, uzunluk, IP adresleri, port numaraları ve gönderilecek veri, paket yapısından kolaylıkla elde edilebilir.



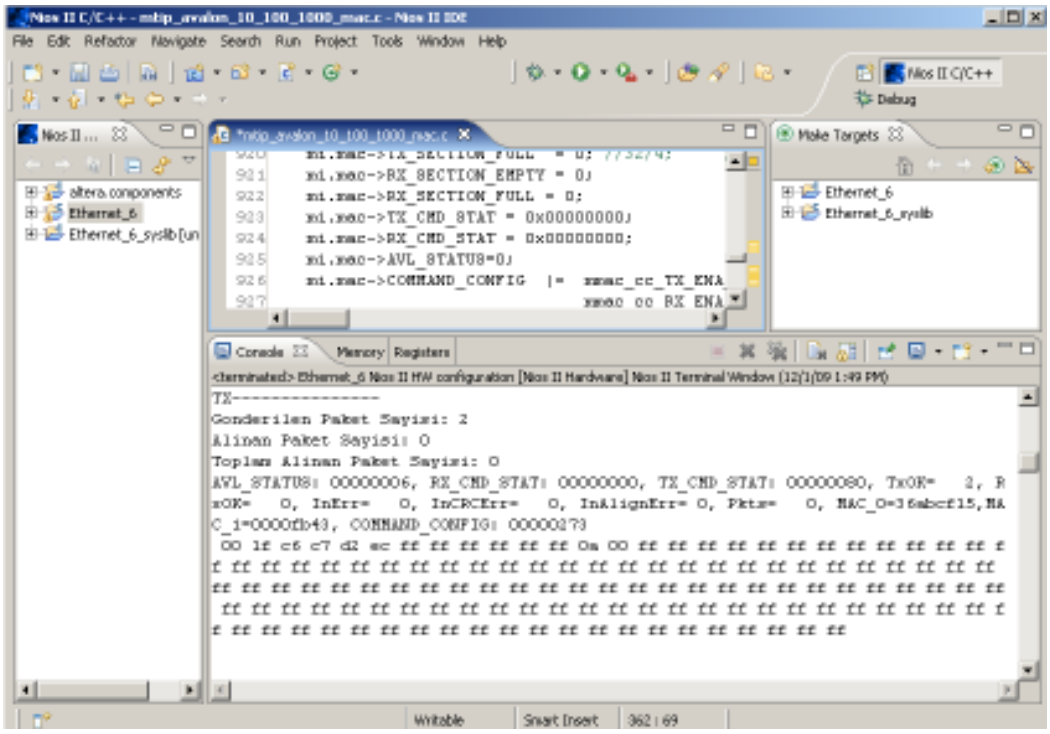
Şekil 6.4 Colasoft ile gönderilen UDP paketi

Oluşturulan bu paket bilgisayardan, gömülü sistemin bulunduğu karta (Cross Kablo ile bağlı) gönderildiğinde, gömülü sistem gönderilen paketi Şekil 6.5' de gösterildiği gibi almıştır.



Şekil 6.5 STS tarafından yakalanan UDP paketi

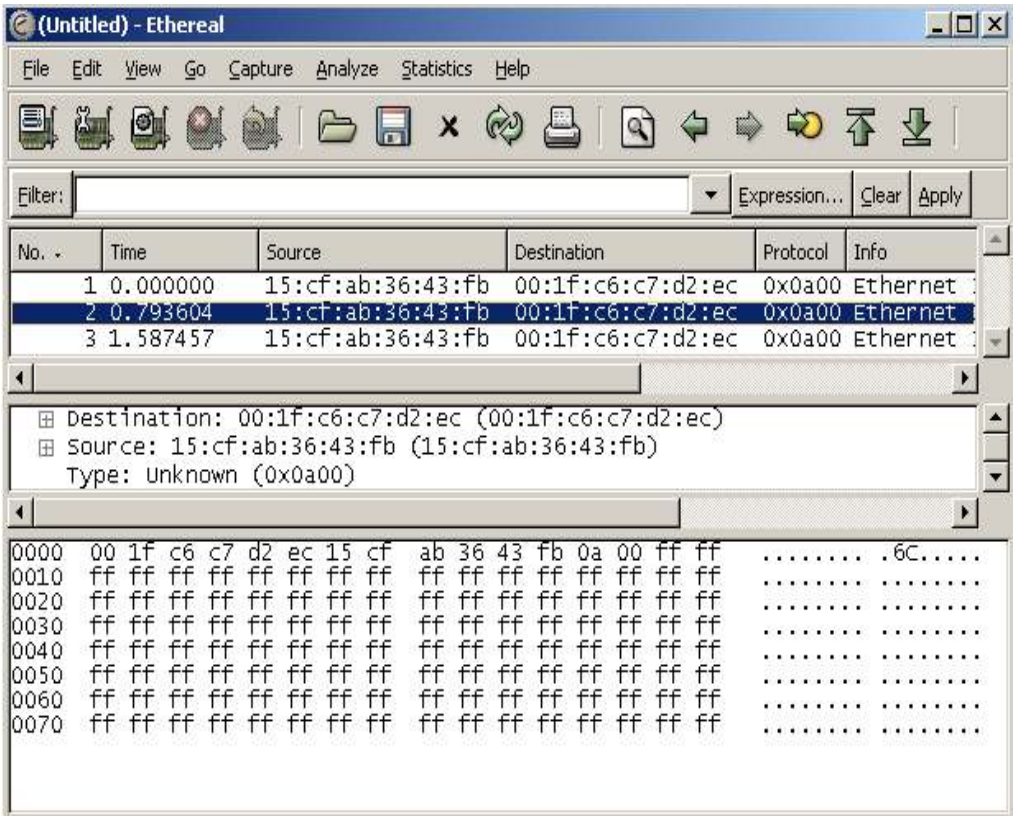
Geliştirilen STS'den bilgisayara paket göndermek için öncelikle hafızada tüm bitleri 0xFF olan bir paket oluşturulmuştur. Şekil 6.6'da görüleceği üzere hedef MAC adres alanı 00 1F C6 C7 D2 EC (paketi alacak bilgisayarın MAC adresi) iken kaynak MAC adresi olarak FF FF FF FF FF FF yazılmıştır.



Şekil 6. 6 Hafıza alanında oluşturulup bilgisayara gönderilen paket

10/100 Ethernet MAC Core, kendi MAC_0 ve MAC_1 (Tablo 4.4'deki) kayıtçılarına önceden kaydedilmiş MAC adreslerini paketin kaynak MAC adres alanına yazdıktan sonra paketi gönderir[57].

Şekil 6.7' de gösterildiği gibi Etheral paket programı yardımıyla STS'den bilgisayara gelen paketler yakalanmıştır. Etheral tarafından yakalanan pakette hem bilgisayarın hem de kartın MAC adresleri görülmektedir.



Şekil 6.7 Etheral programı ile yakalanan paket

Geliştirilen Gömülü STS' nin bilgisayar veya ağ ile paket alış veriş işlemini doğru bir şekilde gerçekleştirdiği böylelikle görülmüştür. Bu aşamadan sonra gömülü STS'nin paketleri sınıflandırması için oluşturulan yazılımlar yedinci bölümde detaylı olarak açıklanacaktır.

7. STS DONANIMI İLE PAKET SINIFLANDIRMA

Bölüm 6'da gömülü STS donanımının, bağlanacağı ağ ortamından paket alış-verişi yapabilmesi için, gerekli konfigürasyon ve paket alma gönderme fonksiyonlarını yerine getiren programlama aşamaları ve örnek bir paket alma/gönderme işleminin gerçekleştirilmesi açıklanmıştı.

Bu bölümde ise gömülü STS'nin, çevrim-içi çalışarak bilgisayar ağından gelen paketleri Karar ağacı, Naive Bayesian ve YSA algoritmalarına göre ayrı ayrı değerlendirerek sınıflandırabilmesi için gerekli süreç açıklanacaktır. Bu süreç Şekil 6.2' deki akış şemasının 8., 9. ve 10. adımlarını kapsamaktadır.

Öncelikle, paketlerin sınıflandırma algoritmaları tarafından değerlendirilebilmesi için ağdan gelen paketlerin bir ön işlemden geçirilmesi gerekir. Bu ön işlemde sınıflandırma algoritmalarında kullanılmak üzere, paketin karakteristik bilgisini taşıyan (MAC adresleri, Ip numaraları, port numaraları, paket boyutları, paket tipi, paket başlıklarındaki kontrol bayraklarının durumları, Acil paket sayısı, Bağlantı süresi v.s) başlık bilgisi, zaman tabanlı ve bağlantı tabanlı bilgi özetleri elde edilir. Bu önişlem süreci akış şemasındaki Adım.8'de gerçekleştirilir.

Adım.9'da ise Karar ağacı, Naive Bayesian ve YSA algoritmalarına göre paket sınıflandırması yapabilmek için C/C++ ortamında programlar oluşturulmuştur. Bu algoritmaların çalışma mantıkları Bölüm 2'de verilmiştir. Çevrim-içi çalışma için oluşturulan bu programların girdileri Adım.8'de oluşturulmuş paket özetleridir. Gömülü STS, belirtilen algoritmalara göre paket özetlerini değerlendirerek paket sınıfını belirleyip karar verecektir. Bu işlem ise Adım.10'u oluşturmaktadır.

Nios IDE 8.0 ortamında ayrı ayrı yazılan bu programlar Ek.2-3-4'te verilmiştir. Programlar Nios IDE 8.0 ortamında derlenip Nios işlemcinin Assembler koduna dönüştürülüp, STS donanımının program hafızasına yüklenmiştir.

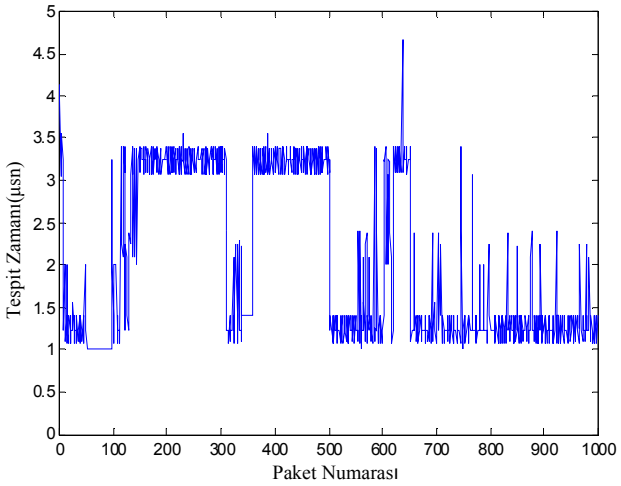
Gerçekleştirilen STS'nin testlerinin yapılması için Bölüm 2.4 te bahsedilen KDD CUP 99 veri setindeki 1000 paket, karar ağacı yöntemiyle gerçek zamanda sınıflandırılmıştır. Bu durumda hazır paket özetleri kullanıldığında Adım.8'deki işlemler atlanarak, sınıflandırma programının testi gerçekleştirilmiştir.

Çevrim dışı çalışmada olduğu gibi, Çevrim içi çalışmada da; 23, 5 ve 2 saldırı sınıf sayısı için testler yapılmıştır. Elde edilen 2, 5, 23 sınıf sayısı için paket sınıf tespit

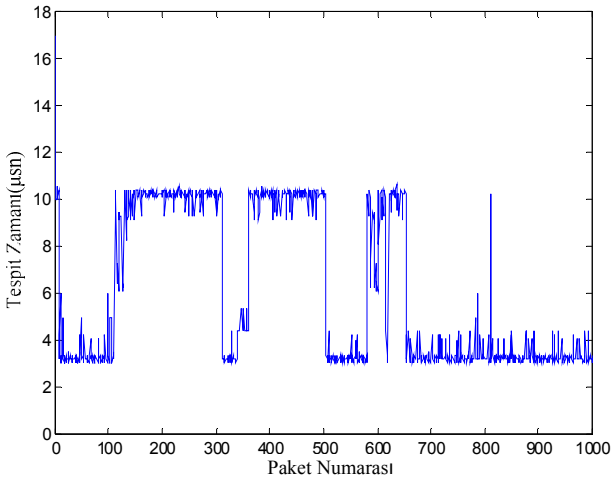
zamanları, Şekil 7.1, Şekil 7.2 ve Şekil 7.3'de verilmiştir. Bu şekillerden görüleceği üzere sınıf sayısının azaltılması ile paket tespit zamanı azalmaktadır. Tablo 7.1 ise 23, 5 ve 2 sınıf için doğru ve yanlış tespit edilen paketlerin sayısını göstermektedir.

Tablo 7.1 Doğru ve yanlış olarak tespit edilen paket sayıları

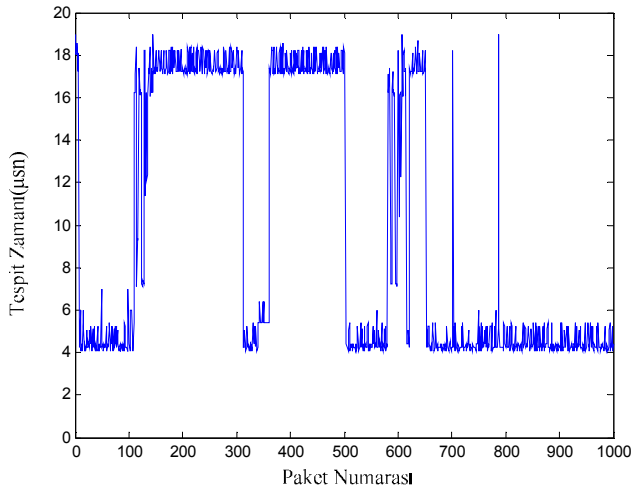
	Doğru Tespit Edilen Paket Sayısı	Yanlış Tespit Edilen Paket sayısı
2 Sınıf	999	1
5 Sınıf	996	4
23 Sınıf	998	2



Şekil 7.1 2 sınıf için gerçek zamanda paket sınıflarının tespit zaman değişimi



Şekil 7.2 5 sınıf için paket sınıflarının tespit zaman değişimi



Şekil 7.3 23 sınıf için paket sınıflarının tespit zaman değişimi

7.1 Paket Başlığından Paket Özetlerinin Elde Edilmesi

Bu tezde gerçekleştirilen çevrim içi çalışan sınıflandırma algoritmalarının girdileri olarak kullanılacak paket özetleri, ağdan gelen sınıflandırılacak paketlerin başlık bilgileri kullanılarak elde edilmiştir. Bu işlem, Şekil 6.2'deki akış şemasının 8. adımının gerçekleştirilmesidir.

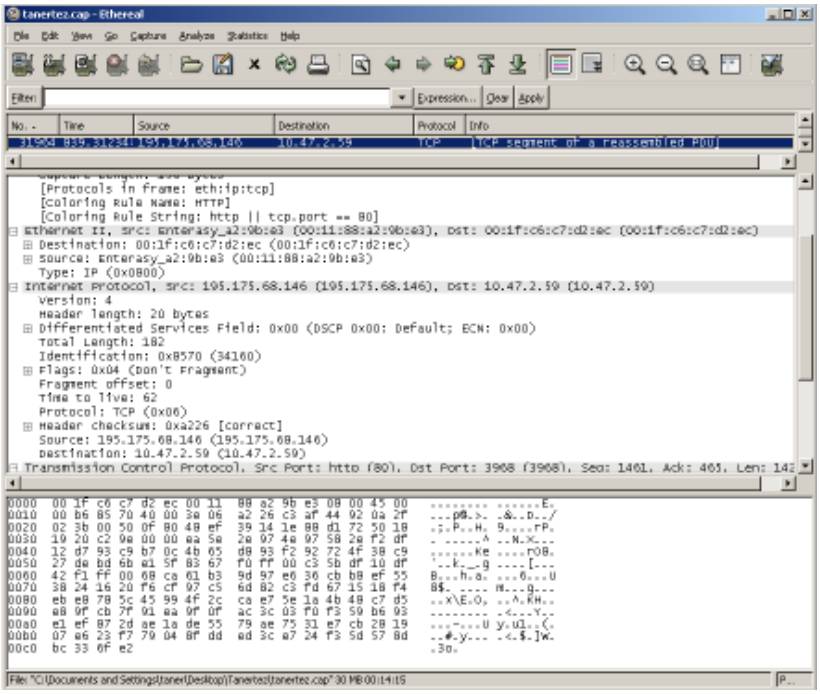
Paket özetleri paketin başlık yapısındaki kaynak ve hedef MAC adresleri, kaynak ve hedef IP adresleri, kaynak ve hedef port numaraları, paketin tipi ve paketin uzunluğunu içermektedir. Ağdan elde edilen herhangi bir paket önceden belirlenmiş hafıza bölgesine kaydedilir. Paket başlık yapısındaki alanlar normalize edilerek elde edilen paket özellikleri belirlenmiş bir hafıza bölgesine kaydedilir. Normalizasyon şartları aşağıdaki gibidir.

Alınan paketin hedef ve kaynak MAC adresleri, 00:00:00:00:00:00 ve 10:00:00:00:00:00 arasında ise 1 değil ise 2 olarak normalize edilmiştir. Kaynak ve Hedef IP adresleri ise IPV4 formatına göre A sınıfı IP adresi için 1, B sınıfı IP adresi için 2, diğer IP adresleri için 3 olarak belirlenmiştir. Paketin kaynak ve hedef port numaralarının normalize edilmesi ise en çok kullanılan port numaralarına göre yapılmıştır. 0-256 arası port numaraları 1, 256-1000 arası port numaraları ise 2, diğer port numaralarının normalize değerleri ise 3 olarak belirlenmiştir. Paket başlığındaki protokol alanı TCP, UDP, ICMP gibi değerler aldığından dolayı TCP için 0x06, UDP için0x11'e karşılık olarak 1 ve 2 değerleriyle normalize edilmiştir. Paketin uzunluk alanı ise en küçük paket boyutu 64 bayt ve 256 bayt arası 1, 257 ile 1000 bayt arası 2 ve diğer paket uzunluklarıysa 3 olarak

normalize edilmiştir. Tablo 7.2'de paket başlık alanlarının normalizasyon değerleri verilmiştir.

Tablo 7.2 Paket başlık alanları ve tanım aralıkları

Alan adı	Tanımlı aralık	Normalize değer
Hedef ve Kaynak MAC Adresleri	00:00:00:00:00:00-10:00:00:00:00:00	1
	10:00:00:00:00:00-FF:FF:FF:FF:FF:FF	2
Hedef ve Kaynak IP Adresleri	0.0.0.0 – 126.255.255.255	1
	128.0.0.0 – 191.255.255.255	2
	192.0.0.0 – 255.255.255.255 ve diğer	3
Hedef ve Kaynak Port Numarası	0 - 100	1
	100 - 1000	2
	1001-65536	3
Protokol Tipi	TCP:6	1
	UDP:17	2
	Diğer	3
Uzunluk	64-256	1
	257- 1000	2
	Diğer	3



Şekil 7.4 Ağdan yakalanmış bir TCP paketi

Şekil 7.4' de ağdan yakalanmış bir örnek paket gösterilmektedir. Bu paketin STS'de çevrim içi olarak işlenebilmesi için normalize edilerek özetinin elde edilmesi gereklidir. Tablo 7.3 yukarıda anlatılanlara göre elde edilmiş paket özeti görülmektedir. Paket özeti Hedef MAC adresi(H.MAC), Kaynak MAC adresi(S.MAC), Protokol, Kaynak IP numarası(K.IP), Hedef IP numarası(H.IP) Kaynak Port numarası (K.Port) ve Hedef Port numarası (H.Port) ve Uzunluk alanlarından oluşmaktadır.

Tablo 7.3 Örnek paket özeti

H.MAC	S.MAC	Protokol	K.IP	H.IP	K.Port	H.Port	Uzunluk
1	1	1	3	1	3	1	1
001188a29be3	001fc6c7d2ec	c3af4492	0a2f022b	0050	0f80	06	00b6

7.2 Eğitim Verilerinin Oluşturulması

Paket sınıflandırma algoritmalarında, eğitim verileri olarak kullanılmak üzere, paket özetleri Bölüm 7.1'de açıklanan biçimde yapay olarak oluşturulmuştur. Tablo 7.4' te bu paket özeleri ve sınıfları verilmiştir. Bu paketlerin 17 adedi saldırı, 34 adedi ise normal sınıfına atanmıştır.

Tablo 7.4 Eğitim verisi

H.MAC	K.MAC	Protokol	K.IP	H.IP	K.Port	H.Port	Uzunluk	Sınıf
1	1	1	1	3	2	3	3	Saldırı
2	1	3	1	3	3	3	3	Normal
1	1	1	2	2	1	2	1	Saldırı
2	1	3	2	2	3	3	2	Normal
1	1	2	3	1	2	2	2	Normal
2	1	3	3	1	3	3	3	Saldırı
1	1	2	1	3	1	1	1	Normal
2	2	2	1	3	2	2	3	Saldırı
1	2	3	2	2	3	3	3	Saldırı
2	2	1	2	2	1	3	3	Normal
1	2	1	3	1	2	2	2	Saldırı
2	2	1	3	1	3	3	1	Normal
1	2	3	1	3	3	3	2	Normal
2	2	2	1	3	2	2	2	Saldırı
1	1	3	2	2	3	3	3	Saldırı
2	1	2	2	2	2	3	1	Normal
1	1	1	3	1	3	3	2	Saldırı
2	1	3	3	3	3	3	3	Normal
1	1	3	1	1	3	3	2	Saldırı
2	1	1	1	1	2	3	1	Normal
1	1	2	2	2	3	3	1	Normal
2	2	3	2	2	3	3	1	Normal
1	2	2	3	3	1	1	2	Normal
2	2	2	3	3	2	2	3	Saldırı
1	2	3	1	3	3	3	1	Normal
2	2	1	1	3	1	3	2	Saldırı
1	2	1	2	2	2	1	2	Normal
2	2	1	2	2	1	1	3	Normal
1	1	3	3	1	3	3	3	Normal
2	1	3	3	1	3	3	3	Normal
1	1	3	1	2	3	3	3	Normal
2	1	2	1	2	2	2	2	Normal
1	1	1	2	1	1	3	1	Normal
2	1	3	2	1	3	3	1	Saldırı
1	1	1	3	3	3	3	2	Normal
2	2	1	3	3	3	3	2	Normal
1	2	2	1	3	3	3	2	Normal
2	2	2	1	3	2	3	2	Normal
1	2	2	2	1	1	1	1	Normal
2	2	3	2	1	3	3	1	Normal
1	2	1	3	2	3	1	2	Saldırı
2	2	1	3	2	3	2	2	Saldırı
1	1	3	1	2	3	3	2	Normal
2	1	1	1	2	2	2	3	Normal
1	1	2	2	3	2	1	3	Normal
2	1	2	2	3	2	3	1	Normal
1	1	3	3	1	3	3	1	Normal
2	1	3	3	1	3	3	2	Normal
1	1	1	1	2	1	2	2	Normal
2	2	1	1	2	3	3	2	Saldırı
1	2	3	2	3	3	3	3	Normal
2	2	3	2	3	3	2	3	Normal
1	2	2	3	1	3	1	1	Normal
2	2	2	3	1	3	3	1	Saldırı

7.3 Naive Bayesian Temelli Çevrim İçi Paket Sınıflandırma

Eğitim verisi, 2. bölümde matematiksel temelleri verilen Naive Bayesian istatistiksel sınıflandırıcı ile sınıflandırılmıştır. Sınıflandırma işleminin nasıl yapılacağı aşağıdaki örnekte gösterilmiştir.

Ağdan alınan X paketinin, paket özetinin Tablo 7.4' deki 2. kayıt (2,1,3,1,3,3,3,3) olduğunu varsayalım. Bu durumda Tablo 7.4'e göre Saldırı ve Normal sınıflar için olasılıklar aşağıdaki gibi olacaktır.

P(Sınıf="Saldırı")=17/54

P(Sınıf="Normal")=37/54

Her bir paket alanının şartlı olasılıkları için aşağıdaki değerler elde edilir.

P(H.MAC="2" | Sınıf="Saldırı")=9/17

P(H.MAC="2" | Sınıf="Normal")=18/37

P(S.MAC="1" | Sınıf="Saldırı")=7/17

P(S.MAC="1" | Sınıf="Normal")=21/37

P(Protokol="3" | Sınıf="Saldırı")=5/17

P(Protokol="3" | Sınıf="Normal")=15/37

P(K.IP="1" | Sınıf="Saldırı")=6/17

P(K.IP="1" | Sınıf="Normal")=12/37

P(H.IP="3" | Sınıf="Saldırı")=5/17

P(H.IP="3" | Sınıf="Normal")=13/37

P(K.Port="3" | Sınıf="Saldırı")=10/17

P(K.Port="3" | Sınıf="Normal")=21/37

P(H.Port="3" | Sınıf="Saldırı")=10/17

P(H.Port="3" | Sınıf="Normal")=25/37

P(Uzunluk="3" | Sınıf="Saldırı")=6/17

P(Uzunluk="3" | Sınıf="Normal")=11/37

Bu olasılıkları kullanarak X paketinin sınıfı aşağıdaki gibi elde edilir.

P(X|Sınıf="Saldırı")=9/17*7/17*5/17*6/17*5/17*10/17*10/17*6/17=8,128.10^{-4}

P(X|Sınıf="Normal")=18/37*21/37*15/37*12/37*13/37*21/37*25/37*11/37=14,54. 10^{-4}

P(X|Sınıf="Saldırı").P(Sınıf="Saldırı)= 8,128.10^{-4}*17/54=2,25. 10^{-4}

P(X|Sınıf="Normal").P(Sınıf="Normal)= 14,54. 10^{-4}*37/54=9,96. 10^{-4}

Elde edilen sonuçlara göre *Normal* sınıfının olasılığı *Saldırı* sınıf olasılığından daha büyük olduğu için X (2,1,3,1,3,3,3,3) paket özetine sahip paketler *Normal* sınıfına ait olmalıdır..

Bu açıklamalara göre eğitim verisi için elde edilen karışıklık matrisi Tablo 7.5' teki gibidir.

Tablo 7.5 Naive Bayesian için karışıklık matrisi

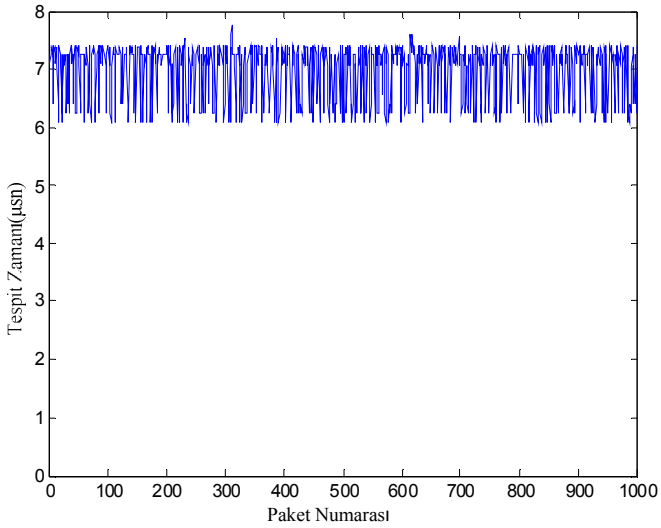
		Tahmin	
		Negatif	Pozitif
Gerçek	Negatif	2	15
	Pozitif	4	33

Tablo 7.5 kullanılarak Naive Bayesian sınıflandırma için başarım değerlendirmesi ise Tablo 7.6' teki gibi olacaktır.

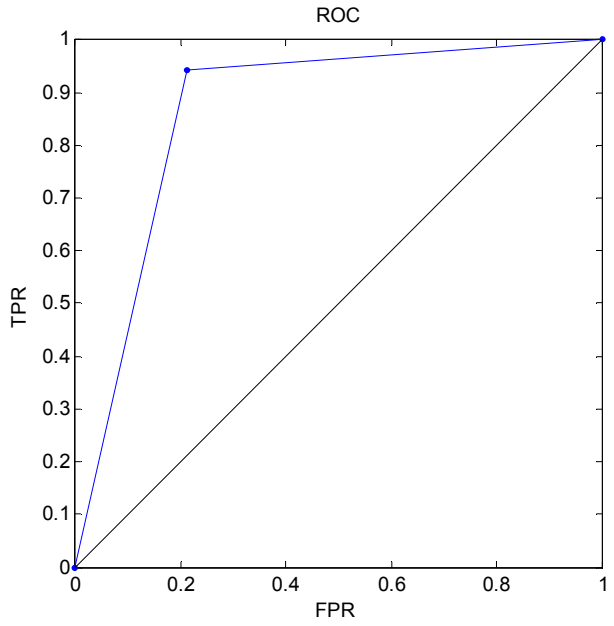
Tablo 7.6 Naive Bayesian başarımı

Doğruluk	TPR	FPR	TNR	FNR	Kesinlik
$\frac{a+d}{a+b+c+d}$	$\frac{d}{c+d}$	$\frac{b}{a+b}$	$\frac{a}{a+b}$	$\frac{c}{c+d}$	$\frac{d}{b+d}$
0,648	0,891	0,882	0,117	0,108	0,687

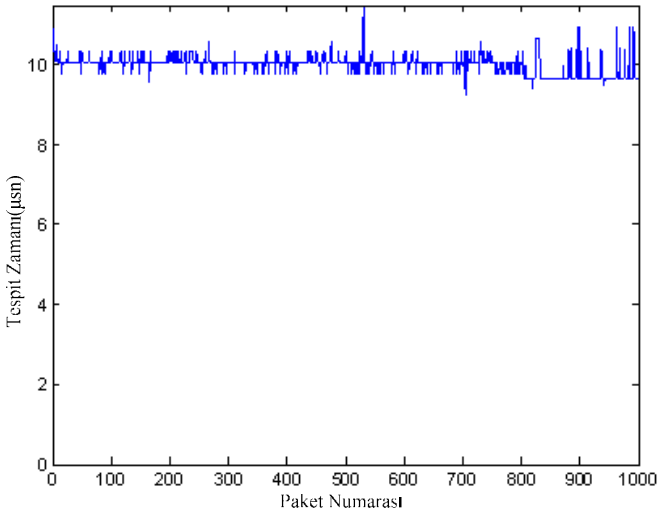
Programlanabilir gömülü sistem kullanılarak paketlerin sınıfını çevrim içi olarak belirleyebilmek için işlemcinin Nios IDE 8.0 ortamında Naive Bayesian sınıflandırıcı ile programlanması gerçekleştirilmiştir. STS sistemi ile ağdan akan 1000 paket sınıflandırılmıştır. Bu paketlerden 718 adedi normal 282 adedi ise saldırı olarak sınıflandırılmıştır. Her bir paketin sınıflandırılma zamanı ve sınıfları belirlenmiştir. Ayrıca Naive Bayesian sınıflandırıcının performansını ölçmek için Tablo 7.6'ya ilaveten ROC eğrisi elde edilmiştir. Şekil 7.5 paketlerin sınıflandırılma zamanını Şekil 7.6 ise ROC eğrisinin değişimini göstermektedir. Ayrıca paketlerin sınıflandırma zamanlarının tespiti için 3 GB ve çift işlemcili 2.0 Ghz hızına sahip bir diz üstü bilgisayarda JAVA programlama dilinde testler yapılarak Şekil 7.7 deki tespit zamanı değişimi çevrim dışı elde edilmiştir.



Şekil 7.5 Naive Bayesian için gerçek zamanda paket sınıfının tespit zaman değişimi



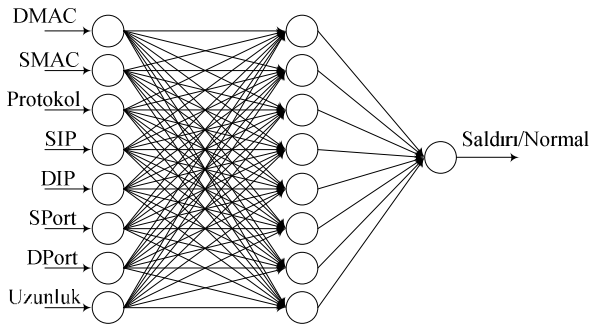
Şekil 7.6 Naive Bayesian için ROC eğrisi



Şekil 7.7 Naive Bayesian için JAVA ortamında paketlerin tespit zaman değişimi

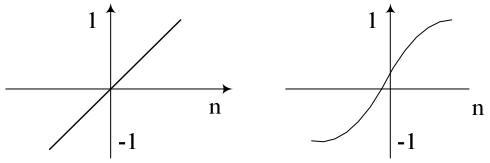
7.4 YSA Temelli Çevrim İçi Paket Sınıflandırma

2.Bölümde temel yapısı anlatılan YSA kullanılarak paketlerin normal veya saldırı özelliğinde olduğunun belirlenmesi için Şekil 7.8' deki YSA yapısı kullanılmıştır.



Şekil 7.8 YSA yapısı

Ağın eğitilmesi için Şekil 7.9 da gösterilen Hiperbolik Tanjant Sigmoid ve Lineer Transfer fonksiyonları kullanılmıştır.



Şekil 7.9 Lineer ve Hiperbolik Tanjant Sigmoid transfer fonksiyonları

Ağın eğitilmesinde geriye yayılım algoritması kullanılarak ağırlıklar güncellenmiştir. Ağın eğitimi için hata değeri olarak 0,01 ve öğrenme oranı olarak ta 0,005 değerleri alınmıştır. Tablo 7.7 Şekil 7.8'deki ağın eğitilmesi sonucu elde edilen, giriş katmanı-ara katman ve ara katman-çıkış katmanı arasındaki ağırlıkları göstermektedir. Tablo 7.7' nin her bir satırı giriş katmanından ara katmandaki sinir hücresine yapılan bağlantının ağırlık değerini göstermektedir.

Tablo 7.7 Katmanlar arası ağırlıklar

Giriş Katmanı – Ara Katman Ağırlıkları							
1.8897	1.1884	0.6062	-0.6350	-0.2461	0.3538	-0.0994	-0.0334
-1.3386	-0.6903	0.3014	0.0972	0.8934	-1.2231	-0.0781	1.3849
1.3100	0.7362	-0.3116	-1.3104	-0.3148	0.1016	-0.9370	2.1111
0.1592	1.5388	1.8455	-1.2284	0.7711	-0.4691	1.0945	-0.1701
1.7359	-0.1937	-0.8316	0.0030	1.1486	0.5596	-0.7042	-0.0796
1.5118	0.3976	-0.9146	-0.7471	0.8699	0.4003	-0.4027	1.3761
-0.6115	-1.5910	0.7869	-1.2458	-0.2896	0.5318	-0.7486	0.3361
-1.9297	-1.1001	-1.6517	1.2177	0.5500	-0.1145	-0.0641	0.1447
Ara Katman – Çıkış Katman Ağırlıkları							
1.3458	0.6297	-1.2064	-0.7031	-1.2567	1.2823	0.5542	-1.1478

Tablo 7.4' teki eğitim verilerinin sınıflandırılması sonucunda karışıklık matrisi Tablo 7.8 ve sınıflandırmanın başarımı Tablo 7.9 deki gibi elde edilmiştir.

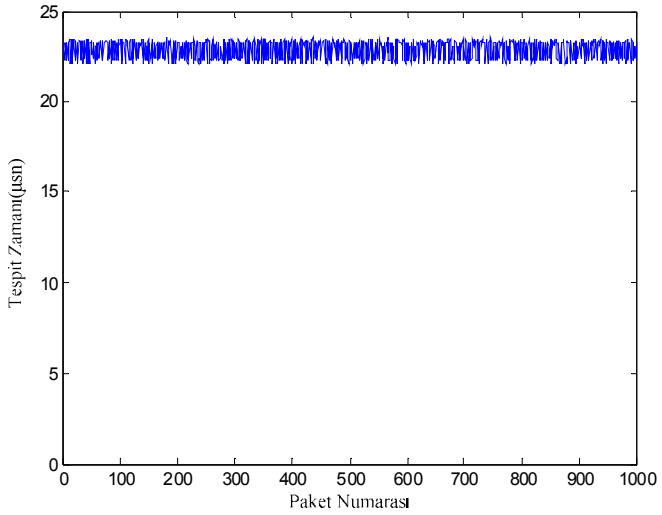
Tablo 7.8 YSA için karışıklık matrisi

		Tahmin	
		Negatif	Pozitif
Gerçek	Negatif	1	16
	Pozitif	0	37

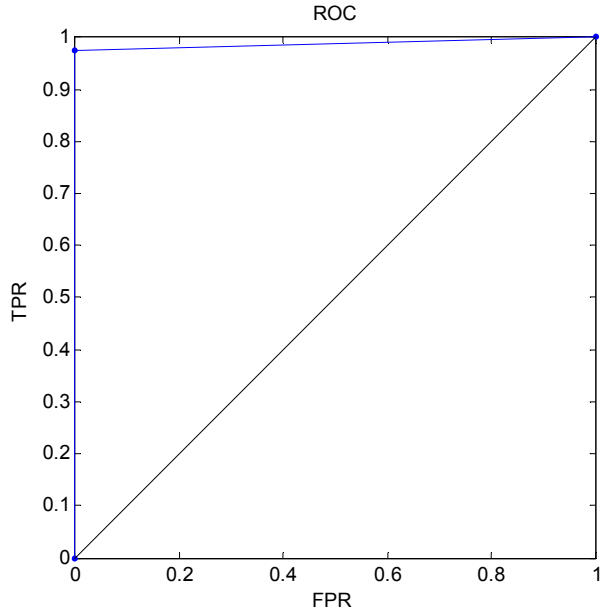
Tablo 7.9 YSA başarımı

Doğruluk	TPR	FPR	TNR	FNR	Kesinlik
$\frac{a+d}{a+b+c+d}$	$\frac{d}{c+d}$	$\frac{b}{a+b}$	$\frac{a}{a+b}$	$\frac{c}{c+d}$	$\frac{d}{b+d}$
0,703	1	0,941	0,058	0	0,698

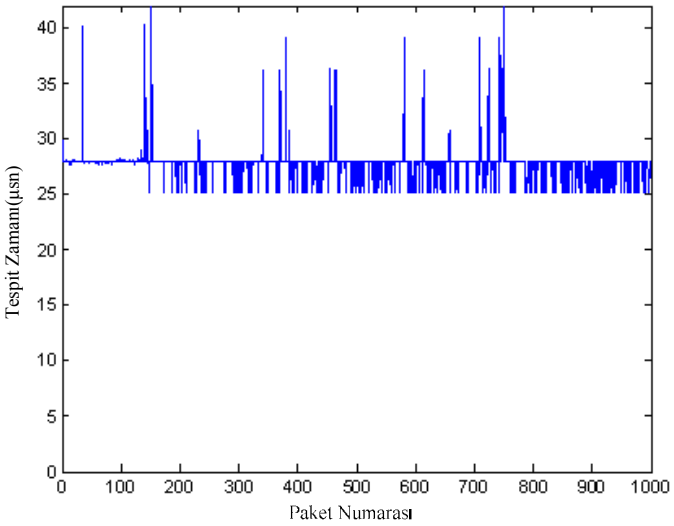
Programlanabilir gömülü sistemin paketlerin sınıfını belirleyebilmesi için, Şekil 7.8 deki Yapay Sinir ağ yapısı eğitim verileriyle öğretilmiştir. Sistem ağdan akan 1000 paket ile teste tabi tutularak paketlerin sınıflandırılma zamanı ve paketlerin sınıfları belirlenmiştir. Bu paketlerden 725 adedi normal 275 adedi ise saldırı olarak tespit edilmiştir. Yapay sinir ağları ile gerçekleştirilen sınıflandırıcının performansını ölçmek için Tablo 7.9' a ilaveten ROC eğrisi elde edilmiştir. Şekil 7.10 paketlerin sınıflandırılma zamanını Şekil 7.11 ise ROC eğrisinin değişimini göstermektedir. Ayrıca Java ortamında çevrim dışı olarak paketler teste tabi tutulmuş ve Şekil 7.12'deki değişim elde edilmiştir.



Şekil 7.10 YSA için gerçek zamanda paket sınıfının tespit zaman değişimi



Şekil 7.11 YSA için ROC eğrisi



Şekil 7.12 YSA için JAVA ortamında paketlerin tespit zaman değişimi

7.5. Karar Ağacı Temelli Çevrim İçi Paket Sınıflandırma

Bölüm 2'de detayları açıklanan Karar ağacı algoritmasıyla, paketlerin çevrim içi olarak sınıflandırılması için eğitim verileri kullanılarak, sınıflandırma kuralları elde edilmiştir. Gerçek zamanda çalışacak sistemin paket sınıflarını belirleyebilmesinde kullanılacak saldırı sınıfına ait bu kurallar aşağıda verilmiştir.

1. Eğer(protocol <2 & K.Port<=2 & H.IP<3 & K.IP<3 & uzunluk<2 & H.IP>=2) ise Sınıf=Saldırı.
2. Eğer(protocol<2 & K.Port<=2 & H.IP>2) ise Sınıf=Saldırı.
3. Eğer(protocol<2 & K.Port>=3 & H.IP<3 &uzunluk>=2) ise Sınıf=Saldırı.
4. Eğer(protocol>=2 & H.Port>=2 & uzunluk<3 & H.Port<3 & H.IP>=3) ise Sınıf=Saldırı.
5. Eğer(protocol>=2 & H.Port>=2 & uzunluk<3 & H.Port>=3 & K.IP<2) ise Sınıf=Saldırı.
6. Eğer(protocol>=2&H.Port>=2&uzunluk<3& H.Port>=3 & K.IP>=2 & protocol<3) ise Sınıf=Saldırı.
7. Eğer(protocol>=2 & H.Port>=2 & uzunluk<3 & H.Port>=3 & H.IP<2 & K.IP>=2 & protocol>=2 & HMAC>=2 & KMAC<2 & uzunluk<2) ise Sınıf=Saldırı.
8. Eğer(protocol>=2 & H.Port>=2 & uzunluk>=3 & protocol<3) ise Sınıf=Saldırı.
9. Eğer(protocol>=2 & H.Port>=2 & uzunluk>=3 & protocol>=3 & H.IP<3 & KMAC<2 & (K.IP= =2 | HMAC>=2)) ise Sınıf=Saldırı.

10. Eğer(protocol>=2&H.Port>=2&uzunluk>=3&protocol>=3&H.IP<3&KMAC>=2) ise Sınıf=Saldırı.

Tablo 7.4' teki eğitim verilerinin sınıflandırılması sonucunda karışıklık matrisi Tablo 7.10 ve sınıflandırmanın başarımı Tablo 7.11'deki gibi elde edilmiştir.

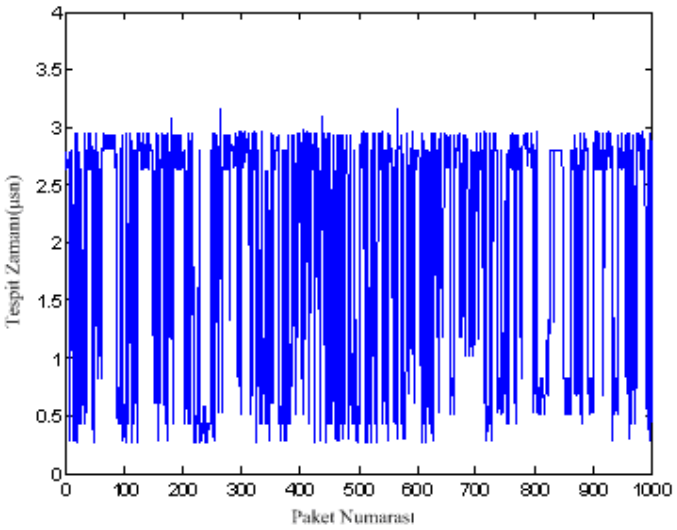
Tablo 7.10 Karar Ağacı için karışıklık matrisi

		Tahmin	
		Negatif	Pozitif
Gerçek	Negatif	0	17
	Pozitif	3	34

Tablo 7.11 Karar Ağacı başarımı

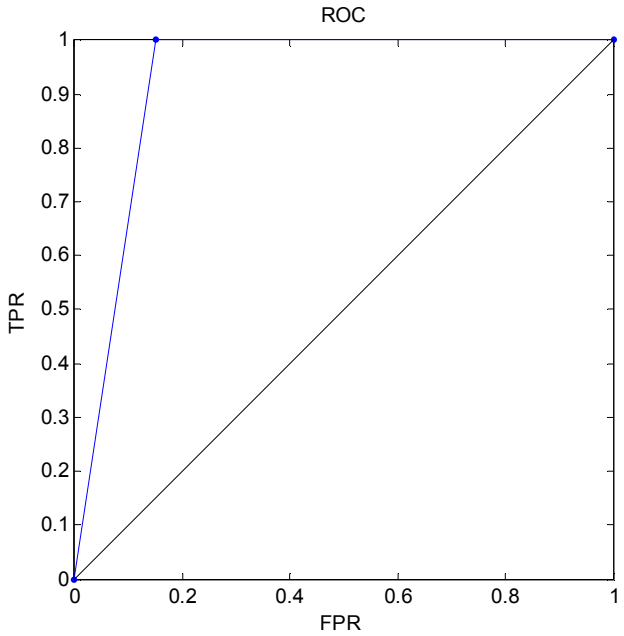
Doğruluk	TPR	FPR	TNR	FNR	Kesinlik
$\frac{a+d}{a+b+c+d}$	$\frac{d}{c+d}$	$\frac{b}{a+b}$	$\frac{a}{a+b}$	$\frac{c}{c+d}$	$\frac{d}{b+d}$
0,629	0,919	1	0	0,081	0,666

Sistem ağdan akan 1000 paket ile teste tabi tutularak paketlerin sınıflandırılma zamanı ve paketlerin sınıfları belirlenmiştir. Bu paketlerden 721 adedi normal 279 adedi ise saldırı olarak tespit edilmiştir. Karar Ağacı ile gerçekleştirilen sınıflandırıcının performansını ölçmek için Tablo 7.11' e ilaveten ROC eğrisi elde edilmiştir. Şekil 7.13 paketlerin sınıflandırılma zamanını Şekil 7.14 ise ROC eğrisinin değişimini göstermektedir.

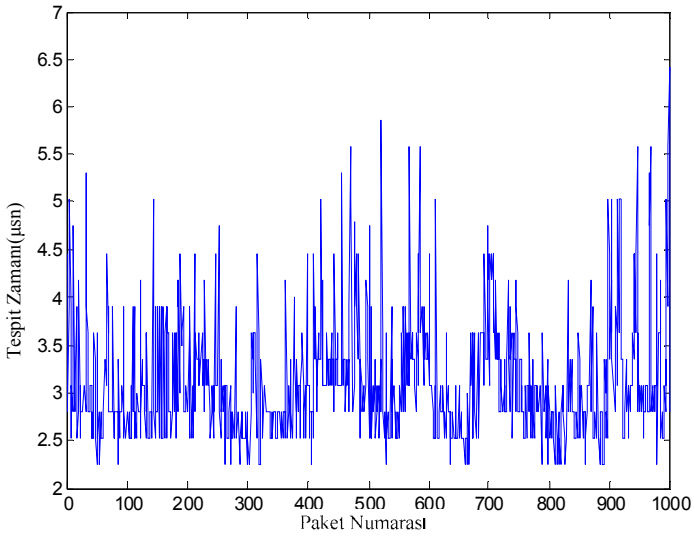


Şekil 7.13 Karar Ağacı için gerçek zamanda paket sınıfının tespit zaman değişimi

Ayrıca Java ortamında çevrim dışı olarak paketler teste tabi tutulmuş ve Şekil 7.15'teki paketlerin tespit zamanının değişimi elde edilmiştir.



Şekil 7.14 Karar ağacı için ROC eğrisi

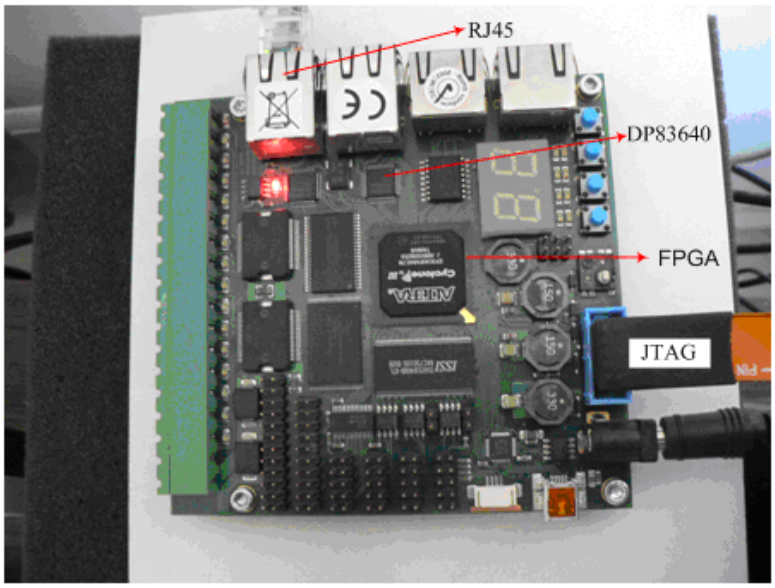


Şekil 7.15 Karar Ağacı için JAVA ortamında paketlerin tespit zaman değişimi

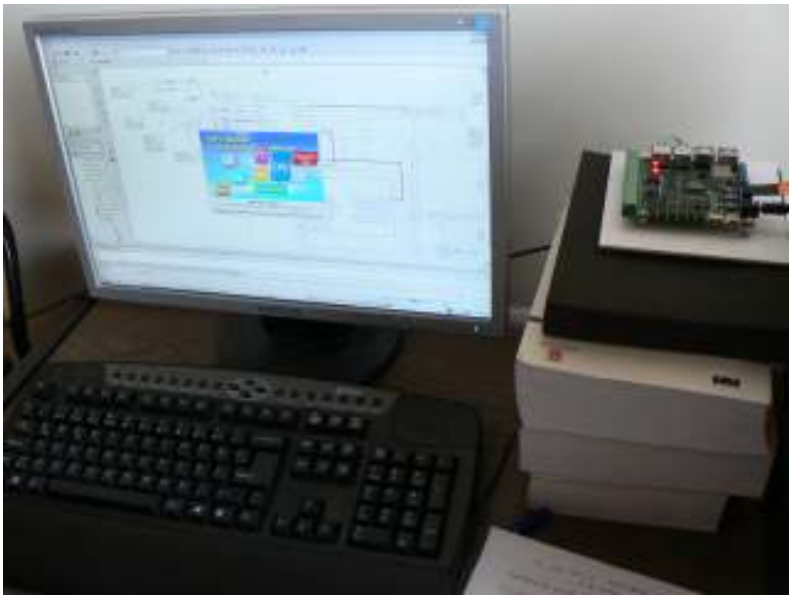
Her üç sınıflandırma ile elde edilen sonuçlara göre, paketlerin sınıflandırma başarımları birbirlerine çok yakındır. Eğitim verileri dikkate alınarak YSA sınıflandırıcının başarımı en iyi olmasına rağmen, (sistem paketleri sınıflandırırken birçok aritmetik

işlemler yaparak paketin sınıfına karar verdiğinden dolayı), paketlerin tespit zamanı diğer yöntemlere göre çok daha kötüdür. Naive Bayesian sınıflandırmada da paketlerin sınıfının belirlenmesinde aritmetik işlemler kullanıldığından dolayı paketlerin sınıflarının tespit zamanı Karar Ağacına göre yüksektir. Karar Ağaçları ile sınıflandırma yönteminde paket sınıflarının tespit süreleri diğer yöntemlere göre çok daha iyidir.

STS sisteminin gerçekleştirildiği DBC3C40 kartı ve çalışma ortamının fotoğrafları Şekil 7.16'da görülmektedir. DBC3C40 kartına ait özellikler Ek-5'te verilmiştir.



(a)



(b)

Şekil 7.16 a) DBC3C40 kartı b) Çalışma ortamının genel görünümü

8. SONUÇLAR

STS'lerin güvenilir ve hızlı bir şekilde paket sınıflandırması yapabilmesi için hem sınıflandırma algoritmasının doğru seçilmesi ve algoritmanın hızlı bir şekilde çalışabilmesini sağlayacak yazılımın oluşturulması hem de ilgili donanımsal devrenin gerçek zamanlı çalışmayı desteklemesi gerekir.

Bu tez çalışmasında çevrim içi çalışabilen ağ tabanlı bir STS'nin tasarlanması ve gerçekleştirilmesi yapılmıştır. Karar ağacı, YSA, Naive Bayesian algoritmalarına göre paket sınıflandırması yapabilen bu sistem FPGA ortamında donanımsal olarak gerçekleştirilip uygun bir şekilde programlanmıştır.

Tezde ilk olarak, kullanılan karar ağacı sınıflandırma algoritması ile paket özelliklerinin 23, 5, 2 alt gruplamalara ayrılarak paket sınıflandırması ayrı ayrı yapılmıştır. Klasik KDDCUP 99 veri setinin eğitim ve test verileri kullanılarak elde edilen sonuçlara göre sınıflama süresinin gruplama sayısının azaltılması ile küçülmesine rağmen, doğru sınıflandırma sayısında önemli bir azalma olmadığı tespit edilmiştir.

Daha sonra yukarıda bahsedilen her üç algoritma için yazılan simülasyon programları, güncel paketlerden elde edilen paket özetleri ile oluşturulan eğitim ve test verileri kullanılarak çalıştırılmıştır.

Ayrıca aynı algoritmaların çevrim içi sistemde çalışabilmesi için FPGA platformunda programlanabilir donanımsal bir STS oluşturulmuştur. Gerçekleştirilen bu donanımsal sistemde algoritmaların üzerinde çalışacağı bir soft işlemeci, hafıza modülü ve paketlerin ağdan alınıp gönderilebilmesi için yazılımsal olarak oluşturulabilen donanımsal bir 10/100 Ethernet MAC Core modülü mevcuttur. Bu donanımsal devre konfigüre edildikten sonra, yukarıda bahsedilen algoritmaların çevrim içi çalıştırılmasını sağlayan programlar yazılmış olup soft işlemcinin program hafızasına yüklenmiştir.

Karar ağacı, Naive Bayesian ve YSA sınıflandırma algoritmaları çevrim dışı ve çevrim içi çalıştırılarak aşağıda tartışılan sonuçlar elde edilmiştir.

YSA algoritmasıyla elde edilen sonuçlarda diğer iki algoritmaya göre daha yüksek doğruluk ve daha düşük yanlış alarm oranları elde edilmiştir. Ancak saldırıları tespit zamanı diğer iki algoritmaya göre daha yüksektir. Bu sonuç YSA sisteminde kullanılan çarpma işlemlerinden kaynaklanmaktadır.

Naive Bayesian algoritması ile elde edilen sonuçlarda doğru tespit oranı diğer iki algoritmaya göre daha düşüktür ancak yanlış alarm oranı ise daha yüksektir. Saldırı sınıfı tespit zamanı YSA algoritmasına göre daha düşük olmasına rağmen Karar ağacı algoritmasına göre daha yüksektir.

Karar ağacı algoritmasıyla elde edilen sonuçlara göre, doğru tespit oranı YSA algoritmasına göre düşük, Naive Bayesian algoritmasına göre ise yüksektir. Ancak yanlış alarm oranı diğer iki algoritmaya göre en düşüktür. Tespit zamanının ise diğer iki algoritmaya göre düşük olmasının nedeni sistemde eğer-ise kurallarının olmasıdır.

Karar ağacı algoritması normal ve saldırı paketlerini farklı zaman sürelerinde tespit ederken diğer iki algoritma yaklaşık olarak aynı zaman süresi içinde hem saldırı hem de normal paketlerin sınıflarını belirlemişlerdir.

Her üç sistemin başarımları için ROC eğrileri baz alınırsa en uygun algoritmanın YSA algoritması olduğu görülür. Ancak saldırıların tespit ve engelleme zamanı dikkate alındığında en uygun algoritmanın Karar Ağacı algoritması olduğu görülecektir.

Kullanıcının defalarca programlayarak donanımsal devre oluşturabileceği FPGA platformu bu tezdeki gerçek zamanlı STS'nin geliştirilmesi amacı için çok uygun bir platform oluşturmaktadır. Aşağıda bu platformun kullanılmasıyla elde edilen kazanımlar sıralanmıştır.

1. STS'nin gerçekleştirilmesi sürecinde FPGA ortamı esnek bir yapı sunmuştur. Sistem her türlü yazılımsal ve donanımsal değişikliğin kolaylıkla yapılabileceği şekilde oluşturulmuştur. FPGA ortamında donanımsal olarak sistem oluşturulması için VHDL diliyle sistemi tanımlamak yeterli olacaktır.
2. Gerçek zamanlı STS, FPGA ortamında bir SOPC (Tek yonga programlanabilir sistem) şeklinde oluşturulmuştur. Yani FPGA ortamında oluşturulan NIOS II işlemcisi ile STS sistemi denetlenir haldedir. Dolayısıyla oluşturulan bu soft işlemcinin ayrıca yazılımsal olarak ta programlanabilmesi, bu tür sistemlerin geliştirilmesi aşamasında çok önemli bir esneklik sağlayacaktır. Tez çalışması sürecinde oluşturulan böyle bir platform ile STS sistemleri için değişik yöntemlerin uygulanması ve test edilmesi rahatlıkla yapılabilecektir. Tez çalışmasından elde edilen önemli kazanımlardan biriside budur. .
3. Herhangi bir verinin FPGA ortamında yani donanımsal olarak işlenebilmesi için en alt seviyede kodlanmış ve elektriksel işaretler şeklinde oluşturulmuş olması gerekir. Ağdan gelen çerçevelerinde donanımsal olarak işlenebilmesi için, OSI modelinin

veri bağı katmanında tarif edilen tüm işlemleri başarabilen donanımsal bir modül üzerinden FPGA ortamına alınması gereklidir. Bu tez çalışması sonunda elde edilen önemli kazanımlardan biriside VHDL ortamında yazılmış böyle bir modülün FPGA'ya gömülüp donanımın oluşturulması ve konfigüre edilmesi olduğu söylenebilir. Çünkü bu durumda Ethernet ağı ile FPGA, veri bağı katmanı seviyesinde gerçek zamanda iletişim kurabilmektedir. Ağdan alınabilecek veri, ses veya video paketlerinin gerçek zamanda FPGA içerisinde işlenebilmesi için önemli bir platform da bu suretle elde edilmiştir.

Oluşturulan bu platformda, her üç algoritmada gerçek zamanda çalıştırılmıştır. Her biri için paketlerin sınıflandırma başarımları, tespit zamanları ayrı ayrı elde edilmiştir.

Çevrim dışı ve çevrim içi çalışmadan elde edilen sonuçlar karakteristik olarak birbirlerine benzemekle beraber, eğri karakterlerinin tamamen uyumlu çıkmamasının (paket tespit sürelerinin eşit olması değil, genel karakteristiklerinin uyumlu olması beklenir) nedeni bilgisayar işlemcisinin iş yükünün değişkenliği ile yorumlanabilir.

Yapılan bu teze paralel olarak yürütülen ve sonuçlandırılan 108E166 sayılı TUBİTAK projesinin, teze ve özellikle bu platformun oluşturulmasına büyük katkısı olmuştur[61]. Bu platformu kullanarak yapılacak çalışmalardan elde edilen sonuçların yayınlanacağı bildiri ve makalelerde ilgili TUBİTAK proje desteği belirtilecektir.

KAYNAKLAR

[1] Stallings, W., 2003, Network Security, ed: Horton, M.J.,Prentice Hall, New Jersey,

[2] Bidgoli, B.M., Analoi, M., Rezvani, M.H., Shahhoseini H.S., 2008. Performance Evaluation of Decision Tree For Intrusion Detection Using Reduce Feature Spaces, Trends in Intelligent System and Computer Engineering, LNCS 1876, **(6),** Heidelberg.

[3] Ohta, S., Kurebayashi, R., Kobayashi, K., 2008. Minimizing False Positives of a Decision Tree Classifier for Intrusion Detection on the Internet, Journal of Network and System Management, **16 (4),** 399-419.

[4] Owais, S., Snasel, V., Kromer, P., Abraham, A., 2008. Survey: Using Genetic Algorithm Approach in Intrusion Detection System Techniques, 7th Computer Information System and Industrial Management Applications, 300-307.

[5] Amor, N.B., Benferhat, S., Elouedi, Z., 2004. Naïve Bayes vs Decision Trees in Intrusion Detection System, Proceedings of The ACM Symposium on Applied Computing, Nicosia, 420-424.

[6] Shun, J., Malki, A.H., 2008. Intrusion Detecting System Using Neural Networks, Fourth International Conference on Natural Computation, **(5),** 242-246.

[7] Jahankhani, H., Hessami A.G., Hsu, F., Beqiri, E., 2009. Neural Networks for Intrusion Detection System, Communications in Computer and Information Science, **(45),** 156-165.

[8] Pasham, V., Trimberger S., 2001. High-Speed Des and Triple Des Encryptor/ Decryptor. Xilinx Application Note.

[9] Huang, W.J., Saxena, N., McCluskey, E.J., 2000. A Reliable LZ Data Compressor on Reconfigurable Coprocessor. IEEE Symposium on Field Programmable Custom Computing Machines. Napa USA.

[10] Hassan, A.A., Elnakib, A., Abo-Elsoud, M., 2008. FPGA-Based Neuro-Architecture Intrusion Detection System. International Conference Computer Engineering& System, 268-273.

[11] Liu, Y., Xu, D., Liu, D., L.S 2009. A Fast and Configurable Pattern Matching Hardware Architecture for Intrusion Detection. Second International Workshop on Knowledge Discovery and Data Mining, 614-618.

[12] Song, H., Lockwood J.W., 2005. Efficient Packet Classification for Network Intrusion Detection Using FPGA. Proceedings of the ACM/SIGDA 13th International Symposium on Field-Programmable Gate Arrays, Monterey USA, 238-245.

[13] Park, S-K., Oh, J-T., Nam, T-Y., Jang, J-S., 2007. High Speed Concurrent Attack Responding System Using FPGA Supporting IPv4/IPv6. ICACT 2007, 1252-1257.

[14] Baba-Ali, A.R., 2009. A High Speed Network Intrusion Detection System Based on FPGA Circuits. IJCSNS International Journal of Computer Science and Network Security, **9(11),** 301-304.

[15] Kang, D-H., Kim, B-K., Oh, J-T., Nam, T-Y., Jang, J-S., 2006. FPGA Based Intrusion Detection System Against Unknow and Know Attacks. Lecture in Computer Science, **(4048),** 801-806.

[16] Clark, C.R., Ulmer, C.D., Schimmel, D.E., 2006. An FPGA-Based Network Intrusion Detection System on-chip Network Interface. International Conference Computer Engineering & System, 268-273.

[17] Lee, J., Hwang, S.H., Park, N., Lee, S-W., Jun, S., Kim, Y.S., 2007. A High Performance NIDS Using FPGA-Based Regular Expression Matching. Symposium on Applied Computing, Seoul, Korea, 1187-1191.

[18] Weaver N., Paxson, V., Gonzalez, M.J., 2007. The Shunt an FPGA-Based Accelerator for Network Intrusion Prevention, FPGA2007.

[19] Jorge, B., Carlos T. C., 2007. A Low-Cost Embedded IDS to Monitor and Prevent MAN-in-the-Middle Attacks on Wired LAN Environments. International Conference on Emerging Security Information System and Technologies, 122-127.

[20] Das, A., Nuguyen, D., Zambreno, J., Memik, G., Choudhary, A., 2008. An FPGA-Based Network Intrusion Detection Architecture. IEEE Transaction on Information Forensics and Security, **3, 1,** 118-132.

[21] Narayan, R., Hanbo, D., Memik, G., Choudhary, A., Zambreno, J., 2007. An FPGA Implementation Decision Tree Classification. Proceedings of the Conference on Design Automation and test in Europa Nice, France, 189-194.

[22] Katashita, T., Yamaguchi, Y., Maeda, A., Toda, K., 2007. FPGA- Based Intrusion Detection System for 10 Gigabit Ethernet, IEICE Transaction Information & System. **E90-D, 12** 1923-1931.

[23] Sokolova, M., Jopkowicz, N., Szpakowicz, S., 2006. Beyond Accuracy, F-Score, ROC:A Family of Discriminant Measures for Performance. Australian Conference on Artifical Intelligence. 1015-1021.

[24] Scarfone, K., Mell, P., 2007, Guide to Intrusion Detection and Prevention System (IDPS), National Institute of Standards and Technology.

[25] Valeur, F., 2006. Real-Time Intrusion Detection Alert Correlation, Phd University of California Santa Barbara. USA.

[26] Bishop, M., 2003. Computer Security, Addison-Wesley, Boston USA.

[27] Monnila, H., 2002. Local and Global Method in Data Mining: Basic Techniques and Open Problems, International Colloquium and Automata, Languages and Programing Malaga, Spain, Springer Verlag.

[28] Mahoney M.V, Chan, P.K., 2003. Learnig rules for Anomaly Detection of Hostile Network Traffic IEEE International Conference on Data Mining, Melbourne, 601-604.

[29] Lee, W., Stolfo S.J., Mak, K.W., 2000. Adaptive Intrusion Detection: A data Mining Approach, Artifical Intelligence Review, 14(6), 533-567.

[30] Bass, T., 2000. Intrusion detection System and Multisensor Data Fusion, Communication of the ACM, 43(4), 99-105.

[31] Li, T., Pan, W., 2005. Intrusion Detection System Based an New Assocation Rule Mining Model, IEEE International Conference on Granular Computing, (2), 512-515.

[32] Leu, F-Y., Yang, W-J., 2005. Intrusion Detection with CUSUM for TCP-Based DDOS, Lecture Notes in Computer Science, **3823**, 1255-1264.

[33] Siris, A.V., Papagalou, F., 2006. Application of Anomaly Detection Algorithms For Detecting SYN Flooding Attacks, Computer Communications, 29, 1433-1442.

[34] Oshita, Y., Ata, S., Murata, M., 2004. Detecting Distributed Denial of Service Attacks by Analyzing TCP SYN Packets Statistically. GLOBECOMM2004. 2043-2049.

[35] Wang, H., Zang, D., Shin K.G., 2004. Change Point Monitoring For The Detection of DoS Attacks. IEEE Transaction on Dependable and Secure Computing **1(4).** 193-208.

[36] Thottan, M., Ji, C., 2003. Anomaly Detection in IP Networks, Proceeding of IEEE Transaction on Signal Processing (51), 2191-2204.

[37] Tuncer, T., Tatar, Y., 2007. TCP SYN Saldırılarının Bulanık Mantık Kullanılarak Tespiti, Information Security & Cyrptology International Participation, Ankara, 254-257.

[38] Tuncer, T., Tatar, Y., 2008. Detection SYN Flooding Attacks Using Fuzzy Logic, International Conference on Information Security on Assurance, Busan Korea, 321-325.

[39] Depren, O., Topallar, M., Anarım, E., Cılız, M.K., 2005. Intelligent Intrusion Detection System (IDS) For Anomaly And Misuse Detection in Computer Networks, Expert Systems With Applications, **(29),** 713-722.

[40] Yang, W., Fang, B-X., Liu, B., Zhang, H-L., 2004. Intrusion Detection System For High-Speed Network, Computer Communications, (27), 1288-1294.

[41] Boughaci, D., Ider, K., Yahiaoui, S., 2007. Design and Implementation of a Misused Intrusion Detection System Using Autonomous and Mobile Agents, Proceedings of the 2007 Euro American Conference on Telematicsand Information System, Faro, Portugal.

[42] Kim, T-K., Lee, D-Y., Chung, M., 2002. Mobile Agent-Based Misuse Intrusion Detection Rule Propagation Model for Distributed System, Lecture Notes in Computer Science**, (2510),** 842-849.

[43] Eduardo, M-R., Amparo, A-B., Belen, B.d.R., Jesus, L.P., 2007. A Misuse Derection Agent for Intrusion Detection in a Multi-Agent Architecture, Lecture Notes in Computer Science, **(4496),** 466-475.

[44] Bon, K.S., 2005. Signature-Based Approach for Intrusion Detection, Lecture Notes in Computer Science, **(3587),** 526-536.

[45] Hwang, K., Cai, M., Chen Y., Qin, M., 2007. Hybrid Intrusion Detection With Weighted Signature Generation Over Anomalous Internet Episodes, IEEE Transaction on Dependable and Secure Computing, **(4), 1,** 41-53.

[46] Botha, M., Solms, R., Perry, K., Loubser, E., Yamoyany, G., 2002. The Utilization of Artificial Intelligence in a Hybrid Intrusion Detection System, Annual Conference of the South African Institute of Computer Scientific and Information Technologists, Somerset West, 149-155.

[47] Gomez J., Gil, C., Padilla, N., Banos, R., Jimenez, C., 2009. Design of a Snort-Based Hybrid Intrusion Detection System, (5518), 515-522.

[48] Ali A.M., Zalim, H. A., Ceylan K.G., 2009. A Hybrid Intrusion Detection system Design for Computer Network Security, Computer & Electrical Engineering **35 (3)**, 517-526.

[49] Quinlan, J. R., 1986. Induction of Decision Trees, Machine Learning-I (1), 1, Springer Netherlands, 81-106.

[50] Silahtaroğlu, G., 2008. Kavram ve Algoritmalarıyla Temel Veri Madenciliği, Papatya Yayıncılık, İstanbul.

[51] Elseiver, 2008, FPGAs:Instant Access, The new Instant Access Series.

[52] Altera, 2010 Cyclone III Device Handbook, Volume 1.

[53] Perry, D.L., 2002. VHDL: Programming by Example, McGraw-Hill, New York.

[54] Wilder, F., 1998, A Guide To The TCP/IP Protocol Suite, Artech House, Boston.

[55] Morethanip, 2006. 10/100 Mbps Ethernet MAC Core for Avalon Reference Guide, Germany

[56] Altera Corporation, 2007. Nios II Hardware Development Tutorial.

[57] Tuncer, T., Tatar, Y., 2009. Programmable Embedded System Design: A Study of Sniffer, 3th International Conference on Application of Information Communication Technologies, Baku.

[58] Morethanip, 2008. 10/100 Ethernet IP MAC CORE for Avalon User's Guide Germany.

[59] Khan, S., 2005. Basic Input/Output Functions Morethanip, Germany.

[60] Tuncer T., Tatar, Y., 2009. Karar Ağacı Kullanılarak Saldırı Tespit Sistemlerinin Performans Değerlendirmesi IV. İletişim Teknolojileri Ulusal Sempozyumu Adana, 77-82.

[61] Tatar, Y., Tuncer, T., 2009. Bilgisayar Ağlarında Bir Saldırı Tespit Ve Engelleme Sisteminin (STES) Gerçekleştirilmesi (TUBITAK EEEAG 108E166 numaralı Proje Sonuç Raporu), Tübitak, Ankara.

EK.1 KDD CUP 99 Veri Seti Paket Özellikleri ve Saldırı Türleri

Özellik Adı	Tanım
Duration	Bağlantının uzunluğu
Protocol_type	Protokol tipi (tcp, udp gibi)
Service	Hedef ağ servisi (http, telnet gibi)
Src_bytes	Kaynaktan hedefe verilerin bayt sayısı
Dst_bytes	Hedeften kaynağa verilerin bayt sayısı
Flag	Bağlantıların normal veya hatalı durumları
Land	Bağlantı aynı host/port'a ait ise 1 değilse 0
Wrong_fragment	Yanlış fragmentlerin sayısı
Urgent	Acil paketlerin sayısı
Hot	"hot" göstericilerinin sayısı
Num_failed_logins	Yanlış loginlerin sayısı
Logged_in	Başarılı login sayısı, 1 değilse 0
Num_compromised	Anlaşma şartlarının sayısı
Root_shell	Root shell sayısı elde edilirse 1 değilse 0
Su_attempted	Su_root komutları girişimi varsa 1 değilse 0
Num_root	Root'a erişim sayısı
Num_file_creations	Dosya oluşturma içerikleri sayısı
Num_shells	Shell prompt sayısı
Num_access_files	Erişim kontrol dosyalarındaki işlemlerin sayısı
Num_outbound_cmds	ftp oturumlarında outbound komutlarının sayısı
İs_hot_login	Loginler hot listesine ait ise 1 değilse 0
İs_guest_login	Loginler misafir listesine ait ise 1 değilse 0
Count	Geçmiş 2 saniye içinde aynı host'a bağlantıların sayısı
Srt_count	Geçmiş 2 saniye içinde aynı servise bağlantıların sayısı
Serror_rate	SYN hatalarına sahip bağlantıların yüzdesi
Srv_error_rate	SYN hatalarına sahip bağlantıların yüzdesi
Rerror_rate	REJ hatalarına sahip bağlantıların yüzdesi
Srv_rerror_rate	REJ hatalarına sahip bağlantıların yüzdesi
Same_srv_rate	Aynı servise bağlantıların yüzdesi
Diff_srv_rate	Farklı servise bağlantıların yüzdesi
Srv_diff_host_rate	Farklı hostlara bağlantıların yüzdesi
Dst_hst_count	Aynı hedefe sahip bağlantıların sayısı
Dst_hst_srv_count	Aynı servisi kullanan aynı hotsa sahip hedef bağlantıların sayısı
Dst_hst_same_srv_rate	Aynı servisi kullanan aynı hotsa sahip hedef bağlantıların sayısı
Dst_hst_diff_srv_rate	Şimdiki hostta farklı servislerin yüzdesi
Dst_hst_sm_src_prt_rate	Aynı kaynak porta sahip şimdiki hosta bağlantıların yüzdesi
Dst_hst_srv_diff_hst_rat	Farklı hostlardan gelen aynı servise bağlantıların yüzdesi
Dst_hst_serror_rate	S0 hatasına sahip şimdiki hosta bağlantıların yüzdesi
Dst_hst_srv_serror_rate	S0 hatasına sahip özel servis ve şimdiki hosta bağlantıların yüzdesi
Dst_hst_rerror	RST hatasına sahip şimdiki hotsa bağlantıların yüzdesi
Dst_hst_srv_rerror rate	RST hatasına sahip özel servis ve şimdiki hotsa bağlantıların yüzdesi

Saldırı Tipi	Sınıflar	Tanım
DoS	Back Land Neptune Pod Smurf Teardrop	Bu saldırılar genel olarak TCP/IP protokol yapısındaki açıklardan faydalanılarak bir sunucuya birden çok bağlantı isteği göndererek yasal kullanıcıların hizmet almasını engellemeye yöneliktir. Protokol hatalarına dayalı saldırılara ölümüne ping saldırısı (bir tek büyük boyutlu ICMP eko paketinin gönderilmesi) örnek verilebilir. Yine bir başka saldırı TCP SYN Flooding saldırısı bu tür saldırılara örnektir.
Probe	Ipsweep Nmap Portsweep Satan	Bu tür saldırılar bir sunucunun yada herhangi bir makinanın geçerli IP adreslerini, aktif portlarını veya işletim sistemini öğrenmek için geliştirilmiştir.
R2L	Ftp_write Guess_passwd Imap Multihop Phf Spy Warezclient Warezmaster	Yönetici Hesabı ile Yerel Oturum Açma saldırıları, kullanıcı haklarına sahip olunmadığı durumda misafir yada başka bir kullanıcı olarak sisteme izinsiz erişim yapılarak gerçekleştirilir.
U2R	Buffer_Overflow Loadmodule Perl Rootkit	Kullanıcı Hesabının Yönetici Hesabına Yükseltilmesi ile gerçekleştirilecek saldırılarda sisteme girme izni olana fakat yönetici olmayan bir kullanıcının yönetici izni gerektirecek işler yapmaya kalkışmasıdır.
	Normal	

EK.2 Paketlerin gönderilmesi ve alınmasını ve sınıflandırılmasını sağlayan YSA programı

```
#include <stdlib.h>
#include "system.h"
#include <mtip_avalon_10_100_1000_mac_regs.h>
#include <altera_avalon_dma_regs.h>
//Birinci katmanin girisleri
double layer1in[layer1count];
//Birinci katmanin cikislari
double layer1out[layer1count], layer2in[layer2count], layer2out[layer2count];
double layer3in, layer3out;
//Birinci katmanin agirliklari
double w1[layer1count][layer2count];
//İkinci katmanin agirliklari
double w2[layer2count], layer2bias[layer2count], layer3bias, bias2w[layer2count], bias3w;
//Birinci katmandaki noronun işlevi
double layer1func(double inValue) {
   return (double)2/(1+pow(2.718,-2*inValue))-1;
}
double layer2func(double inValue){
   return (double)2/(1+pow(2.718,-2*inValue))-1;
}
double layer3func(double inValue){
   return inValue;
}
void computeLayer1out(){
    for (int i = 0;i<layer1count;i++)     {
       layer1out[i] = layer1func(layer1in[i]);
    }
}
void computeLayer2in(){
    double sumOfInputs;
    for (int i = 0; i<layer2count;i++){
       sumOfInputs = 0;
       for (int j=0;j<layer1count;j++)        {
           sumOfInputs += layer1out[i]*w1[i][j] + bias2w[i];
       }
       layer2in[i] = sumOfInputs;}}
void computeLayer2out(){
    for (int i=0;i<layer2count;i++){
       layer2out[i] = layer2func(layer2in[i]);
    }}
void computeLayer3in(){
    double sumOfInputs;
    sumOfInputs = 0;
    for (int j=0;j<layer2count;j++){
       sumOfInputs += layer2out[i]*w2[i];
    }
```

```
    layer3in = sumOfInputs +bias3w;
}
void computeLayer3out(){
    layer3out = layer3func(layer3in);
}
double ysaHesapla(){
   computeLayer1out();
   computeLayer2in();
   computeLayer2out();
   computeLayer3in();
   computeLayer3out();
   return layer3out;
}
void initNN(){
   //1. katman ile 2. katman arasındaki ağırlıkları ve bias değerlerini giriniz.
}
//Başlatma Fonksiyonu
void mtip_mac_initTransInfo2( mtip_mac_trans_info *mi, int  mac_base, int  dma_base,
      int  dma_rx_base, int  rx_fifo_base, int  tx_fifo_base, int  cfgflags, int  *rxbuf ) {
      mi->mac    = (np_mtip_mac*)mac_base;
      IOWR(&mi->mac->TX_CMD_STAT,0, mmac_tcs_SHIFT16_mask);
      if( IORD(&mi->mac->TX_CMD_STAT,0) == mmac_tcs_SHIFT16_mask ) {
         mtip_hasShift16=1;
      } else {
         mtip_hasShift16=0;
      }      }
// Paket gönderme fonksiyonu
//DMA kullanılarak hafızadaki paket Transmit FIFO' ya kopyalanır.
int  mtip_mac_sTxWrite( mtip_mac_trans_info *mi,      // MAC ve DMA Adresleri
                        int *src,     // Paketin hafıza alanındaki adresi
                        int length // Paketin uzunluğu) {
}
//Paket alma Fonksiyonu
// DMA kullanılarak, Receive FIFO' daki paket hafızaya kopyalanır.
int  mtip_mac_sRxRead( mtip_mac_trans_info *mi,      // MAC ve DMA adresleri
         int *dest,    // paketin kopyalanacağı adres
         int length// Paketin uzunluğu) {
}
mtip_mac_trans_info mi;
char myAddr[6] ={0xFB,0x43,0x36,0xAB,0xCF,0x15}; //Gömülü sistemin MAC adresi
char destAddr[6] ={0x00,0x1F,0xC6,0xC7,0xD2,0xEC}; //Paketin gönderileceği hedefin
MAC adresi
int waitRxEmpty(mtip_mac_trans_info *mi ) {
      int timeout, lenrx, status, rxcnt, int i;
      do {
            if( mtip_mac_isFrameAvail( mi-> mac ) ) {
                  status = mi->mac->RX_CMD_STAT;
                  lenrx = mtip_mac_getFrameLength( mi->mac );
                  mtip_mac_sRxRead( mi, (int*)g_rxbuf, lenrx );
```

```
                if( status & mmac_rcs_ERROR_mask ) printf(". ERR flag");
                timeout = 0;
                rxcnt++;
            } else {
                timeout++;
            }
        } while( timeout < 1000000 );
        return rxcnt;
}
```
//10/100 Ethernet MAC Core Registerleri
```
void print_mac_status(np_mtip_mac *pmac) {
printf("AVL_STATUS: %08x, RX_CMD_STAT: %08x, TX_CMD_STAT: %08x,
MAC_0=%08x,MAC_1=%08x, COMMAND_CONFIG: %08x\n",
                pmac->AVL_STATUS, pmac->RX_CMD_STAT, pmac->TX_CMD_STAT,
                pmac->MAC_0, pmac->MAC_1, pmac->COMMAND_CONFIG);}
```
//Gönderilecek paketin hedef adres alanına hedef MAC adresinin yazılması
```
void setDestAddr( char *buf, char * addr) {
   for(int i=0; i < 6; i++ ) {
     ((char*)buf)[i] = addr[i];}}
```
//Len uzunlugunda çerçeve olusturma
```
void createNormalFrame( char *buf, char *addr, int len) {
    for( i = 0; i < len; i++ ) {
     ((char*)buf)[i] = i;
   }
   l = len-14;
   buf[12] = (l >>8) & 0xff;
   buf[13] = l & 0xff;
   setDestAddr( buf, addr );
}
#define NTLPHY_ID 0x20005c70    /* National DP83865 */
#define MVLPHY_ID 0x01410cc0    /* Marvell 88E1111 */
#define NTL848PHY_ID 0x20005c90 /* National 83848, 10/100 */
#define TDKPHY_ID    0x0300e540   /* TDK 78Q2120 10/100 */
int main(){
alt_u32 ms_per_tick;
int tm,len,is1000,dat,ds, phy_id, allOk, mac_base, dma_base, rx_fifo_base, tx_fifo_base;
int *rxbuf;
char *xmem;
```
//Alinacak paketlerin veya gönderilecek paketlerin saklanacağı adres alanları
```
        xmem    = (char*)((int)malloc(8192) );
       g_rxbuf = (char*)((int)malloc(8192) );
       xmem = (char*)alt_remap_uncached( xmem, 8192 );
       g_rxbuf = (char*)alt_remap_uncached( g_rxbuf, 8192 );
       rxbuf = (int*)malloc(1536);
```
//Baslatma fonksiyonu ile donanımlar baslatılır.
```
mtip_mac_initTransInfo2( &mi, MAC_CONTROL_PORT_BASE,
DMA_BASE, 0,MAC_RXFIFO_BASE, MAC_TXFIFO_BASE,0, rxbuf );
isMVL =0;
       do {
```

```
          retry = 0;
          for(i=0; i < 32; i++) {
             mi.mac->MDIO_ADDR0 = i;
             dat = IORD(&mi.mac->mdio0.PHY_ID1,0);
             ds  = IORD(&mi.mac->mdio0.PHY_ID2,0);
// MDIO adreslerinin taranmasi
             if( dat != ds ) {
                if( i == 31 ) {
                   retry = 1;
          IOWR(&mi.mac->mdio0.CONTROL,0,0x9000); // MDIO adreslerinin yazılması
                   usleep(10000);
                }
                break;}}}
 while(retry);
// Fiziksel entegre tanınıyor
       phy_id = ((dat & 0xffff) << 16) | (ds & 0xfff0);
       IOWR(&mi.mac->mdio0.CONTROL,0,0x9000);
       usleep(10000);
       #if 1
//Fiziksel arabirim hızı belirleme
       dat = IORD(&mi.mac->mdio0.CONTROL,0) & 0x001f;
       #ifdef GIGABIT
       #else
       if( isMVL ) {
          IOWR(&mi.mac->mdio0.CONTROL,0,dat | 0xe100);
       }
       IOWR(&mi.mac->mdio0.CONTROL,0,dat | 0x6100);
       #endif
       #endif
       x=0;
       while( (IORD(&mi.mac->mdio0.STATUS,0) & PCS_ST_rx_sync) == 0 ){
          if( x++ > 100000) break;
       }
       printf("OK. x=%d, STATUS=%04x\n",x, IORD(&mi.mac->mdio0.STATUS,0));
       usleep(10000);
       dat = IORD(&mi.mac->mdio0.reg11,0);
       is1000 = ( dat==2 ) ? 1 : 0;
       if(     dat==0) printf(" 10Mbps");
       else if(dat==1) printf(" 100Mbps");
       else if(dat==2) printf(" 1000Mbps");
       printf("-----------------------------------\n");
       printf("MAC resetleniyor..");
    if( is1000 ) {
       mi.mac->COMMAND_CONFIG = mmac_cc_SW_RESET_mask |
mmac_cc_TX_ENA_mask | mmac_cc_RX_ENA_mask | mmac_cc_ETH_SPEED_mask;
    } else {
    }
    x=0;
    while(mi.mac->COMMAND_CONFIG & mmac_cc_SW_RESET_mask) {
```

```
      if( x++ > 1000 ) break;
   }
   dat = IORD(&mi.mac->COMMAND_CONFIG,0);
   if( (dat&0x03)  != 0 ) {
      printf("WARN: RX/TX not disabled after reset... missing PHY clock?
CMD_CONFIG=0x%08x\n", dat);
   } else {
      printf("OK, x=%d, CMD_CONFIG=0x%08x\n\n", x, dat);
   }
//10/100 Ethernet MAC Core registerleri konfigüre et
      i = waitRxEmpty( &mi );
      if(i)printf("RX FIFO  %d frame\n", i);
//Çerçeve oluşturmak için hafıza alanına veri yaz
      for(i=0; i < 512; i++ ) {
            xmem[i] = i;
           }
xx:
//Çerçeve gönderme ve alma fonksiyonları çalıştırılıyor
//Çerçeve alınmamış ise hafızadaki çerçeveyi gönder.
//Çerçeve alınmış ise hafızaya yaz ve hafızadaki çerçeveyi geri gönder
 tm=0;
      while( !(mtip_mac_isFrameAvail( mi.mac)) ) {
           if( ++tm == 1500000 ) {
                 setDestAddr( xmem, destAddr );
                 xmem[12] = 0x0a;
                 xmem[13] = 0x00;
                 mtip_mac_sTxWrite( &mi, (int*)xmem, 128 );}}
        do {
          printf("aliniyor... ---------- \n");
          len = mtip_mac_getFrameLength( mi.mac );
          printf("rx: Alinan çerçeve uzunlugu: %d\n", len);
          dat = mtip_mac_sRxRead( &mi, (int*)xmem, len );
//Alinan paketi ekrana yaz
                          for(i=0; i < len; i++ ) {
                 printf(" %02x", (int)(xmem[i])&0xff );  }
//YSA Algoritmasına göre paketin sınıfını belirle
dx=round(ysaHesapla());}
if(dx>0.5) {
      printf("Paket Normaldir.”);
   }
   else{
       printf("Paket Saldırıdır.”);
   }
//Alma ve gönderme işlemlerini tekrar et.
          goto xx;
        }
}
```

EK.3 Paketlerin gönderilmesini, alınmasını ve sınıflandırılmasını sağlayan Naive Bayesian programı

```
#include <stdlib.h>
#include "system.h"
#include <mtip_avalon_10_100_1000_mac_regs.h>
#include <altera_avalon_dma_regs.h>
olasilikhesapla(){
//Eğitim setindeki tüm niteliklerin sınıf için olasılıklarını hesapla
}
bayeshesapla(){
// Sınıf olasılıkları
sal=dmacs[dmacc-1]*smacs[smacc-1]*protocols[protocoll-1]*sips[sipp-1]*dips[dipp-
1]*sports[sportt-1]*dports[dportt-1]*lenghts[lenghtt-1];
nor=dmacn[dmacc-1]*smacn[smacc-1]*protocoln[protocoll-1]*sipn[sipp-1]*dipn[dipp-
1]*sportn[sportt-1]*dportn[dportt-1]*lenghtn[lenghtt-1];
 }
//Başlatma Fonksiyonu
void mtip_mac_initTransInfo2( mtip_mac_trans_info *mi, int  mac_base, int  dma_base,
      int  dma_rx_base, int  rx_fifo_base, int  tx_fifo_base, int  cfgflags, int  *rxbuf ) {
      mi->mac    = (np_mtip_mac*)mac_base;
      IOWR(&mi->mac->TX_CMD_STAT,0, mmac_tcs_SHIFT16_mask);
      if( IORD(&mi->mac->TX_CMD_STAT,0) == mmac_tcs_SHIFT16_mask ) {
         mtip_hasShift16=1;
      } else {
         mtip_hasShift16=0;
      }      }
// Paket gönderme fonksiyonu
//DMA kullanılarak hafızadaki paket Transmit FIFO' ya kopyalanır.
int  mtip_mac_sTxWrite( mtip_mac_trans_info *mi,      // MAC ve DMA Adresleri
                        int *src,     // Paketin hafıza alanındaki adresi
                        int length // Paketin uzunluğu) {
}
//Paket alma Fonksiyonu
// DMA kullanılarak, Receive FIFO' daki paket hafızaya kopyalanır.
int  mtip_mac_sRxRead( mtip_mac_trans_info *mi,      // MAC ve DMA adresleri
         int *dest,    // paketin kopyalanacağı adres
         int length// Paketin uzunluğu) {
}
mtip_mac_trans_info mi;
char myAddr[6] ={0xFB,0x43,0x36,0xAB,0xCF,0x15}; //Gömülü sistemin MAC adresi
char destAddr[6] ={0x00,0x1F,0xC6,0xC7,0xD2,0xEC}; //Paketin gönderileceği hedefin
MAC adresi
int waitRxEmpty(mtip_mac_trans_info *mi ) {
      int timeout, lenrx, status, rxcnt, int i;
      do {
            if( mtip_mac_isFrameAvail( mi-> mac ) ) {
                  status = mi->mac->RX_CMD_STAT;
                  lenrx = mtip_mac_getFrameLength( mi->mac );
```

```
                mtip_mac_sRxRead( mi, (int*)g_rxbuf, lenrx );
                if( status & mmac_rcs_ERROR_mask ) printf(". ERR flag");
                timeout = 0;
                rxcnt++;
            } else {
                timeout++;
            }
        } while( timeout < 1000000 );
        return rxcnt;
}
```

//10/100 Ethernet MAC Core Registerleri

```
void print_mac_status(np_mtip_mac *pmac) {
printf("AVL_STATUS: %08x, RX_CMD_STAT: %08x, TX_CMD_STAT: %08x,
MAC_0=%08x,MAC_1=%08x, COMMAND_CONFIG: %08x\n",
                pmac->AVL_STATUS, pmac->RX_CMD_STAT, pmac->TX_CMD_STAT,
                pmac->MAC_0, pmac->MAC_1, pmac->COMMAND_CONFIG);}
```

//Gönderilecek paketin hedef adres alanına hedef MAC adresinin yazılması

```
void setDestAddr( char *buf, char * addr) {
   for(int i=0; i < 6; i++ ) {
     ((char*)buf)[i] = addr[i];}}
```

//Len uzunlugunda çerçeve olusturma

```
void createNormalFrame( char *buf, char *addr, int len) {
    for( i = 0; i < len; i++ ) {
     ((char*)buf)[i] = i;
   }
   l = len-14;
   buf[12] = (l >>8) & 0xff;
   buf[13] = l & 0xff;
   setDestAddr( buf, addr );
}
#define NTLPHY_ID 0x20005c70    /* National DP83865 */
#define MVLPHY_ID 0x01410cc0    /* Marvell 88E1111 */
#define NTL848PHY_ID 0x20005c90 /* National 83848, 10/100 */
#define TDKPHY_ID    0x0300e540   /* TDK 78Q2120 10/100 */
int main(){
alt_u32 ms_per_tick;
int tm,len,is1000,dat,ds, phy_id, allOk, mac_base, dma_base, rx_fifo_base, tx_fifo_base;
int *rxbuf;
char *xmem;
```

//Alinacak paketlerin veya gönderilecek paketlerin saklanacağı adres alanları

```
       xmem    = (char*)((int)malloc(8192) );
       g_rxbuf = (char*)((int)malloc(8192) );
       xmem = (char*)alt_remap_uncached( xmem, 8192 );
       g_rxbuf = (char*)alt_remap_uncached( g_rxbuf, 8192 );
       rxbuf = (int*)malloc(1536);
```

//Baslatma fonksiyonu ile donanımlar baslatılır.

```
mtip_mac_initTransInfo2( &mi, MAC_CONTROL_PORT_BASE,
DMA_BASE, 0,MAC_RXFIFO_BASE, MAC_TXFIFO_BASE,0, rxbuf );
isMVL =0;
```

```
        do {
           retry = 0;
           for(i=0; i < 32; i++) {
              mi.mac->MDIO_ADDR0 = i;
              dat = IORD(&mi.mac->mdio0.PHY_ID1,0);
              ds  = IORD(&mi.mac->mdio0.PHY_ID2,0);
// MDIO adreslerinin taranmasi
              if( dat != ds ) {
                 if( i == 31 ) {
                    retry = 1;
           IOWR(&mi.mac->mdio0.CONTROL,0,0x9000); // MDIO adreslerinin yazılması
                    usleep(10000);
                 }
                 break;}}}
 while(retry);
// Fiziksel entegre tanınıyor
        phy_id = ((dat & 0xffff) << 16) | (ds & 0xfff0);
        IOWR(&mi.mac->mdio0.CONTROL,0,0x9000);
        usleep(10000);
        #if 1
//Fiziksel arabirim hızı belirleme
        dat = IORD(&mi.mac->mdio0.CONTROL,0) & 0x001f;
        #ifdef GIGABIT
        #else
        if( isMVL ) {
           IOWR(&mi.mac->mdio0.CONTROL,0,dat | 0xe100);
        }
        IOWR(&mi.mac->mdio0.CONTROL,0,dat | 0x6100);
        #endif
        #endif
        x=0;
        while( (IORD(&mi.mac->mdio0.STATUS,0) & PCS_ST_rx_sync) == 0 ){
           if( x++ > 100000) break;
        }
        printf("OK. x=%d, STATUS=%04x\n",x, IORD(&mi.mac->mdio0.STATUS,0));
        usleep(10000);
        dat = IORD(&mi.mac->mdio0.reg11,0);
        is1000 = ( dat==2 ) ? 1 : 0;
        if(     dat==0) printf(" 10Mbps");
        else if(dat==1) printf(" 100Mbps");
        else if(dat==2) printf(" 1000Mbps");
        printf("-----------------------------------\n");
        printf("MAC resetleniyor..");
     if( is1000 ) {
        mi.mac->COMMAND_CONFIG = mmac_cc_SW_RESET_mask |
mmac_cc_TX_ENA_mask | mmac_cc_RX_ENA_mask | mmac_cc_ETH_SPEED_mask;
     } else {
     }
     x=0;
```

```
   while(mi.mac->COMMAND_CONFIG & mmac_cc_SW_RESET_mask) {
      if( x++ > 1000 ) break;
   }
   dat = IORD(&mi.mac->COMMAND_CONFIG,0);
   if( (dat&0x03)  != 0 ) {
      printf("WARN: RX/TX not disabled after reset... missing PHY clock?
CMD_CONFIG=0x%08x\n", dat);
   } else {
      printf("OK, x=%d, CMD_CONFIG=0x%08x\n\n", x, dat);
   }
//10/100 Ethernet MAC Core registerleri konfigüre et
      i = waitRxEmpty( &mi );
      if(i)printf("RX FIFO  %d frame\n", i);
//Çerçeve oluşturmak için hafıza alanına veri yaz
      for(i=0; i < 512; i++ ) {
            xmem[i] = i;
          }
xx:
//Çerçeve gönderme ve alma fonksiyonları çalıştırılıyor
//Çerçeve alınmamış ise hafızadaki çerçeveyi gönder.
//Çerçeve alınmış ise hafızaya yaz ve hafızadaki çerçeveyi geri gönder
 tm=0;
      while( !(mtip_mac_isFrameAvail( mi.mac)) ) {
           if( ++tm == 1500000 ) {
                 setDestAddr( xmem, destAddr );
                 xmem[12] = 0x0a;
                 xmem[13] = 0x00;
                 mtip_mac_sTxWrite( &mi, (int*)xmem, 128 );}}
        do {
         printf("aliniyor... ---------- \n");
         len = mtip_mac_getFrameLength( mi.mac );
         printf("rx: Alinan çerçeve uzunlugu: %d\n", len);
         dat = mtip_mac_sRxRead( &mi, (int*)xmem, len );
//Alinan paketi ekrana yaz
                        for(i=0; i < len; i++ ) {
                 printf(" %02x", (int)(xmem[i])&0xff );  }
//Paketin normalize değerlerini elde et
if(((int)(xmem[0])&0xff) <=1){
               dmacc=1;
             }
               else{
                  dmacc=2;
               }
if(((int)(xmem[6])&0xff) <=0x1){
               smacc=1;
             }

               else{
                  smacc=2;
```

```
                }
if(((int)(xmem[26])&0xff) <= 126){
                sipp=1;
              }
              else if (((int)(xmem[26])&0xff) >=128 & ((int)(xmem[26])&0xff) <=191){
                  sipp=2;
                }
                else{
                  sipp=3;
                }
if(((int)(xmem[30])&0xff) <= 126){
                dipp=1;
              }
              else if (((int)(xmem[30])&0xff) >=128 & ((int)(xmem[30])&0xff) <=191){
                  dipp=2;
                }
                else{
                  dipp=3;
                }
//Alınan paketin normalize değeri
printf("\n %d%d%d%d%d%d%d%d \n %d
\n%d",dmacc,smacc,protocoll,sipp,dipp,sportt,dportt,lenghtt,saldiri,normal);
//Paket için olasılıkları hesaplama
bayeshesapla(dmacc,smacc,protocoll,sipp,dipp,sportt,dportt,lenghtt,saldiri,normal);
saldiri=round((sal)/2.2158288342);
normal=round((nor)/512.321365182);
//Paketin saldırı veya normal olduğunu belirle
if (saldiri>= normal){
//printf("saldiri");
}
else{
//printf("saldiri degil");
}
//Alma ve gönderme işlemlerini tekrar et.
          goto xx;
        }
}
```

EK.4 Paketlerin gönderilmesini, alınmasını ve sınıflandırılmasını sağlayan Karar Ağacı Programı

```
#include <stdlib.h>
#include "system.h"
#include <mtip_avalon_10_100_1000_mac_regs.h>
#include <altera_avalon_dma_regs.h>
// Alınan paketin normalize sonuçlarına göre sınıfını belirle
kurallar( ){
if(protocol<1.5 & K.Port<2.5 & H.IP<2.5 & K.IP<2,5 & uzunluk<1.5 & H.IP>=1.5)
if(protocol<1.5 & K.Port<2.5 & H.IP>=2.5)
if(protocol<1.5 & K.Port>=2.5 & H.IP<2.5 &uzunluk>=1.5)
if(protocol>=1.5 & H.Port>=1.5 & uzunluk<2.5 & H.Port<2.5 & H.IP>=2.5)
if(protocol>=1.5 & H.Port>=1.5 & uzunluk<2.5 & H.Port>=2.5 & K.IP<1.5)
return sonuç;
}
//Başlatma Fonksiyonu
void mtip_mac_initTransInfo2( mtip_mac_trans_info *mi, int  mac_base, int  dma_base,
      int  dma_rx_base, int  rx_fifo_base, int  tx_fifo_base, int  cfgflags, int  *rxbuf ) {
      mi->mac    = (np_mtip_mac*)mac_base;
      IOWR(&mi->mac->TX_CMD_STAT,0, mmac_tcs_SHIFT16_mask);
      if( IORD(&mi->mac->TX_CMD_STAT,0) == mmac_tcs_SHIFT16_mask ) {
         mtip_hasShift16=1;
      } else {
         mtip_hasShift16=0;
      }      }
// Paket gönderme fonksiyonu
//DMA kullanılarak hafızadaki paket Transmit FIFO' ya kopyalanır.
int  mtip_mac_sTxWrite( mtip_mac_trans_info *mi,      // MAC ve DMA Adresleri
                        int *src,     // Paketin hafıza alanındaki adresi
                        int length // Paketin uzunluğu) {
}
//Paket alma Fonksiyonu
// DMA kullanılarak, Receive FIFO' daki paket hafızaya kopyalanır.
int  mtip_mac_sRxRead( mtip_mac_trans_info *mi,      // MAC ve DMA adresleri
         int *dest,    // paketin kopyalanacağı adres
         int length// Paketin uzunluğu) {
}
mtip_mac_trans_info mi;
char myAddr[6] ={0xFB,0x43,0x36,0xAB,0xCF,0x15}; //Gömülü sistemin MAC adresi
char destAddr[6] ={0x00,0x1F,0xC6,0xC7,0xD2,0xEC}; //Paketin gönderileceği hedefin
MAC adresi
int waitRxEmpty(mtip_mac_trans_info *mi ) {
      int timeout, lenrx, status, rxcnt, int i;
      do {
            if( mtip_mac_isFrameAvail( mi-> mac ) ) {
                  status = mi->mac->RX_CMD_STAT;
                  lenrx = mtip_mac_getFrameLength( mi->mac );
                  mtip_mac_sRxRead( mi, (int*)g_rxbuf, lenrx );
```

```
                if( status & mmac_rcs_ERROR_mask ) printf(". ERR flag");
                timeout = 0;
                rxcnt++;
            } else {
                timeout++;
            }
        } while( timeout < 1000000 );
        return rxcnt;
}
```
//10/100 Ethernet MAC Core Registerleri
```
void print_mac_status(np_mtip_mac *pmac) {
printf("AVL_STATUS: %08x, RX_CMD_STAT: %08x, TX_CMD_STAT: %08x,
MAC_0=%08x,MAC_1=%08x, COMMAND_CONFIG: %08x\n",
                pmac->AVL_STATUS, pmac->RX_CMD_STAT, pmac->TX_CMD_STAT,
                pmac->MAC_0, pmac->MAC_1, pmac->COMMAND_CONFIG);}
```
//Gönderilecek paketin hedef adres alanına hedef MAC adresinin yazılması
```
void setDestAddr( char *buf, char * addr) {
   for(int i=0; i < 6; i++ ) {
     ((char*)buf)[i] = addr[i];}}
```
//Len uzunlugunda çerçeve olusturma
```
void createNormalFrame( char *buf, char *addr, int len) {
    for( i = 0; i < len; i++ ) {
     ((char*)buf)[i] = i;
   }
   l = len-14;
   buf[12] = (l >>8) & 0xff;
   buf[13] = l & 0xff;
   setDestAddr( buf, addr );
}
#define NTLPHY_ID 0x20005c70    /* National DP83865 */
#define MVLPHY_ID 0x01410cc0    /* Marvell 88E1111 */
#define NTL848PHY_ID 0x20005c90 /* National 83848, 10/100 */
#define TDKPHY_ID    0x0300e540   /* TDK 78Q2120 10/100 */
int main(){
alt_u32 ms_per_tick;
int tm,len,is1000,dat,ds, phy_id, allOk, mac_base, dma_base, rx_fifo_base, tx_fifo_base;
int *rxbuf;
char *xmem;
```
//Alinacak paketlerin veya gönderilecek paketlerin saklanacağı adres alanları
```
        xmem    = (char*)((int)malloc(8192) );
        g_rxbuf = (char*)((int)malloc(8192) );
        xmem = (char*)alt_remap_uncached( xmem, 8192 );
        g_rxbuf = (char*)alt_remap_uncached( g_rxbuf, 8192 );
        rxbuf = (int*)malloc(1536);
```
//Baslatma fonksiyonu ile donanımlar baslatılır.
```
mtip_mac_initTransInfo2( &mi, MAC_CONTROL_PORT_BASE,
DMA_BASE, 0,MAC_RXFIFO_BASE, MAC_TXFIFO_BASE,0, rxbuf );
isMVL =0;
        do {
```

```
          retry = 0;
          for(i=0; i < 32; i++) {
             mi.mac->MDIO_ADDR0 = i;
             dat = IORD(&mi.mac->mdio0.PHY_ID1,0);
             ds  = IORD(&mi.mac->mdio0.PHY_ID2,0);
// MDIO adreslerinin taranmasi
             if( dat != ds ) {
                if( i == 31 ) {
                   retry = 1;
          IOWR(&mi.mac->mdio0.CONTROL,0,0x9000); // MDIO adreslerinin yazılması
                   usleep(10000);
                }
                break;}}}
 while(retry);
// Fiziksel entegre tanınıyor
       phy_id = ((dat & 0xffff) << 16) | (ds & 0xfff0);
       IOWR(&mi.mac->mdio0.CONTROL,0,0x9000);
       usleep(10000);
       #if 1
//Fiziksel arabirim hızı belirleme
       dat = IORD(&mi.mac->mdio0.CONTROL,0) & 0x001f;
       #ifdef GIGABIT
       #else
       if( isMVL ) {
          IOWR(&mi.mac->mdio0.CONTROL,0,dat | 0xe100);
       }
       IOWR(&mi.mac->mdio0.CONTROL,0,dat | 0x6100);
       #endif
       #endif
       x=0;
       while( (IORD(&mi.mac->mdio0.STATUS,0) & PCS_ST_rx_sync) == 0 ){
          if( x++ > 100000) break;
       }
       printf("OK. x=%d, STATUS=%04x\n",x, IORD(&mi.mac->mdio0.STATUS,0));
       usleep(10000);
       dat = IORD(&mi.mac->mdio0.reg11,0);
       is1000 = ( dat==2 ) ? 1 : 0;
       if(     dat==0) printf(" 10Mbps");
       else if(dat==1) printf(" 100Mbps");
       else if(dat==2) printf(" 1000Mbps");
       printf("-------------------------------------\n");
       printf("MAC resetleniyor..");
    if( is1000 ) {
       mi.mac->COMMAND_CONFIG = mmac_cc_SW_RESET_mask |
mmac_cc_TX_ENA_mask | mmac_cc_RX_ENA_mask | mmac_cc_ETH_SPEED_mask;
    } else {
    }
    x=0;
    while(mi.mac->COMMAND_CONFIG & mmac_cc_SW_RESET_mask) {
```

```
      if( x++ > 1000 ) break;
   }
   dat = IORD(&mi.mac->COMMAND_CONFIG,0);
   if( (dat&0x03)  != 0 ) {
      printf("WARN: RX/TX not disabled after reset... missing PHY clock?
CMD_CONFIG=0x%08x\n", dat);
   } else {
      printf("OK, x=%d, CMD_CONFIG=0x%08x\n\n", x, dat);
   }
//10/100 Ethernet MAC Core registerleri konfigüre et
      i = waitRxEmpty( &mi );
      if(i)printf("RX FIFO  %d frame\n", i);
//Çerçeve oluşturmak için hafıza alanına veri yaz
      for(i=0; i < 512; i++ ) {
            xmem[i] = i;
          }
xx:
//Çerçeve gönderme ve alma fonksiyonları çalıştırılıyor
//Çerçeve alınmamış ise hafızadaki çerçeveyi gönder.
//Çerçeve alınmış ise hafızaya yaz ve hafızadaki çerçeveyi geri gönder
 tm=0;
      while( !(mtip_mac_isFrameAvail( mi.mac)) ) {
           if( ++tm == 1500000 ) {
                 setDestAddr( xmem, destAddr );
                 xmem[12] = 0x0a;
                 xmem[13] = 0x00;
                 mtip_mac_sTxWrite( &mi, (int*)xmem, 128 );}}
        do {
         printf("aliniyor... ---------- \n");
         len = mtip_mac_getFrameLength( mi.mac );
         printf("rx: Alinan çerçeve uzunlugu: %d\n", len);
         dat = mtip_mac_sRxRead( &mi, (int*)xmem, len );
//Alinan paketi ekrana yaz
                        for(i=0; i < len; i++ ) {
                 printf(" %02x", (int)(xmem[i])&0xff );  }
//Paketin normalize değerlerini elde et
if(((int)(xmem[0])&0xff) <=1){
              dmacc=1;
            }
              else{
                 dmacc=2;
              }
if(((int)(xmem[6])&0xff) <=0x1){
              smacc=1;
            }

              else{
                 smacc=2;
              }
```

```
if(((int)(xmem[26])&0xff) <= 126){
                sipp=1;
              }
              else if (((int)(xmem[26])&0xff) >=128 & ((int)(xmem[26])&0xff) <=191){
                  sipp=2;
                }
                else{
                  sipp=3;
                }
if(((int)(xmem[30])&0xff) <= 126){
                dipp=1;
              }
              else if (((int)(xmem[30])&0xff) >=128 & ((int)(xmem[30])&0xff) <=191){
                  dipp=2;
                }
                else{
                  dipp=3;
                }
//Alınan paketin normalize değeri
printf("\n %d%d%d%d%d%d%d%d \n %d
\n%d",dmacc,smacc,protocoll,sipp,dipp,sportt,dportt,lenghtt,saldiri,normal);
//Paket sınıfının tespiti için paketi kurallara uygula
kurallar(dmacc,smacc,protocoll,sipp,dipp,sportt,dportt,lenghtt,saldiri,normal)
//Paketin saldırı veya normal olduğunu belirle
if (tip== 1){
//printf("saldiri");
}
else{
//printf("saldiri degil");
}
//Alma ve gönderme işlemlerini tekrar et.
          goto xx;
        }
}
```

EK.5 DBC3C40 Geliştirme Kartı Özellikleri

· EP3C40F484C7N
· EPCS64 configuration device
· 2x DB83640 10/100 Ethernet Phy
· 2x 4 channel LVDS link on RJ45 sockets
· USB OTG transceiver
· LVDS TFT interface
· 16Mbyte SDRAM
· 1Mbyte SRAM
· 8Mbyte flash
· LM74 I²C temperature sensor
· 1x UART transceiver
· 2x CAN transceiver
· 4x RS485 transceiver
· 32 pin I/O connector
· 16bit 24V I/O interface
· 8x User LEDs
· 2 digit seven segment display
· 4x user buttons
· Navigation key
· JTAG interface
· Touch Screen Controller
· Security Eprom
· on board 12V, 5V, 3.3V and 1.2V power supply
· dimension : 100x100mm

ÖZGEÇMİŞ

1974-Elazığ doğumlu olan Taner TUNCER ilk, orta ve lise öğrenimini Elazığ'da tamamladı. 1992'de girdiği Fırat Üniversitesi Elektrik-Elektronik Mühendisliği Bölümünden 1996'da mezun oldu. 1997 yılında Bilgisayar Mühendisliği bölümüne uzman olarak atandı. 2000 yılında Fırat Üniversitesi, Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği Ana Bilim dalında yüksek lisans yapma hakkı kazandı. 2003 yılında yüksek lisan eğitimini tamamladı ve aynı yıl Fırat Üniversitesi, Fen Bilimleri Enstitüsü, Elektrik-Elektronik Ana Bilim dalında doktora eğitimine başladı. Halen Bilgisayar Mühendisliği Bölümünde Uzman olarak görevine devam etmektedir.