İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

MPC8260ADS GELİŞTİRME KARTI ÜZERİNDE İŞLETİM SİSTEMİ KURULUMU VE ÖRNEK AĞ UYGULAMALARI GELİŞTİRİLMESİ

YÜKSEK LİSANS TEZİ

Müh. İbrahim Evren ULUSOY

Anabilim Dalı : BİLGİSAYAR MÜHENDİSLİĞİ

Programı : BİLGİSAYAR MÜHENDİSLİĞİ

HAZİRAN 2007

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

MPC8260ADS GELİŞTİRME KARTI ÜZERİNDE İŞLETİM SİSTEMİ KURULUMU VE ÖRNEK AĞ UYGULAMALARI GELİŞTİRİLMESİ

YÜKSEK LİSANS TEZİ
Müh. İbrahim Evren ULUSOY
(504041518)

Tezin Enstitüye Verildiği Tarih : 7 Mayıs 2007
Tezin Savunulduğu Tarih : 14 Haziran 2007

Tez Danışmanı : **Prof.Dr. Emre HARMANCI (İ.T.Ü.)**
Diğer Jüri Üyeleri **Prof.Dr. Tevfik AKGÜN (Y.T.Ü.)**
Doc.Dr. Sema OKTUĞ (İ.T.Ü.)

HAZİRAN 2007

ÖNSÖZ

Komünikasyon sektöründe mikroişlemcilerin yeri giderek önem kazanmaktadır. Teknoloji ilerledikçe daha karmaşık ve daha fonksiyonel mikroişlemciler bugün olduğu gibi ileride de mühendislerin kullanımına sunulmaya devam edilecektir. Yarıiletken üreticiler, işlemcilerini test edip üzerilerinde örnek uygulamalar geliştirebilmeleri için mühendislere uygulama geliştirme kartları sunmaktadırlar. Bu kartlar işlemci ile birlikte bellekler, veri yolları gibi bir bilgisayar mimarisinde bulunması gereken ünitelerle birlikte gelmektedir.

RISC işlemci üretiminde dünyanın önde gelen üreticilerinden olan Freescale firması ürettiği işlemceler için çeşitli uygulama geliştirme kartları sunmaktadır. Bu tezde bir Freescale ürünü olan MPC8260ADS ağ uygulama geliştirme kartı üzerinde çalışılmış ve Linux işletim sistemi kullanılarak örnek bir uygulama geliştirilmiştir.

Bu çalışmada kartı temin eden Motorola firmasına, Motorola'da mühendis olarak görev alan Ali İrfan beye ve tez süresince yardımcı olan Prof. Emre Harmancı hocamıza teşekkür ederim.

MAYIS 2007 İbrahim Evren Ulusoy

İÇİNDEKİLER

ÖNSÖZ i
KISALTMALAR iv
TABLO LİSTESİ vi
ŞEKİL LİSTESİ vii
ÖZET viii
SUMMARY x

1. GİRİŞ VE ÇALIŞMANIN AMACI 1

2. MPC8260ADS UYGULAMA GELİŞTİRME KARTI VE ETHERNET PROGRAMLAMA 4
2.1 PowerPC™ Mikroişlemciler 4
2.1.1 Benzer Ürünler 7
2.2 MPC8260 BÜTÜNLEŞİK KOMUNİKASYON İŞLEMCİSİ 8
2.2.1 MPC603e İşlemci 11
2.2.2 Komünikasyon İşlemci Modülü CPM 17
2.2.3 MPC8260 Bellek Denetleyici 20
2.2.4 MPC8260 Veri Akışları 21
2.3 MPC8260ADS Uygulama Geliştirme Kartı 23
2.3.1 Özellikler 23
2.3.2 Kurulum 25
2.3.3 Yazılımlar 26
2.3.4 Codewarrior Yazılımı ile Uygulama Geliştirme 27
2.4 MPC8260ADS Ethernet Programlama 28

3. MPC8260ADS ÜZERİNDE LINUX İŞLETİM SİSTEMİ KURULUMU 41
3.1.1 Çapraz Derleme 42
3.1.2 Boot Loader (Das UBoot) Kurulumu 44
3.1.3 Codewarrior ile Flash Programlama 46
3.1.4 Çekirdek Derleme 51
3.1.5 Dosya Sistemi Oluşturma (BusyBox) 59
3.1.6 Brctl Kurulumu (Köprü Uygulaması) 61
3.1.7 IPtables2 ve Kurulumu 64
3.1.8 IProute2 ve Kurulumu 68
3.1.9 UBoot ile Flash Üzerine Çekirdek ve Dosya Sistemi Programlama 70

4 MPC8260ADS'NİN AĞ UYGULAMARINDA KULLANIMI 73
4.1 MPC8260ADS'nin Köprü Cihazı Olarak Kullanılması 73
4.2 MPC8260ADS'nin Yönlendirici Olarak Kullanılması 77
4.3 MPC8260ADS Üzerinde Ağ İletişim Hizmet Kalitesi Belirlenmesi (QoS) 78
4.4 MPC8260ADS Üzerinde Global Bir Örnek Uygulama 83

5. SONUÇLAR VE TARTIŞMA 93
5.1 İleriki Çalışmalar 94

KAYNAKLAR 96

EKLER 98

ÖZGEÇMİŞ 116

KISALTMALAR

ADS	: Application Development System
AIM	: Apple, IBM, Motorola
ALU	: Arithmetic Logic Unit
ATM	: Asynchronus Transfer Mode
BCSR	: Board Control and Status Register
BIOS	: Basic Input Operating System
BPU	: Branch Processing Unit
BRG	: Baud Rate Generator
CEPT	: Conference des administrations Europeanes des Postes et Telecommunications (European Conference of Postal and Telecommunications Administrations).
CP	: Communications Processor
CPM	: Communications Processor Module
DHCP	: Dynamic_Host_Configuration_Protocol
ELDK	: Embedded Linux Development Kit
EPP	: ECP Parallel Port
EXT2	: 2. Geliştirilmiş Dosya Sistemi
FCC	: Fast Communications Channel
FCCE	: FCC Ethernet Event Register
FCCM	: FCC Ethernet Mask Register
FDSR	: FCC Data Synchronization Register
FLOPS	: Floating-Point Operations per Second
FPSMR	: FCC Ethernet Mode Registers
GCI	: General Circuit Interface
GFMR	: General FCC Mode Registers
HDLC	: High-Level Data Link Control
IDE	: Integrated Development Environment
IMMR	: Internal Memory Map Register
IO	: Input/Output
ISDN	: Integrated Services Digital Network
JTAG	: Joint Test Action Group
L2	: Level 2
LSU	: Load Store Unit
MBIT	: Megabit
MCC	: Multi Channel Controller
MDC	: Management Data Clock
MDDIS	: Management Data Disable
MDIO	: Management Data Input Output
MF	: Multi Function
MHZ	: Megahertz
MMU	: Memory Management Unit
MPC8260-TCOM	: MPC8260ADS Daughter Card for Telephony Applications (T1)
NAT	: Network Address Translation
OEA	: Operating Environment Architecture
PCM	: Pulse-code modulation
PDA	: Personal Digital Assistant
POWERPC	: Performance Optimization with Enhanced Risc Performance

Computing

QUICC : Quad Integrated Communication Controller
RCCR : RISC Controller Configuration Register
RISC : Reduced Instruction Set Computing
SCC : Serial Communications Controller
SDLC : Synchronous Data Link Control
SDRAM : Synchronous Dynamic Random Access Memory
SMC : Serial Management Controllers
TCP : Transmission Control Protocol
TDM : Time Division Multiplexing
TSA : Time Slot Assigner
UART : Universal Asynchronous Receiver/Transmitter
UISA : User Instruction Set Architecture
VEA : Virtual Environment Architecture RISC

TABLO LİSTESİ

Sayfa No

Tablo 2.1: RX Tampon Bölge Alan Açıklamaları…… 30
Tablo 2.2: TX Tampon Bölge Alan Açıklamaları…… 31
Tablo 2.3: LXT970 Saklayıcı Değerleri…… 36
Tablo 2.4: MF Voltaj Seviyeleri…… 37
Tablo 2.5: MPC8260ADS ve TCOM Üzerinde Tespit Edilen MF Değerleri…… 37
Tablo 2.6: LXT970 Sürücü Konfigürasyonları…… 38
Tablo 3.1: Flash Bellek Adresleme…… 47
Tablo 3.2: LH28F016SCT-Z4 Komutlar…… 49
Tablo 3.3: Sürücü Yamaları…… 57
Tablo 4.1: Ethernet Kapı Adresleri…… 85

ŞEKİL LİSTESİ

		Sayfa No
Şekil 2.1:	PowerPC Mimarisinde Saklayıcılar……	6
Şekil 2.2:	MPC8260 Fonksiyonel Bloklar……	9
Şekil 2.3:	MPC8260 Üzerinde Veriyolları……	10
Şekil 2.4:	MPC8260 Blok Diyagramı……..…	12
Şekil 2.5:	603e ‘de Blok Yapısı……..	14
Şekil 2.6:	Adres Dönüşümü……..	15
Şekil 2.7:	Önbellek Yapısı……	16
Şekil 2.8:	Önbellek Küme Yapısı……	17
Şekil 2.9:	MPC8260 Bellek Denetleyeciler……..	20
Şekil 2.10:	MPC8260 Veri yolları……..	22
Şekil 2.11:	MPC8260ADS Blok Diyagramı……..	24
Şekil 2.12:	FCC Genel Blok Diyagramı……	28
Şekil 2.13:	FCC Tampon Bellek Yapısı……	29
Şekil 2.14:	Tampon Bölge Tanımlayıcıları……	30
Şekil 2.15:	MPC8260 ve Ethernet Fiziksel Sürücü Bağlantı……..	33
Şekil 2.16:	LXT970 Ethernet Fiziksel Sürücü……	35
Şekil 2.17:	LXT970 MII Yolu Üzerinden Veri Okuma……..	36
Şekil 2.18:	LXT970 MII Yolu Üzerinden Veri Yazma……..	36
Şekil 2.19:	LXT970 MF Bacağının Alabileceği Değerler……..	37
Şekil 3.1:	CodeWarrior Flash Bellek Programlayıcısı……....	46
Şekil 3.2:	OCD Commander……	48
Şekil 3.3:	Flash Programlayıcı Ayar Ekranı……...	51
Şekil 3.4:	Netfilter/IPTABLES Trafik Akışı……..	66
Şekil 4.1:	MPC8260ADS ile Köprü Uygulaması……	73
Şekil 4.2:	Köprü Uygulaması Çalışırken……..…	75
Şekil 4.3:	Hiyerarjik Jetonlu Kova Yapısı……	81
Şekil 4.4:	Örnek Uygulama Topolojisi……..	83
Şekil 4.5:	Örnek Uygulamada Servis Kalitesi Yapısı……	86
Şekil 4.6:	"tc" Komutları Çalıştırılmadan Ölçülen Trafik……....	91
Şekil 4.7:	"tc" Komutları Çalıştırıldıktan Sonra Ölçülen Trafik…….	91

MPC8260ADS GELİŞTİRME KARTI ÜZERİNDE İŞLETİM SİSTEMİ KURULUMU VE ÖRNEK AĞ UYGULAMALARI GELİŞTİRİLMESİ

ÖZET

Bu tez çalışmasında Motorola firmasına ait MPC8260ADS isimli ağ uygulama geliştirme kartı üzerinde çalışılmıştır. Çalışma sonrasında MPC8260ADS geliştirme kartı ve MPC8260 işlemcinin çalışma prensipleri anlatılmış ve pratikte kart işlevsel hale getirilmiştir.

MPC8260ADS üzerinde MPC8260 isimli işlemciyi barındırmaktadır. Temel olarak MPC8260, PowerPC mimarisine sahip bir ana işlemci ve komünikasyon modülünden oluşmaktadır. Komünikasyon modülü ise RISC mimarisine sahip komünikasyon işlemcisi ve çeşitli çevrebirim denetleyecilerinden oluşmaktadır. MPC8260ADS üzerinde geçici ve kalıcı harici bellekler de bulunmaktadır fakat kullanıcı yüklemediği sürece kalıcı bellek üzerinde işletim sistemi yer almamaktadır.

Kartı işlevsel hale getirmek için iki yöntem kullanılmaktadır.. Birincisi; kartı düşük seviyede programlama yöntemleri kullanarak hazırlamaktır. Fakat bu yöntem işlemcinin kesme rutinlerini programlamak gibi detayları da içermektedir. Kolaylık sağlaması adına programlamanın büyük bir kısmında C kullanılabilmesine karşın genelde kodlanan program içerisinde saklayıcılara ve bellek alanlarına özel veriler yazmayı gerektirdiğinden karmaşıklığı fazladır.

Çalışmanın ilk aşamasında kartın daha detaylı tanınması ve dokümente edilebilmesi için karmaşıklığına rağmen ilk yöntem kullanılmıştır. Bu yöntem ile MPC8260ADS üzerindeki iki Ethernet kapı arasında paket iletimi görevini yerine getiren bir uygulama geliştirilmiştir. Uygulamanın kodlanması detaylı ve karmaşık olması sayesinde kontrol edilebilecek ve yapılandırılması gereken tüm alanlar tespit edilmiştir. Böylece bu uygulamada karşılaşılan problemler ve çözümleri bir sonraki yöntemde yaşanan problemlerin çözümleri için yardımcı olmuştur.

İkinci yöntem, kart üzerinde bir işletim sistemi çalıştırmayı gerektirir. Böylece kullanıcı, detaylı ve karmaşık düzeydeki düşük seviyeli işlemlerle uğraşmak yerine geliştirilen ağ uygulamasının detaylarına odaklanabilir. Ayrıca ihtiyaç duyulduğunda açık kodlu hazır yazılımlar MPC8260ADS'ye de uyarlanabilir. Çalışmanın ikinci kısmında MPC8260ADS üzerinde işletim sistemi koşturularak bu yöntem kullanılmıştır. İşletim sistemi olarak gömülü sistemlerdeki üstünlüğünden dolayı Linux seçilmiştir. 2.4 sürümüne sahip Linux çekirdek MPC8260ADS'ye uygun olarak değiştirildikten sonra PowerPC mimarisine uygun olarak derlenmiştir. Temel bir sistemin ihtiyaç duyduğu kök dizin yapısı da yine PowerPC mimarisine uygun olarak hazırlanmış ve çekirdek ile birlikte kalıcı belleğe programlanmıştır. Daha sonra karta işlevsellik kazandırabilmek için ağ alanında açık kodlu farklı yazılımlar derlenip kök dizin imajına eklenmiştir. MPC8260ADS, farklı yerel ağları internete tek bir IP üzerinden ulaştırabilmek için NAT sunucu olarak yapılandırılmıştır. Internet'e doğru yapılan dosya transferlerinin tersi yönde yapılan transferlerin kalitesini en az seviyede düşürmek için gönderilen paketlere farklı öncelikler verilmiştir. Böylece MPC8260ADS'nin bir servis kalitesi düzenleyicisi olarak çalıştırıldığı gösterilmiştir. DHCP uygulamasının da kart üzerinde kurulmasıyla birlikte yerel ağ üzerindeki bilgisayarlar adreslerini MPC8260ADS üzerindeki yapılandırma dosyasına bağlı

olarak otomatik alabilmektedir. Linux çekirdeğinin özelliklerinden faydalanarak MPC8260ADS yönlendirici olarak da çalışabilmekte böylece farklı ağ adreslerine sahip yerel ağlar birbirlerine erişebilmektedir. Ayrıca farklı bir uygulamada da MPC8260ADS bir yerel ağı üçe bölerek ikinci seviye köprüsü olarak da çalıştırılmıştır. SSH sunucunun kurulumuyla birlikte sistemi kontrol etmek için seri bağlantı kablosunun kullanım zorunluğu kaldırılmış ve sisteme dışarıdan güvenli bir şekilde erişilebilmiştir. Çalışmanın bu kısmının sonunda MPC8260ADS amaçlandığı gibi çeşitli ağ görevlerini yardımcı bir bilgisayara bağlı olmadan tek başına yerine getirecek konuma getirilmiştir.

INSTALLING AN OPERATING SYSTEM AND DEVELOPING SAMPLE NETWORK APPLICATIONS ON MPC8260ADS DEVELOPMENT BOARD

SUMMARY

In this study, a network related application development board from Motorola named as MPC8260ADS is studied. At the end of the study the working principles of the MPC8260ADS is documented and the board made functional in practical as well.

MPC8260ADS development board includes the MPC8260 versatile communications processor. The MPC8260 processor involves two cores; a PowerPC RISC based processor and a RISC based communication processor which is included in the communications processor module part of the processor.

There are two ways in order to make the board functional. Programming the board in low level is the first way. But this method includes complex details such as coding the interrupt handlers. Complexity is high because even though in sake of simplicity C language can be used during the coding phase, most of the program still contains transfering special values to the registers and mapped memory areas.

In order to know the details of the board and those to be documented in the first part of the study an application which forwards packets between two Ethernet ports is developed on MPC8260ADS. Because the codinging of this application is complicated and detailed, all problems experienced and their solutions helped in order to solve the problems experienced in the next method.

Second method requires an operating system running on MPC8260ADS. Therefore user can focus on the details of the network application instead of dealing with complex and detailed low level tasks. If required, also open source applications could be adapted to MPC8260ADS. In the second part of the study this method is used. As an operating system Linux is chosen because of its dominance in embedded operating systems area. 2.4 version of Linux kernel modified to the needs of MPC8260ADS and compiled as suitable for PowerPC architecture. As a basic operating system requirement, a root file system is also compiled suitable with PowerPC architecture and programmed into the permanent memory with the kernel. Afterwards, in order to make the board's functionality more sensible, several open source network applications compiled and added into the root file system image. Therefore, MPC8260ADS served as a NAT server in order to connect hosts to Internet over a single IP address. During file uploads, in order not to decrease the quality of downloads, different priorities assigned to different flows. Hence it's proven that the MPC8260ADS can be used as a quality of service manager device. By installing the DHCP server into MPC8260ADS, all hosts connected to the local networks can get their IP configuration information automatically from the MPC8260ADS. By the routing capability of Linux kernel, MPC8260ADS can be used as IP router therefore any two hosts on different networks can reach eachother. Also in another application MPC8260ADS served as a layer 2 bridge by dividing one local network into three pieces. After compiling and installing the SSH server into MPC8260ADS, necessity of using serial cable while connecting to Linux console is removed. Therefore using SSH connection, MPC8260ADS can be reached from any host which has only a SSH client.

At the end of this part of the study, MPC8260ADS could operate standalone without connecting to a personal computer and also can handle networking tasks such as being a NAT server, router and quality of service manager.

1. GİRİŞ VE ÇALIŞMANIN AMACI

MPC8260ADS, MPC8260 PowerQUICC II™ işlemcisini barındıran bir uygulama geliştirme kartıdır. MPC8260, Freescale yarı-iletken üretici firmanın 1998 yılında piyasaya sürdüğü tek bir yonga içerisinde hem yüksek performanslı Power PC™ RISC işlemciyi hem de esnek bir komünikasyon işlemci ve ünitelerini birleştiren bir tümleşik işlemcidir.

Bu çalışmada MPC8260ADS'nin temel çalışma prensipleri araştırıldıktan sonra kartı pratikte işlevsel bir konuma getirmek amaçlanmıştır. MPC8260ADS'nin üç adet ethernet kapısından faydalanarak MPC8260ADS'yi IP yönlendirici, ikinci seviye köprü cihazı, servis kalitesi düzenleyici, DHCP sunucu gibi çeşitli görevleri yerine getirecek şekilde hazırlanması amaçlanmıştır.

Çalışma süresince Motorola'dan temin edilen üç adet temel parça kullanılmıştır. Bunlar MPC8260 işlemcinin üzerinde bulunduğu MPC8260ADS uyguluma geliştirme kartı, ağ giriş çıkışlarının çeşidini ve sayısını arttırmak için kullanılan MPC8260TCOM genişleme kartı ve kartı programlayabilmek için gerekli olan bilgisayar ile MPC8260ADS arasındaki bağlantıyı sağlayan JTAG adaptördür. Donanım haricinde kartın eğitim dokümanları, kullanım kılavuzları ve Codewarriror uygulama geliştirme yazılımının bulunduğu CD de Motorola'dan ayrıca temin edilmiştir.

Çalışmanın ilk kısmında kartın özellikleri araştırılmış ve çalışma prensipleri dokümante edilmiştir. Bölüm 2.1'de PowerPC mimarisi hakkında bilgi verildikten sonra benzer ürünler incelenmiştir. Bölüm 2.2'de MPC8260 bütünleşik işlemcisinin detayları ana işlemci ve komünikasyon modülü alt başlıkları ile incelenmiştir. Ayrıca bu bölümde işlemci içerisindeki veri yolları ve bellek yönetici modüllerinden de bahsedilmiştir. Kart üzerinde üç farklı bellek bulunmaktadır. Bunlardan ikisi geçici, birisi ise kalıcı tür bellektir. Geçici belleklerden bir tanesi iki adet 2 megabaytlık yongalar halinde karta lehimli şekilde 4 megabayttır. Diğeri geçici bellek ise 16 megabaytlık SDRAM modülü olarak kart üzerindeki yuvaya takılmıştır. Kalıcı bellek SHARP firmasının 8 adet yongasına sahip bir modülden oluşmaktadır ve toplam 8 megabaytlık veri alanına sahiptir. Bu modül de kart üzerindeki uygun yuvaya yerleştirilmiştir.

Kart ile ilgili araştırmalar yapıldıktan sonra PC üzerine Codewarrior yazılımı kurulmuştur. Yazılım için gerekli geçici lisans yine Motorola firmasından temin edilmiştir. Codewarrior yazılımının MPC8260ADS ile iletişim kurabilmesi için JTAG adaptörü kişisel bilgisayarın paralel çıkışı ile MPC8260ADS'nin JTAG giriş çıkış kapısına takılmıştır. MPC8260ADS'yi üzerinde basit bir test koşturabilmek için Codewarrior ile gelen örnek yazılımlar çalıştırılmıştır. Örnek yazılımlar döngü sınaması (loopback) kipinde çalışmaktadır ve çıkış kanalından birkaç paket gönderip, paketlerin geliş kanalından başarılı bir şekilde gelip gelmediğini test eden uygulamalardır. Bu yazılımlar örnek alınmış ve fiziksel bir çıkış kapısından paket gönderip alabilmek için çalışmalar yapılmıştır. Fiziksel ethernet sürücülerini kullanabilmek için örnek yazılıma Ethernet sürücü yongaları üzerinde bulunan saklayıcılara veri yazabilen fonksiyonlar geliştirilmiştir. MPC8260 ve fiziksel sürücü arasında veri akışının olabilmesi için ise MPC8260'nın bir çok saklayıcı için gerekli değerler girilmiştir. Tüm bu çalışmalar bölüm 2.4 içerisinde detaylı olarak anlatılmıştır. Böylece kodlamanın sonunda iki ethernet kapısı arasında paket aktarımı yapan bir uygulama geliştirilmiştir. Bu uygulama daha sonra geliştirlebilecek olan yeni bir uygulamaya da temel oluşturmaktadır. Geliştirme sürecinde yaşanan problemlerin çözümleriyle birlikte kazanılan deneyim ileriki bölümlerde kullanılmıştır.

Codewarriror ile MPC8260ADS ile pratikte faydalı bir uygulama geliştirmek için bir proje grubuna ihtiyaç vardır. Çünkü MPC8260ADS üzerinde işletim sistemi bulunmadığı gibi TCP/IP desteği de mikrokodlarında bulunmamaktadır. Bu nedenlerden dolayı kart üzerinde bir işletim sistemi koşturulması gerekmiştir. İşletim sistemlerinden Linux işletim sistemi, gömülü sistemlerdeki üstünlüğünden ve az yer kaplamasından dolayı seçilmiştir. Linux ve yardımcı yazılımlarının kurulumları bölüm 3 içerisinde anlatılmıştır. Bu bölüm içerisinde ilk olarak kişisel bilgisayarlarda BIOS'un yaptığı görevlerin benzerini yerine getiren ve işletim sistemini gerekli parametrelerle çağıran uboot yazılımı derlenip, flash belleğe programlanmıştır. Daha sonra Linux çekirdeği MPC8260ADS ve MPC8260TCOM'a göre yamalanarak derlenmiştir. Çekirdeğin çalışabilmesi için gerekli olan dosya sistemi ise busybox uygulaması kullanılarak hazırlanmıştır. Busybox, Linux ext2 dosya sistemine uygun bir imaj dosyası hazırlar ve içerisinde temel Linux komutlarını barındırır. Bu aşamadan sonra MPC8260ADS Linux işletim sistemi ile açılabilir konuma gelmiştir. Amaçlanan ağ uygulamaların gerçekleştirilebilmesi için Linux için yazılmış olan açık kodlu yazılımlar kullanılmıştır. Bu kurulumların bitirilmesiyle birlikte MPC8260ADS yönlendirici, köprü cihazı, servis kalitesi düzenleyicisi, SSH ve DHCP sunucusu gibi çeşitli görevleri yerine getirebilecek duruma getirilmiştir.

MPC8260ADS örnek bir ağ yapısında köprü görevini yerine getirecek şekilde yapılandırılmış ve test edilmiştir. Bu yapılandırma ile ilgili detaylar bölüm 4.1 içerisinde anlatılmaktadır. Üç farklı ağ adresine sahip ağ birbirine MPC8260ADS sayesinde bağlanmış ve ağ üzerindeki konakların birbirlerine erişebilmesi MPC8260ADS'nin IP yönlendirici özelliği sayesinde sağlanmıştır. Bu konuyla ilgili detaylar ise bölüm 4.2 içerisinde anlatılmıştır. Bölüm 4.3 içerisinde MPC8260ADS servis kalitesi düzenleyici nasıl kullanılacağı örneklerle anlatılmıştır.

Bölüm 4.4 içerisinde ise çeşitli özellikler bir arada kullanılarak örnek bir uygulama geliştirilmiştir. Diğer bölümlere ek olarak dosya sistemine SSH ve DHCP sunucu da eklenmiştir. Böylece dışarıdan Linux işletim sistemine seri bağlantı kablosunu kullanmadan da bağlanılabilmektedir. DHCP sunucusu ise ağ üzerindeki konaklara otomatik IP adresi verdiği için konaklar üzerinde el ile IP adresi ayarlarının yapılmasına gerek kalmamıştır. Uygulama, ADSL bağlantı yapısının getirdiği dezavantajları azaltabilmek için internet'e doğru olan akışlara farklı öncelik seviyeleri atamaktadır. Aynı zamanda bu uygulamada MPC8260ADS NAT sunucu olarak da görev almaktadır. Böylece farklı yerel ağlardaki konaklar internet'e tek bir IP adres üzerinden çıkabilmektedir. Daha önceki bölümlerdeki gibi MPC8260ADS IP yönlendirici olarak da görev almış ve böylece farkı ağlardaki konaklar birbirlerine erişebilmiştir. Tüm bu özellikleri destekleyen imaj dosyası flash belleğe de programlanmış ve MPC8260ADS artık pratikte birçok ağ görevini yardımcı bir bilgisayara bağlı olmadan yerine getirebilecek şekilde hazırlanmıştır.

Bölüm 5 ve 5.1 içerisinde yapılan çalışmalar özetlenmiş, hazırlanan uygulamaların geliştirilebilecek yanlarından bahsedilmiştir.

2. MPC8260ADS UYGULAMA GELİŞTİRME KARTI VE ETHERNET PROGRAMLAMA

MPC8260ADS kartı, üzerinde PowerPC™ ve RISC mimarisinde iki çekirdeğe sahip MPC8260 işlemcisini barındırmaktadır. Kart, ürünlerinde MPC8260 işlemciyi kullanmayı planlayan donanım ve yazılım geliştiriciler için bir çalışma ortamı sunmaktadır. Bu bölüm kart ve işlemci ile ilgili temel bilgiler ile Ethernet programlama ilgili detaylı bilgileri içermektedir.

2.1 PowerPC™ Mikroişlemciler

1990 yılında beş farklı RISC mimarisine sahip işlemci pazarda birbirleriye yarışmaktaydı. Pazar payının bölünmesi ve standardı olmayan mimarilerin ortaya çıkması RISC işlemcilerin giderek güç kaybetmesine neden oluyordu. Müşterilerin de çoklu işlemcilerin daha iyi fiyat performans oranına sahip olduğuna inanmaya başlaması, üç lider firmayı tek bir RISC mimarisinde buluşmak üzere bir araya getirmeye zorladı. 1991 yılı başlarında Apple, IBM ve Motorola'dan (AIM) işlemci mimarları, derleyici uzmanları, işletim sistemleri geliştiricileri ve işlemci tasarımcıları müşterilerin ihtiyaçlarına cevap verecek yeni bir mimari geliştirmek üzere bir araya geldiler. Fakat AIM'in sıfırdan başlayıp yeni bir mimari inşa etmek için yeterli zamanı olmadığı ortadaydı. Bu sebeple standart olarak kullanacakları yeni tasarımları olan PowerPC™ mimarisinin temellerini, IBM POWER™ mimarisine dayanarak oluşturdular [1].

PowerPC™ mimarisine sahip işlemciler, sunucu ve ev bilgisayarlarında kullanılmasının haricinde yoğun olarak gömülü sistemlerde, otomotiv sektöründe, oyun konsollarında da (Microsoft XBOX, Sony PS3 vb.) kullanılmaktadır. Ayrıca IBM'in Blue Gene süper bilgisayar projesinde 131.072 adet PowerPC 440 kullanılmıştır ve 280.600 FLOPS ile 100.000 FLOPS'un üzerine çıkabilen tek süper bilgisayardır [2].

PowerPC™ mimarisine sahip işlemciler temelde RISC prensipleri kullanır ve süperskalar yapıya imkân veren tasarıma sahiplerdir. Süperskalar bir işlemci, komut katmanında paralellik sağlar. Bu tür işlemciler komut boru hattının bir safhasında birden çok komutu, onları işlemcinin boşta olan uygun fonksiyonel ünitelerine

vererek, aynı anda çalıştırabilmektedirler. Böylece süperskalar bir işlemci aynı saat hızındaki normal bir işlemciye göre daha çok çıktı verebilmektedir.

PowerPC™ tasarımına uygun 32bit veya 64bit veri ile işlem yapabilen işlemci sürümleri mevcuttur. 64bit'lik sürüm, 32bit'lik sürümün süper kümesi olarak tanımlanmıştır böylece 64bit'lik bir PowerPC™ işlemci ile 32bit'lik PowerPC™ mimarisine uygun derlenmiş programlar çalıştırılabilir.

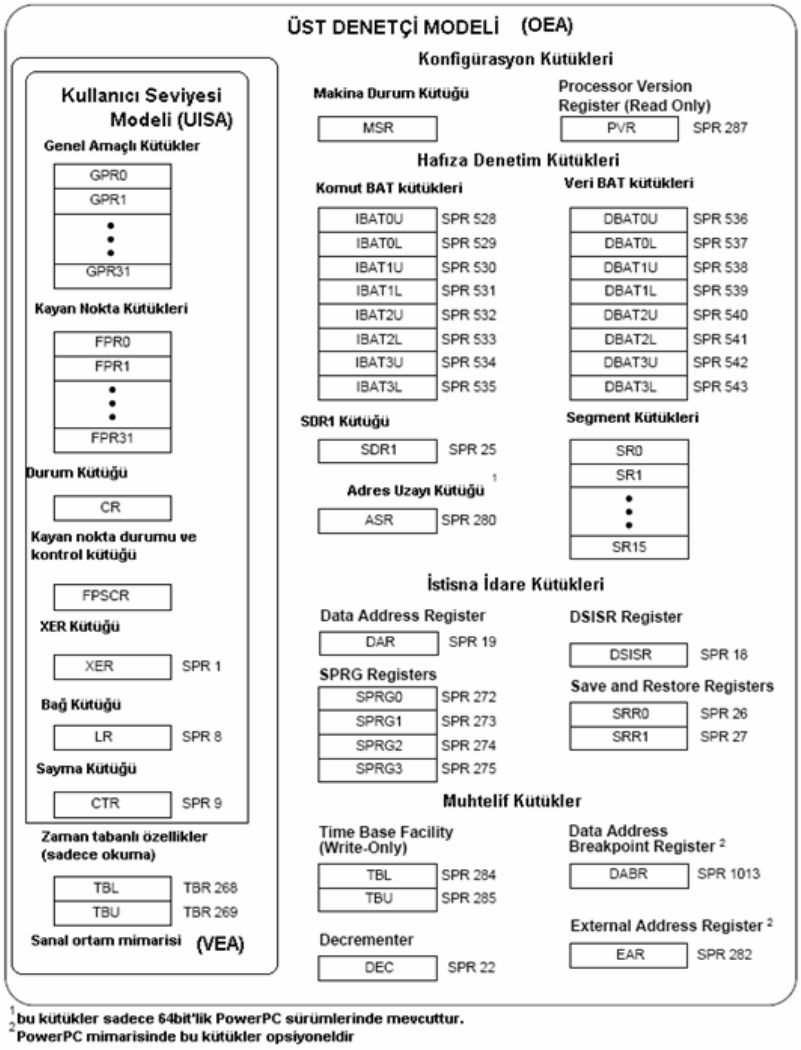
PowerPC™ mimarisi, kullanıcı seviyesi komut kümesi mimarisinden daha özel işletme seviyesi mimarisine kadar, üç programlama ortamına tekabül eden üç farklı seviyede tanımlanmıştır. Bu katmanlı mimari üreticiye esneklik ve çeşitli ürünleri arasında belirli mertebelerde yazılım uyumluluğunu sağlar. PowerPC™ mimarisinin üç farklı seviyesi aşağıdaki gibi tanımlanmıştır;

Kullanıcı komut kümesi mimarisi (UISA): UISA, PowerPC™ mimarisinde yazılımların kullanıcı seviyesinde uyması gereken katmanı tanımlar. Temel kullanıcı seviyesinde komut kümesi, kullanıcı seviyesi saklayıcıları, veri tiplerini ve kullanıcı yazılımından görünen istisna (exception) modelini UISA tanımlar. Tüm PowerPC™ mimarisine sahip işlemciler UISA'ya uyumluluğunu garanti etmektedir.

Sanal ortam mimarisi (VEA): VEA, genel kullanıcı yazılım ihtiyaçları dışında kalan kullanıcı seviyesindeki özellikleri tanımlar. Birden fazla aygıtın belleğe ulaşabildiği ortamlardaki bellek modelini VEA belirler. VEA ayrıca kullanıcı perspektifinden önbelleğe bakış açısını, önbellek komutlarını ve işlemcinin zaman tabanlı özelliklerini (işlemci frekansının belli bir oranında artan sayaç kütüğü gibi) belirler. VEA'yı destekleyen tüm PowerPC™ işlemciler aynı zamanda UISA'ya da uymak zorundadır fakat OEA'ya uymak zorunda değildir.

İşletim ortamı mimarisi (OEA): İşletim sistemi gibi yazılımlar tarafından ihtiyaç duyulan üst denetçi (supervisor) seviyesindeki kaynakları OEA tanımlar. PowerPC™ bellek yönetim modelini, üst denetçi seviyesindeki saklayıcıları, senkronizasyon gereksinimlerini ve üst denetçi seviyesindeki istisna modelini OEA tanımlar. OEA ayrıca üst denetçi bakış açısından zaman bazlı özelliklerin de tanımlandığı mimari seviyesidir. OEA'ya uygun tüm aygıtlar UISA ve VEA'yı desteklemek durumundadır. Tüm PowerPC™ aygıtlar UISA uyumludur ve böylece uygulama yazılımları arasındaki uyum sağlanmış olur fakat bazı aygıtlar gerekli durumlarda VEA ve OEA'nın farklı sürümlerini destekliyor olabilirler. Örneğin; bir gömülü denetleyici ihtiyacı doğrultusunda daha basit seviyede bir VEA ve bir OEA destekleyebilir.

PowerPC™ mimarisindeki saklayıcılar, seviyeleriyle birlikte Şekil 2.1'de gösterilmiştir [3].



Şekil 2.1: PowerPC Mimarisinde Saklayacılar

Komünikasyon ürünlerinde kullanılan yongalarda da PowerPC™ mimarisi gücünü korumaktadır. Bunun sebeplerinden biri ise Freescale, AMCC gibi üreticilerin ağ uygulamaları için tümleşik işlemciler geliştirmeleridir. Bu tip tümleşik işlemciler içlerinde RISC mimarisine sahip ana çekirdek ile birlikte zengin ve esnek fonksiyonlara sahip ağ işlemcileri de bulundururlar. Böylece donanım geliştiriciler iki farklı yonganın ara yüzlerini birbirleri ile konuşturmanın zorluğundan kurtulmakla kalmayıp hem maliyet duyarlı sistemler geliştirebilmektedirler hem de iki farklı işi daha az enerji kullanarak gerçekleyebilen işlemcilere sahip olabilmektedirler.

2.1.1 Benzer Ürünler

Freescale'in ürün yelpazesinde bulunan 32bit mikro denetleyici ve işlemciler, temel olarak dört dala ayrılmaktadır;

- Power Mimarisine sahip işlemciler: Bu mimaride firma karşımıza kendi alanlarında uzmanlaşmış işlemciler ile çıkmaktadır. Bilgisayar ağları, otomotiv, endüstriyel ve tüketici alanlarında değişik ürünler bulunmaktadır. Firmanın güncel bilgisayar ağları odaklı işlemcileri aşağıdakilerdir;
    - MPC603e
    - MPC7XX
    - MPC7XXX
    - MPC8XX PowerQUICC I
    - MPC82XX PowerQUICC II
    - MPC83XX PowerQUICC II Pro
    - MPC85XX PowerQUICC III
    - MPC8641/MPC8641D Dual Core

1993 yılında MC68360 ile başlayan QUICC teknolojisi 1994 yılında Power mimarisi ile birleşerek ilk PowerQUICC teknolojisine sahip MPC860 üretildi. QUICC, dörtlü tümleşik komünikasyon denetleyici (Quad Integrated Communication Controller) anlamına gelmektedir ve bu isim QUICC işlemcilerde 4 adet SCC (seri komünikasyon denetleyici) bulunmasından gelmiştir. [4].

Çoğu PowerQUICC teknolojisine sahip işlemciler, tümleşik işlemcilerdir ve içlerinde iki adet RISC çekirdek bulunur. Böylece ana çekirdek sadece kontrol yüzeyi ile ilgilenirken özel hızlandırıcılar ve giriş çıkış sistemlerine sahip olan komünikasyon modülü, veri yüzeyindeki işlemlerle meşgul olur. Bu iki ayrı işlemci beraber çalışmak üzere optimize edilerek performansı, iki ayrı işlemciye oranla daha hızlandırılmıştır. QUICC Engine™ teknolojisi ise PowerQUICC™ II Pro ailesinde bulunan MPC8360E işlemci ile gelen yeni bir teknolojidir ve getirdiği en çarpıcı özellik CPM içerisinde tek RISC işlemci bulundururken, QUICC motoru içerisinde iki adet RISC işlemci bulundurmasıdır. CPM hızları 333Mhz'e kadar çıkabilirken, QUICC Engine™ teknolojisi 500MHz'e kadar çıkabilmektedir. Bu özellikleriyle CPM'e oranla dört kat daha hızlı veri işleyebilmektedir. Ayrıca CPM'de 100Mbit olan maksimum Ethernet hızı QUICC Engine™ teknolojisi ile 1 Gbit hızına çıkartılmıştır. Yeni gelen diğer bir özellik ise QUICC Engine™ teknolojisi CPM'de veri işleme sınırı olan OSI ikinci katmanını,

üçüncü ve daha yukarı katmanlara çıkartmasıdır. Örneğin anahtarlama, paket iletme, MPLS, ATM'den Ethernet paket aktarma yeni QUICC Engine™ teknolojisi ile desteklenmektedir [5].

- ColdFire: Eski 68K ailesinin yeni ürünü. Ucuz ve performanslı yapısıyla maliyet duyarlı gömülü sistem pazarını hedeflemektedir. RISC mimarisine sahip işlemciler 16, 32 veya 48 bitlik komutlarla çalışabilmeleri daha efektif bellek ve veri yolu kullanımına imkân vermektedir. Geniş ve kapsamlı geliştirme araçlarına sahiptir.
- ARM® İşlemcileri: ARM tabanlı i.MX ailesine ait düşük enerji ile çalışan işlemciler akıllı telefonlarda, telsiz PDA'larda ve diğer mobil telsiz uygulamalarda kullanılmak üzere tasarlanmışlardır.
- MCORE işlemcileri: MCORE işlemciler piyasada düşük enerji kullanımlarıyla tanınmaktadır. Yüksek performans ve maliyet duyarlı denetim uygulamaları için tasarlanmıştır. Portatif ve pille çalışan seyyar ürünler kullanım alanları arasında yer almaktadır.

2.2 MPC8260 BÜTÜNLEŞİK KOMUNİKASYON İŞLEMCİSİ

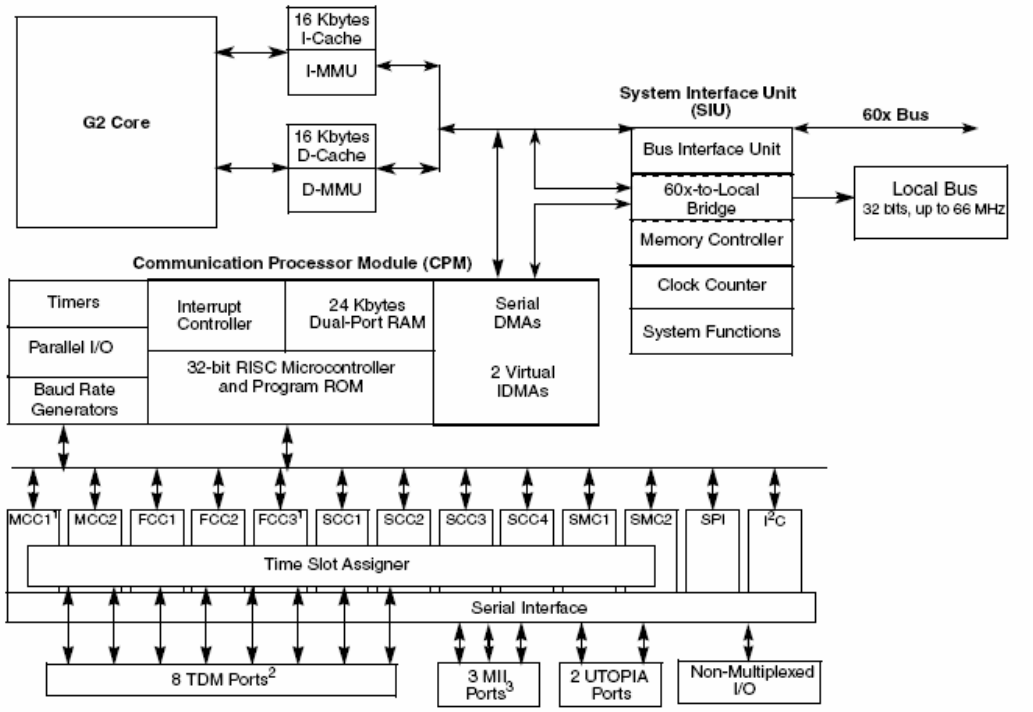
MPC8260 PowerQUICC II™ yüksek performanslı RISC mikroişlemci ile esnek bir entegrasyon ünitesini ve çeşitli komünikasyon çevrebirim denetleyicilerini bir araya getiren bir komünikasyon işlemcidir. Sektörde MPC8260 yoğunlukla komünikasyon ve bilgisayar ağlarında kullanılmaktadır. Örneğin Cisco'nun Catalyst 4224 ürününde 200MHz çekirdek hızına sahip MPC8260 işlemci yer almaktadır.

MPC8260'da kullanılan çekirdek işlemci, PowerPC MPC603e™'nin gömülü sürümüdür. Çekirdek işlemci 16Kb komut ve 16Kb veri önbelleğine sahiptir. Çekirdek işlemcide MPC603e'nin aksine bir kayan nokta işlem ünitesi mevcut değildir.

Komünikasyon işlemci modülü (CPM), komünikasyon işlemciyi (CP), çeşitli çevrebirim denetleyicilerini ve birbirinden bağımsız zamanlayıcıları içerir. MPC8260'da kullanılan CPM, yine Freescale firmasının MPC860 ürününün üst kümesi olarak tanımlanmıştır ve MPC860'a göre hem performans açısından geliştirilmiş hem de ekstra donanım özellikleri eklenmiştir.

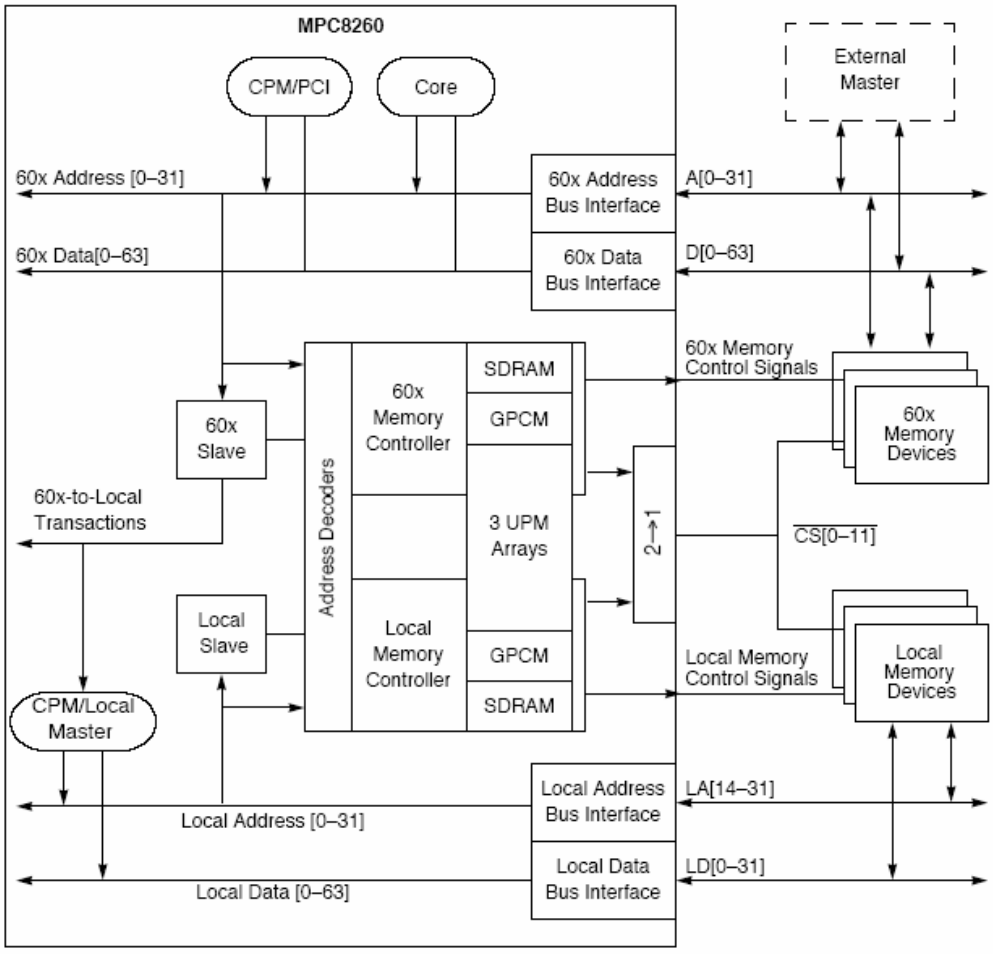
Sistem ara yüz ünitesi ise veri yollarını, bellek denetleyicilerini ve JTAG denetleyiciyi içerir.

MPC8260'ın genel yukarıda bahsedilen üç ana fonksiyonel bloktan oluşan yapısı Şekil 2.2'de verilmiştir [7].



Şekil 2.2: MPC8260 Fonksiyonel Bloklar

Şekil 2.3'de [7] de görüldüğü gibi MPC8260 işlemcide iki adet önemli veri yolu bulunmaktadır; 60x yol 32 bit'lik adres, 64 bit'lik veri genişliğine sahiptir. Lokal yol ise 18 bit'lik adres, 32 bit'lik veri genişliğine sahiptir.



Şekil 2.3: MPC8260 Üzerinde Veriyolları

Genel olarak PowerPC işlemci ile SDRAM arasındaki iletişim için 60x yol, CP ile komünikasyon çevre birimleri arasındaki işlemler için ise lokal yol tercih edilmektedir. MPC8260'ın kaynakları (parametre saklayıcıları, çift kapılı bellek vb.) bellek üzerinde bitişik bir alana eşlenmiştir. Bu alan toplam 128Kb büyüklüğündedir ve IMMR kütüğü ile bellek alanı üzerindeki yeri gösterilir. IMMR, ilk değerini "Hard Reset" konfigürasyon bilgisinden alır.

"Hard Reset" konfigürasyon bilgisi ise iki yöntem ile belirlenebilir; birincisi Flash bellek üzerinden okumak, ikincisi MPC8260ADS'nin üzerinde yer alan fakat işlemci dışında kalan BCSR kütüğünden okumaktır. Kart üzerinde bulunan DS1 anahtar kümesinin ilk anahtarı OFF konumunda ise kaynak BCSR, ON konumunda ise kaynak Flash bellektir. IMMR'nin açılış değeri "Hard Reset" konfigürasyona bağlı

olarak 32 değişik değer alabilir. MPC8260ADS kartında kaynak BCSR olarak belirtilmiş ise IMMR açılışta otomatik olarak 0x0f000000 değerini alır. Daha sonra buraya istenen farklı bir değer yazılarak 128Kb'lık bloğun bellek üzerindeki yeri değiştirilebilir.

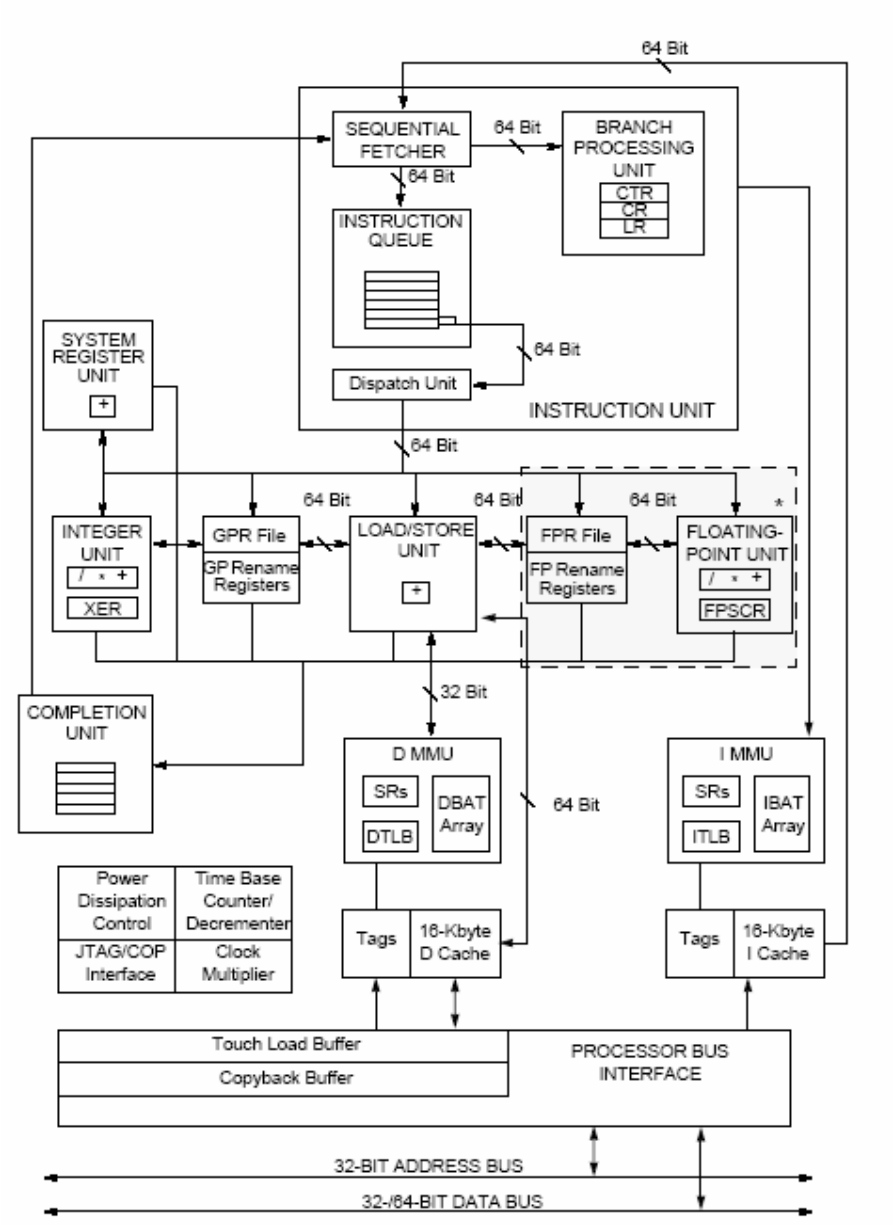
2.2.1 MPC603e İşlemci

MPC603e PowerPC mimarisinin 32bit'lik kısmını gerçekleştirmektedir. Çekirdek 32-bit efektif adresleme yapabilir ve 8,16, 32-bit'lik tamsayı veriler üzerinde işlem yapabilmektedir. MPC603e önbelleği, sürekli dinleyerek (snooping) CPM ve diğer ana işlemciler ile arasındaki veri tutarlılığını sağlar. MP8260, bir sistemde ikincil işlemci olarak da kullanılabilir ve bu durumda sadece MPC603e devre dışı bırakılabilir. Bu yöntemle diğer bir MPC8260 işlemciye veya başka bir işlemciye, ikincil işlemci görevini yerine getirir.

MPC603e'nin teknik özelliklerini aşağıdaki şekilde sıralanabilir;

- Yüksek performanslı, süperskalar işlemci.
- Her zaman biriminde üç komuta kadar işlem hızı.
- Çoğu komut için tek çevrimde işlem.
- 4 bağımsız işlem ünitesi ve iki saklayıcı dosyası
    - BPU (Branch Processing Unit) statik dallanma tahmin edici
    - 32-bit komut ünitesi (IU)
    - Önbellek ve GPR (General Purpose Register) arasında veri transferi için LSU (Load/Store Unit).
    - Durum kütüğünü (CR), özel amaç kütüğünü (SPR) ve tamsayı ekleme ve karşılaştırma komutlarını işleyen SRU.
    - 32 adet tam sayı değerler için GPR.
    - 32 adet kayan noktalı değerler için FPR.
    - Yüksek komut ve veri çıktı hızı.
    - Sıfır çevrimde dallanma yeteneği
    - 16Kbyte veri önbelleği: 4 yollu küme çağrışımlı, fiziksel adreslemeli ve LRU değişim algoritmalı bellek.
    - 16Kbyte komut önbelleği: 4 yollu küme çağrışımlı, fiziksel adreslemeli ve LRU değişim algoritmalı bellek.
    - Her sayfa ya da her blok için belirlenebilen geri yazmalı veya direkt yazmalı önbellek çalışma tarzı.

2.2.1.1 Komut Ünitesi



Şekil 2.4: MPC8260 Blok Diyagramı

Şekil 2.4'de [7] görüldüğü gibi komut ünitesi, bir getirme ünitesi (Fetch Unit), komut kuyruğu (Instruction Queue), sevk edici (Dispatcher Unit) ve bir dallanma kontrol ünitesinden (BPU) oluşmaktadır. Komut ünitesi, komutları komut önbelleğinden alıp, komut kuyruğuna gönderir. Dallanma ünitesi, dallanma komutlarını getirme ünitesinden alır ve henüz çözülmemiş ve duruma bağlı dallanmaların sonuçlarını statik dallanma tahmin yeteneğini kullanarak bulur. Böylece komut ünitesinin bulunan yeni yol üzerindeki yeni komutları getirmesini sağlar. Duruma bağlı

dallanmaların sonuçları belli olmadan önceden getirilen komutların işlenmeleri ardışık işleme modeli bozulmaması için tamamlanmaz. Eğer dallanma tahminlerden biri hatalı çıkar ise bütün tahmin edilen komut yolları ve tahmin edilen yoldan gelen komutlar temizlenir.

2.2.1.2 Tam Sayı Ünitesi (Integer Unit)

Tam sayı ünitesi adından da anlaşacağı gibi tüm tamsayı komutlarının işlenmesinden sorumludur. ALU işlemler esnasında, çarpıcı, bölücü ve XER kütüğünü kullanır. Çoğu tam sayı işlemleri tek devirde sonuçlanır. 32 adet genel saklayıcı, tam sayı işlemlerinde kullanılır. İşlemler esnasında genel saklayıcı çakışmalardan dolayı yavaşlamaları en aza indirgemek için “rename” kütükleri otomatik olarak kullanılabilir. Tamamlama ünitesinden çıktıktan sonra “rename” kütüklerinin değerleri daha önce çakışma yaşandığı için yazılamayan uygun genel kütüğe yazılır.

2.2.1.3 Yükleme / Saklama Ünitesi (Load/Store Unit)

Yükleme ve saklama işlemleri, ara yüz, genel kütükler ve önbellek arasındaki veri transfer işlemleri yükleme ve saklama ünitesi (LSU) içerisinde gerçekleşir. Efektif adreslerin hesaplanması, veri hizalanmasında da bu üniteden faydalanılmaktadır.

2.2.1.4 Sistem kütüğü Ünitesi (System Register Unit)

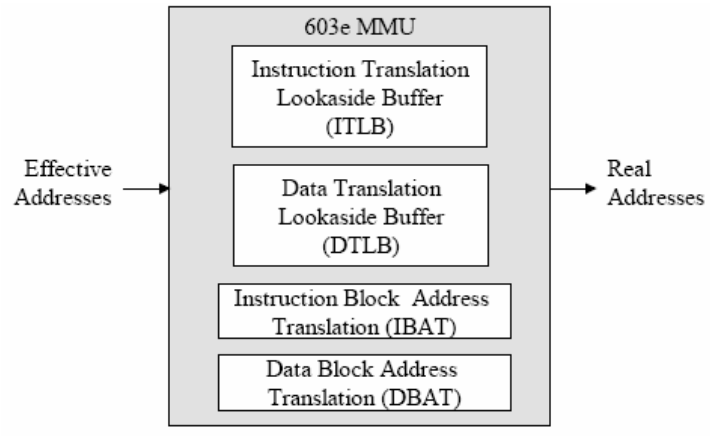
Çeşitli sistem seviyesindeki komutlar sistem kütüğü (SRU) ünitesi tarafından işlenir. Bunların arasında durum kütüğü mantıksal işlemleri, özel-amaçlı saklayıcı için taşıma komutları ve tamsayı toplama ve karşılaştırma işlemleri sayılabilir. Sistem durumunu tutarlı halde tutmak için SRU tarafından işlemler ardışık olarak çalıştırılır.

2.2.1.5 Tamamlama Ünitesi (Completion Unit)

Tamamlama ünitesi bir komutun yaşam süresini baştan sona takip edip eğer ve etkilenen kütükler var ise onlara işlem tutarlı bir seviyeye geldikten sonra doğru değerleri atar. Eğer yanlış bir dallanma tahminden geri dönülme aşamasına gelinirse tamamlama ünitesi sistemi kararlı bir duruma getirme görevini üstlenir.

2.2.1.6 603e Bellek Yönetim Ünitesi (MMU)

Şekil 2.5'de [7] gösterildiği gibi 603e'nin bellek yönetim ünitesi (MMU) dört ana parçadan oluşmaktadır. Bellek yönetim ünitesinin önemli kullanım alanları aşağıdakilerdir;

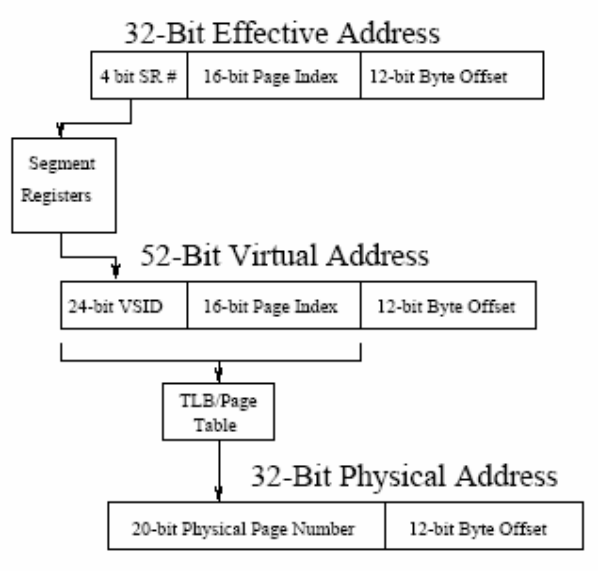


Şekil 2.5: 603e 'deBlok Yapısı

- Hak kontrolü: Süpervizör seviyesindeki alanlara kullanıcı seviyesindeki programların erişimini engellemek.
- Önbellek kontrolü: Önbelleği G/Ç aygıtlarına erişim durumlarında kapalı tutarken diğer alanlarda aktif olarak çalışmasını sağlayabilmek.
- Yazma koruması: Bellekte belirli alanlarını sadece okuma için işaretleme.
- Bellek koruma: Programların bellekte sadece belirli alanların kullanımını mümkün kılma. Böylece farklı programlar, birbirlerinin bellek alanlarını bilerek ya da bilmeyerek bozamazlar.
- Adres çevirme: Efektif adresleri fiziksel adrese çevirir ve farklı programların aynı mantıksal adres aralığını kullanmasına olanak verir.

Çekirdek, MMU'ya efektif adres ile gelir ve MMU bu adresleri gerçek yani fiziksel adrese dönüştürür. Fiziksel adres iki yöntemle bulunabilir; blok adres dönüştürme veya sayfa adresleme. Sayfa adreslemede 32 bitlik efektif adresin (EA0-EA31) en yüksek 4 biti 16 segment kütüğünden birini seçmek için kullanılır. Her segment kütüğünde 24 bitlik sanal segment tanımlayıcısı vardır. 24 bitlik bu adresin yanına efektif adresten 16 bitlik sayfa indeksi getirilerek (E4-E29) sayfa tablosundan fiziksel karşılığı aranır. 24bitlik sanal segment tanımlayıcı, 16bitlik sayfa indeksi ve 12bitlik bayt ofseti toplamda 52bitlik sanal adresi oluşturur. Bu sanal adres 1milyon işlem tarafından (2^{20}) paylaşılır ve her işlem için 4Gb'lık adresleme alanı ayrılmış olur. Fiziksel adres en çok 4Gb olabileceğinden dolayı sayfa tablosunu ve belleği işlem

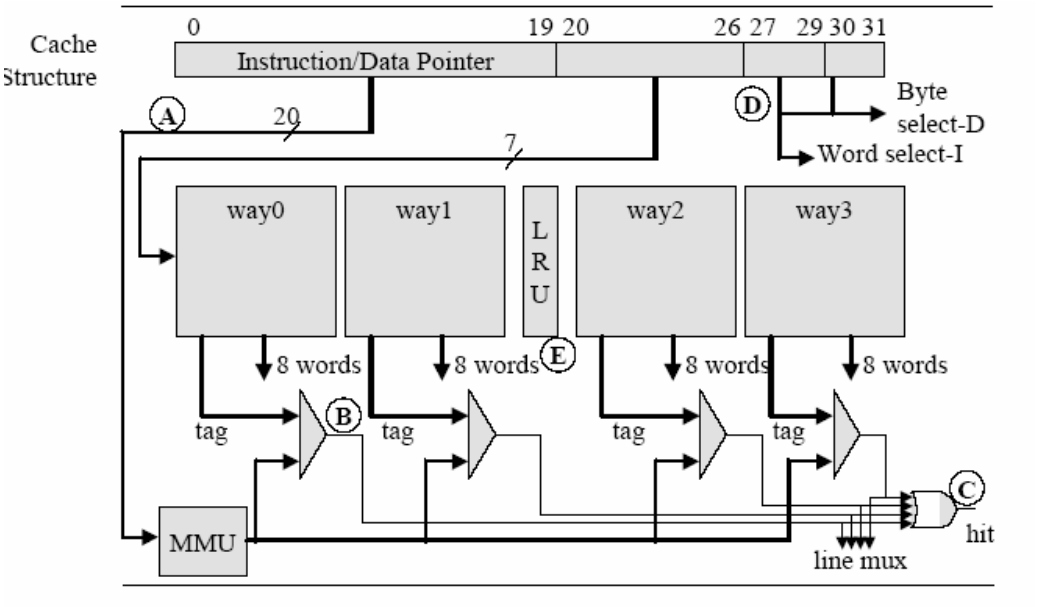
değişimlerinde uygun şekilde değiştirmek gerekir ve bu işletim sisteminin görevidir. Örneğin çok sık kullanılmayan bir program bellekteki bilgilerini sabit disk gibi kalıcı bir ortama alır ve gerektiği zaman yine belleğe yükleyerek çalışmasına devam edebilir. Performans olarak kötü yönde etki eden bir prosedür olmasına rağmen uygulama geliştiricilerin sanki olduğundan daha fazla bellek varmış gibi çalışmasına olanak verir. Sanal adresteki ilk 40bit bellekteki sayfa tablolarında aranmadan önce çevirim önbelleğinde (TLB) aranır ve eğer bulunamaz ise ıska geçmiş olur ve bu değer önbelleğe sayfa listesinden getirilir ve tekrar aranır. İkinci aramada ıska geçilmez ve 20bitlik fiziksel sayfa adresi bulunur ve bayt ofset ile birleştirilir böylece 32bitlik son fiziksel değer, bellek alt sistemlere verilerek fiziksel modül üzerindeki veriye erişim sağlanır (Şekil 2.6 [7]).



Şekil 2.6: Adres Dönüşümü

Eğer efektif adres blok adresleme kütüklerinde tanımlandıysa bu kütüklerde arama yapılırken bulunur ve bu işlemle birlikte paralel yapılan sayfa aramasının sonuçları dikkate alınmaz. Blok adresler bellek üzerinde bitişik alanlardır ve 128Kb'dan 256Mb'a kadar bloklar tanımlanabilirler. Genelde sayfa yapısına göre daha az değişkenlik gösterir. Örneğin işletim sistemin çekirdeği kendini blok olarak tanımlanmış bir alana yükleyebilir, böylece önbellekte (ITLB) yer tutmayarak daha az ıska alınmasına yol açar.

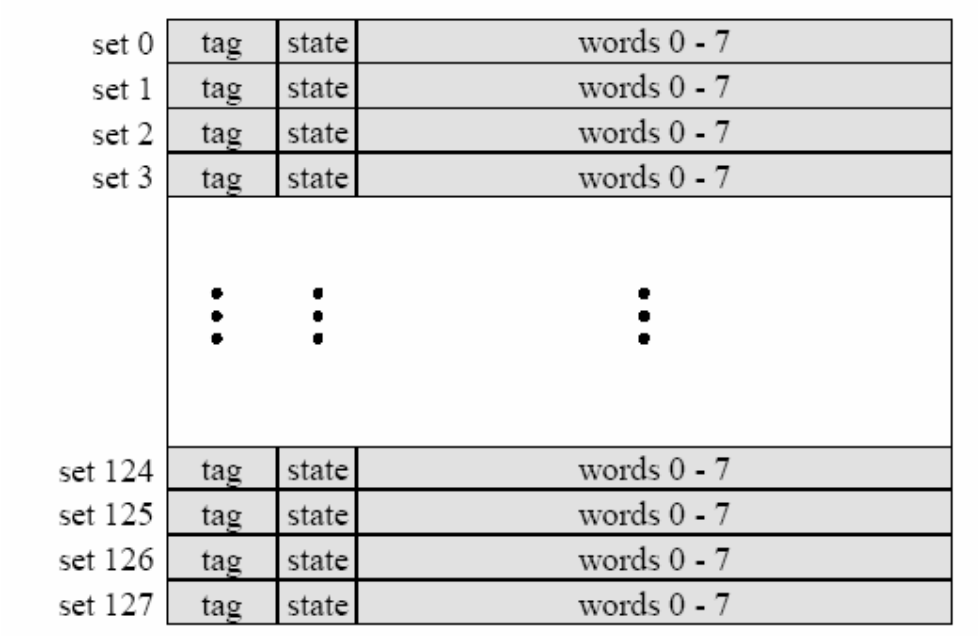
2.2.1.7 Önbellek Üniteleri



Şekil 2.7: Önbellek Yapısı

Şekil 2.7'de [7] gösterilen 603e'nin önbellek yapısı her iki veri ve komut önbelleği için 16Kb'lık bilgi saklayabilir. Önbellekler 4 yollu, küme değişimli fiziksel adreslenebilen türdedir. Her önbelleğin 128 adet kümesi, her kümede 4 adet 8 kelimelik satırı vardır. Komut önbelleği sadece okunabilir kipinde çalışır, veri önbelleği ise yazma/okuma kipinde çalışabilir. Veri önbelleği için iki çalışma şekli tanımlamak mümkündür; birincisi geri yazmalı yani önbellek ile belleğin senkronizasyonu sadece belirli zamanlarda yapılarak belli uygulamalar için performans kazanma şekli veya ikinci yöntem; direkt yazmalı ile önbellek ile belleği devamlı senkron tutma yöntemi.

Şekil 2.7'de gösterilen 4 yoldan birinin iç yapısı Şekil 2.8'de [7] gösterilmiştir.



Şekil 2.8: Önbellek Küme Yapısı

Her bir satır 3 parçadan oluşmaktadır; bir etiket, durum bilgisi ve 8 adet kelime. Etiket 20 bit genişliğindedir ve fiziksel adresin ilk 20 biti ile karşılaştırılarak satırın seçilmesine yardımcı olur. Durum bilgisi tutarsız olursa o veri kullanılamaz, tutarlı olursa o bilginin bellekteki ile aynı olduğu bilinir. Durum bilgisi değiştirilmiş olabilir, bu durumda önbellekte değiştirilmiş olan veri henüz belleğe yazılmamış demektir.

2.2.2 Komünikasyon İşlemci Modülü CPM

Komünikasyon işlemci (CP) diğer adıyla RISC mikro denetleyici, çekirdek işlemciden ayrı veri yolunda bulunan ve bu nedenle de ondan bağımsız olarak çalışabilen CPM içerisindeki 32bitlik denetleyicidir.

Komünikasyon işlemcisi her devirde bir komut işleyebilir ve komünikasyon ortamları için optimize edilmiştir. Örneğin komut kümesi CRC hesaplama gibi işlemleri barındırır. CPM içinde bulunan 24Kb'lık çift kapılı bellek, PowerPC gömülü ana çekirdek işlemci ile CP arasındaki ara yüzlerden birini oluşturur.

CPM temel olarak CPM yapılandırma kütüğü ile kontrol edilir (RCCR) ve RCCR[ERAM] değeri CPM'in komutları sadece dâhili ROM belleğinden mi yoksa hem ROM hem de çift-kapılı bellekten mi alacağını belirtir. Bu özellik Freescale ya da Arabella gibi MPC8260 için yazılım üreten firmalardan alınan mikro kodları çift yönlü belleğe yükleyerek CPM'e yeni protokolleri destekleme imkânı verir. Örneğin Arabella firmasının geliştirmiş olduğu "Expedited Fast Path" mikro kodu köprü olarak

çalışan bir Linux uygulamasında paketleri yönlendirme görevini çekirdek işlemciden alıp, bu paketleri sadece CPM içerisinde işletir, böylece 603e işlemci kullanılmadan paketler minimum bekleme ile iletilmiş olur.

CPM içerisinde bulunan Sanal DMA'lar ise her çevrebirim denetleyicisi ile 60x veya lokal yol üzerindeki bellekler arasında birer ara yüz oluşturmaktadır. FCC, SCC gibi her bir çevrebirim denetleyicisinin bir gönderme ve bir alma için sanal DMA kanalı vardır. Bu SDMA kanallarının özellikleri FCC, SCC gibi denetleyicilerin ayar kütüklerinde yer alır ve kullanıcı buraya MPC8260 el kitabına bağlı kalarak istediği uygun değerleri girer. CP seri protokolleri ve sanal DMA'ları idare etmek ile de görevlidir. CPM'in desteklediği protokolleri çevrebirim denetleyicileri bazında aşağıdaki gibi toplayabiliriz:

Üç adet tam-dubleks hızlı komünikasyon denetleyicisi (FCC) aşağıdaki protokolleri destekler;

- UTOPIA üzerinden ATM (FCC1 ve FCC2)
- IEEE802.3 / Fast Ethernet
- HDLC
- Tamamen saydam operasyon

İki adet çoklu kanal denetleyici (MCC) ise 8 TDM ara yüzüne çoklanmış 64Kbps'lik 256 adet HDLC kanalı kontrol eder.

4 adet tam-dubleks seri komünikasyon denetleyici (SCC) çeşitli protokollere destek verebilmektedir;

- IEEE802.3/Ethernet
- Yüksek seviye/senkron veri hattı denetleyici (HDLC/SDLC)
- LocalTalk
- Evrensel asenkron alıcı/gönderici (UART)
- İkili senkron komünikasyon (BISYNC)
- Tamamen saydam operasyon

İki tam-dubleks seri yönetim denetleyici (SMC) aşağıdaki protokolleri desteklemektedir;

- GCI (ISDN ara yüzü) izlemek ve C/I kanalları
- UART
- Tamamen saydam operasyon

I^2C yol denetleyici MPC8260'ın diğer I^2C aygıtları ile konuşmasına olanak verir. I^2C aygıtları arasında mikro denetleyiciler, EEPROM'lar, gerçek zaman saat aygıtları, A/D dönüştürücüler ve LCD görüntüleme cihazları gibi ürünler yer almaktadır.

Zaman dilimi atayıcı (TSA), SCC, FCC, SMC ve MCC'lerden gelen farklı verileri sekiz zamana bölünmüş çoklayıcı (TDM) ara yüzünde çeşitli protokoller ile bir araya getirmek için CPM içerisinde yer alır. TSA'nın desteklediği çeşitli protokoller aşağıda listelenmiştir;

- T1 / CEPT hatları
- T3/E3
- PCM
- ISDN
- IDL
- GCI
- Kullanıcı tanımlı ara yüzler.

CPM' de sekiz adet baud-rate üreteç (BRG) yer almaktadır. BRG'ler FCC, SCC ve SMC'ler ile kullanılabilmektedir.

Genel amaçlı dört 16 bit veya iki 32 bit zamanlayıcı bulunmaktadır. 66Mhz veri yolu hızında çalışan bir işlemci de tam bir devir 16 nano saniye ile 4 saniye arasında ayarlanabilmektedir. Her devir sonrasında istenirse kesme yaratılabilir.

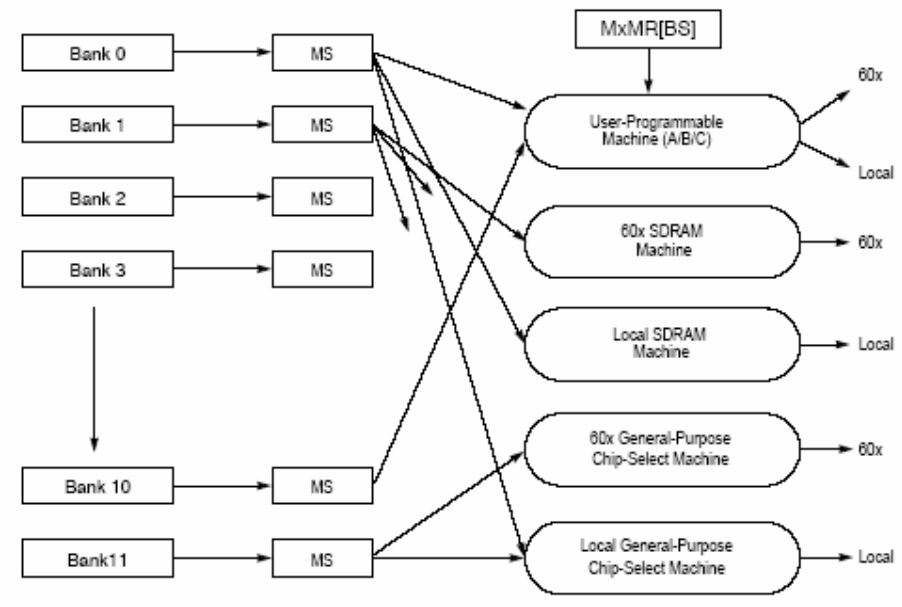
Komünikasyon işlemci, çekirdek işlemciye (603e) göre daha alt seviye komünikasyon işleri ile ilgilenir böylece ana işlemci bu seviyedeki işlemlerle meşgul ederek performansını düşürmez. Örneğin bir ağdan gelen bir paketin belleğe kadar alınması ile CPM ilgilenirken, o paketteki verilerin işlenmesi ile PowerPC çekirdek işlemci ilgilenir. İki işlemci de güçlerini koordineli bir şekilde kullanması gerekir ve bunun da ana yollarından biri CPM içerisindeki dâhili bellek alanına ulaşmaktan geçer. Her iki işlemci de bu alana ulaşabilir, denetleme bitlerini ayarlayabilir ve durum bilgilerini okuyabilir. SDMA'ler CPM tarafından komünikasyon aygıtları ve

bellek arasında veriyi taşımak için kullanılır. Fiziksel olarak iki adet SDMA bulunmaktadır. Bunlardan biri 60x yolunda diğeri ise lokal yol üzerindedir.

CPM, dört adet genel amaçlı G/Ç (A, B, C ve D) destekler. Giriş çıkışlardaki her bacak genel amaçlı bir G/Ç sinyali olarak da ayarlanabilir veya atanmış çevrebirim ara yüz sinyali olarak da konfigüre edilebilir. Bu genel amaçlı giriş çıkışlar MPC8260 işlemcisini fiziksel denetleyicilere bağlar. Örneğin LXT970 fiziksel Ethernet denetleyecisi MPC8260 ile genel amaçlı giriş çıkışlar üzerinden haberleşir. Tüm giriş çıkışlar uygulama başlamadan önce parametre kütüklerine gerekli değerler yazılarak yapılandırılırlar. Giriş çıkışlarla ilgili parametre kütüklerini ayarlamak için Motorola tarafından geliştirilmiş olan PinMux yazılımı da kullanılabilir. Bu yazılım ile geliştirilecek uygulamada kullanılacak çevre birim denetleyicileri seçildikten sonra çıktı olarak giriş çıkış yapılandırma kütüklerin değerleri alınır.

2.2.3 MPC8260 Bellek Denetleyici

MPC8260 bellek denetleyici, bir yüksek performanslı SDRAM makinesi, bir GPCM ve üç adet kullanıcı tarafından programlanabilen makineler (UPC) tarafından paylaşılan maksimum 12 bankın kontrolünden sorumludur (Şekil 2.9 [7]). Bellek denetleyici, çeşitli bellek tiplerinin karmaşıklığını iç sistemlere yansıtmadan sunmak için bir ara yüz oluşturur. SDRAM, SRAM, EPROM, Flash EPROM, toplu transfer yapılabilen RAM gibi çeşitli aygıtları desteklemektedir.



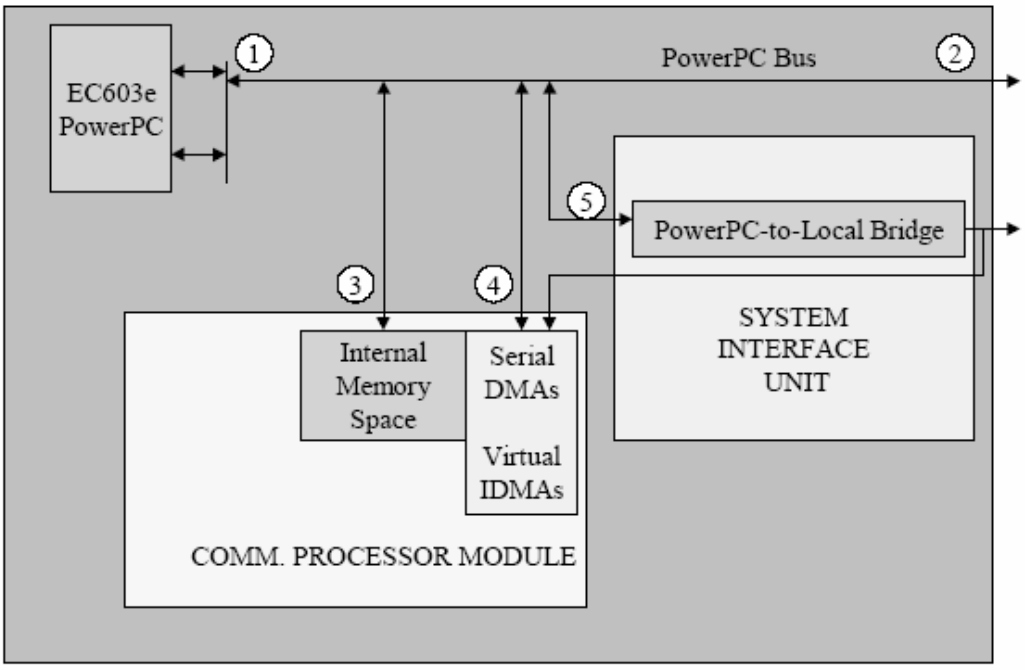
Şekil 2.9: MPC8260 Bellek Denetleyeciler

2.2.4 MPC8260 Veri Akışları

MPC8260 işlemci içerisinde ana iki yol vardır; 60x (PowerPC) yolu ve lokal yol. Her iki yol içinde birer hakem mevcuttur. 603e çekirdek işlemcinin sadece 60x yola direk ulaşımı vardır bu nedenle 603e lokal yola ulaşmak istese dahi 60x yolu kullanmak durumundadır. 603e lokal yola ulaşırkenbellek denetleyici ünitesi tarafından kontrol edilen 60x-lokal köprüsü üzerinden geçer. 603e, CPM'deki çift yönlü belleğe ulaşmak istemesi durumda da aynı yol izlenir. 603e ile CPM in aynı anda birbirlerini kesmeden çalışabilmesi için 603e'nin 60x yolda, CPM'in ise lokal yolda çalışması gerekir. 603e lokal yoldan toplu veri transferi (burst) isteğinde bulunamaz. Bu nedenden dolaylı eğer dışarıdan gelen veri üzerinde yüksek miktarda işlem yapılması gerekiyorsa lokal yol kullanılması performans kaybına yol açar. Aynı nedenden dolayı programlar 60x yol üzerinden ulaşılabilen bir bellekte bulunması gerekmektedir.

CPM ise iki türlü işlem yapabilir; birincisi çift kapılı belleğe ulaşmak isteyebilir ki bunda ne 60x ne de lokal yol kullanılır, CPM'in kendi iç yolu mevcuttur. İkincisi ise DMA tarafından başlatılan ve komünikasyon aygıtları ile bellek arasındaki veri transferi için kullanılan işlemlerdir. Örneğin CPM dış bellekte tanımlanmış olan bir bellek tanımlayıcı ulaşmak isterse yapılandırma parametrelerine bağlı olarak ya lokal yol ya da 60x yol üzerindeki hakemden istekte bulunur ve adres bellek konfigürasyon parametrelerinin bulunduğu kütüklerdeki değerlerle karşılaştırılır. Duruma göre bir bellek denetleyici ve yol seçilir. 60x hakemi seçilmiş bir durumda denetleyici içerisinden de lokal yol seçilmesi için uygulama konfigürasyon parametrelerini doğru ayarlamalıdır, aksi halde gereksiz yere 60x-lokal köprüsünü kullanmak durumunda kalır. Tersi bir durumda yani önce lokal hakem seçilmiş ve ardından da konfigürasyon kütüğünden 60x yolu geldiyse lokalden 60x'e yol olmadığı için işlem dikkate alınmaz.

MPC8260'ın içerisindeki veri akışları Şekil 2.10'da [8], yolların açıklaması ise aşağıda verilmiştir;



Şekil 2.10: MPC8260 Veri yolları

1-2 PowerPC – 60x yol: 603e tarafından önbellekte ıska durumu oluşması halinde belleğe ve bellekten veri ya da komut transferi gerçekleştirmek için kullanılır. Adresler MMU tarafından işlenir.

1-3 PowerPC – Dâhili Bellek Alanı: Kütüklere ve çift kapılı belleğe ulaşmak için kullanılır. Adresler MMU tarafından işlenir. Veriler önbellekte saklanmamalıdır.

4-2 Komünikasyon Aygıtları – PowerPC yolu: Veri transferleri ve tampon tanımlayıcılarına ulaşmak için kullanılır. Toplu transfer yapılabilir (burstable). Adresler MMU tarafından işlenmez ve veriler önbellekte saklanmamalıdır.

4-3 Komünikasyon Aygıtları - Dâhili Bellek Alanı: Veri transferleri ve tampon tanımlayıcılarına ulaşmak için kullanılır. Toplu transfer yapılamaz (single-transfer). Adresler MMU tarafından işlenmez ve veriler önbellekte saklanmamalıdır.

1-5 PowerPC – Lokal Köprü: 603'ün lokal yol üzerine erişimlerinde kullanılır. Adresler MMU tarafından işlenir. Köprü, toplu transferi desteklemez. Önbelleklerin satırları 32Kb olduğundan dolayı 603e önbellek kullanımında toplu transfere ihtiyaç duyar bu sebeple 603e'nin lokal yol üzerinde bulunan bellek erişim işlemlerinde önbellek kullanılamaz.

4-5 Komünikasyon Aygıtlar – Lokal Yol: Veri transferleri ve tampon tanımlayıcılarına ulaşmak için kullanılır. Toplu transfer yapılabilir (burstable). Adresler MMU tarafından işlenmez ve veriler önbellekte saklanmamalıdır.

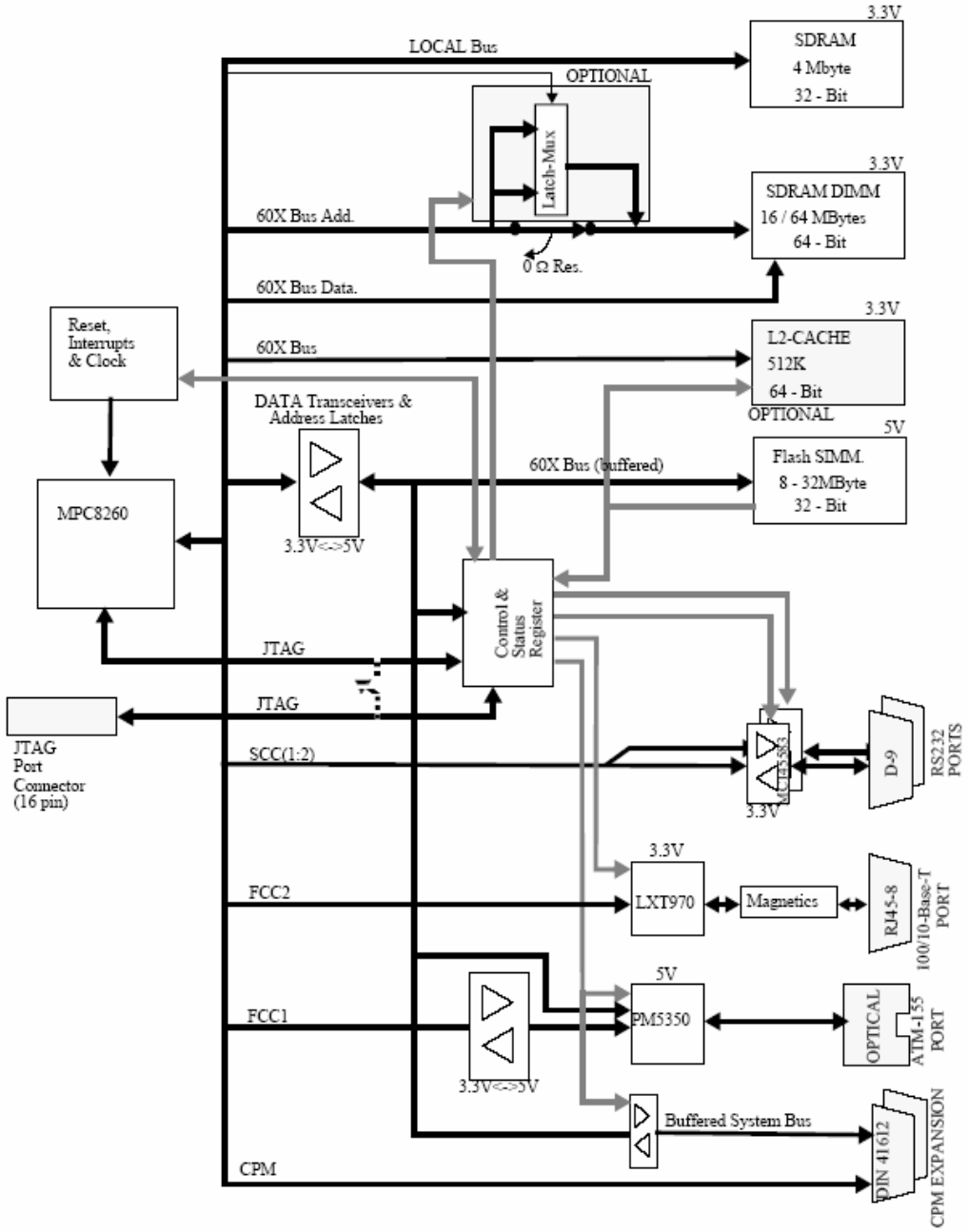
2.3 MPC8260ADS Uygulama Geliştirme Kartı

2.3.1 Özellikler

MPC8260ADS genel olarak aşağıdaki temel özelliklere sahip bir uygulama geliştirme kartıdır.

- 66MHz yol hızında çalışan bir MPC8260 İşlemci
- 60x yolu üzerinde çalışan ve SDRAM makinesi tarafından kontrol edilen 16Mb tamponlanmamış, 168 bacaklı SDRAM
- 5V ve 12V ile programlanabilen 80 bacaklı, 60x yolu üzerinde Flash SIMM bellek.
- Kart üzerine lehimlenmiş ve tamponlanmamış lokal yol üzerinde 4Mb bellek. SDRAM makinesi tarafından kontrol edilir ve 2 yongadan oluşmaktadır.
- Kart kontrol ve durum kütükleri kartın operasyonu hakkında denetleme yapar.
- Sistem açıldığında durum bilgisini Flash bellekten ya da BCSR (dip-switch) kütüğü kullanarak alma.
- FCC1 üzerinde Optical I/F kullanarak 155Mbps ATM UNI. MPC8260'a UTOPIA I/F ile bağlı ve PMC-SIEARA 5350 yongası kullanılıyor.
- Level One - LXT970 kullanılarak FCC2 üzerinde MII ile denetlenebilen 10/100-Base T Ethernet.
- SCC üzerinde 2 adet RS232 ara yüz
- Power-On Reset ve Soft/Hard reset tuşları
- Abort Tuşu
- Genişleme kartı için genişleme ara yüzü.

MPC8260ADS'nin blok diyagramı Şekil 2.11'de [8] verilmiştir.



Şekil 2.11: MPC8260ADS Blok Diyagramı

Şeklin içerisinde kalan ana yol 60x, dışarıda kalan yol ise lokal yoldur. Şekilden de görüldüğü üzere ATM, 100/10 Base-T, Rs232 çevresel birim denetleyicileri lokal yol üzerinde bulunmaktadır. 4Mb'lık SDRAM lokal yolda, 16Mb'lık SDRAM ise 60x yoldadır. Opsiyonel olarak L2 önbellek eklenebilir fakat Motorola'dan temin edilen kart üzerinde bu yer mevcut olmasına karşın kart üzerindeki noktalara lehimlenmiş

bir soket bulunmamaktadır. Dolayısıyla dışarıdan ayrıca temin edilen bir L2 önbelleğin karta monte edilmesi mümkün gözükmemektedir.

2.3.2 Kurulum

Motorola'dan dört adet temel parça temin edilmiştir; MPC8260ADS, Raven JTAG adaptör, elektrik adaptörü, TCOM genişleme kartı. Kartın temel kuruluş prosedürleri için kullanıcı el kitabına bakılmalıdır. Burada ilk dikkat edilmesi gereken noktalar VDDL değeri, kart üzerinde bulunan DS1 anahtarı ve VPP ayarıdır.

Tez çalışmasında kullanılan MPC8260 işlemci sürümü 200Mhz çekirdek PowerPC RISC işlemciye, 133MHz komünikasyon RISC işlemciye ve 66Mhz yol hızına sahiptir. Buna uygun olarak VDDL 2.5V olarak, DS1 anahtarı hız ayarlamasında kullanılır ve [OFF, ON, OFF, ON, OFF, OFF, ON, OFF] şeklinde ayarlanmalıdır. VPP ayarı ise FLASH bellek içindir. Flash bellek 12V ile daha hızlı programlanabilirken, 5V ile programlanmaya da izin vermektedir. Eğer 12V seçilir ise dışarıdan bir 12V çıkış gücü üreten güç kaynağı bağlanmalıdır fakat bu seçenek Flash koruma olarak da kullanılabilmektedir. Eğer sisteme bağlı 12V değerinde bir güç kaynağı yoksa ve VPP 12V seçeneğinde ise Flash bellek sadece okumak için kullanılabilir. Ayrı bir güç kaynağı kullanılmıyor ise Flash bellek üzerine veri depolamak için kullanıcının 5V seçeneğini tercih etmesi mecburidir.

Elektrik adaptörü 5V üretmektedir ve kart üzerinde P19 soketine takılmalıdır. TCOM genişleme kartı, ana kart üzerindeki genişleme soketlerine takılır. SDRAM ve Flash bellekler uygun yuvalara yerleştirilir. JTAG adaptörünün bir ucu kart üzerindeki JTAG sokete diğeri ise bilgisayarın paralel girişine takılır. Bilgisayarın paralel giriş çıkışı BIOS üzerinden EPP olarak ayarlanmalıdır. Bilgisayar üzerinden adaptöre erişim testi için RAVEN_TESTER [6] yazılımı kullanılarak bağlantının başarılı olduğu kontrol edilmelidir.

Kart üzerinde C ile program geliştirmek ya da Flash belleği programlamak için Code Warriror yazılımı kurulmalıdır. Kurulumu yapıldıktan sonra IDE Preferences bölümüne girilik ve "Target Setup" bölümündeki "Configuration File" için kurulumla birlikte gelen "8260_ADS_init.cfg" dosyasının yolu yazılır. Bu dosya bellek modüllerini başlangıç seviyesine getirmek için ayarlar yapar ve yine bellekler ile ilgili kütükleri günceller. MPC8260ADS ile gelen bellek modüllerinden farklı bir modül kullanılıyorsa, ilgili modülün teknik detaylarının bulunduğu doküman yardımıyla "8260_ADS_init.cfg" dosyası güncellenir.

Yine aynı ekranda Flash bellek ayarları mevcuttur. MPC8260ADS ile birlikte gelen Flash bellek modül dört adet SHARP LH28F016SCT-Z4 yongadan oluşmaktadır ve

buna uygun olarak “Personality File”, “ADS_Sharp_LHF” olarak seçilmelidir. Tez çalışmasında Flash belleği Raven JTAG adaptör ile programlandığı için “Comm IO”, “MSI COP Raven” olarak seçilmiştir. Flash bellek ayarlarının bulunduğu bölümdeki diğer alanlar, belleğe yüklenecek uygulamaya göre değişiklik gösterdiğinden dolayı bu alanlar Code Warrior’un Flash programlayıcı modülünü kullanıldığı sırada doldurulur. Motorola’dan gelen lisans dosyası kurulum dizinine kopyalanır ve MPC8260ADS, üzerinde uygulama geliştirmeye hazır hale getirilmiş olur.

2.3.3 Yazılımlar

Çalışma aşamasında çeşitli yazılımlar kullanılmıştır. Bu yazılımları kısaca aşağıdaki gibi özetleyebiliriz;

Codewarrior: Daha önceleri PowerPC işlemciler için derleyici geliştirilen Metrowerk firması 1999 yılında Motorola’nın bünyesine katılmasının ardından gömülü işlemci konusunda çalışmalarını arttırdı. Motorola mikro işlemci alanındaki işlerini Freescale firması üzerinden sürdürmeye başladığında Codewarrior halen Motorola yani Freescale harici mikro işlemcileri de destekliyordu. Bu yapı şirketin stratejilerine ters düştüğünden dolayı artık Codewarrior sadece Freescale ürünlerini desteklemektedir. Metrowerk firması da Freescale’in bünyesine katılarak artık ürünün adı önündeki Metrowerk kelimesi de kaldırılıp Freescale Codewarrior olmuştur. Bu ürünün 6,0 sürümünün bulunduğu bir CD ve lisans dosyası, kart ile birlikte Motorola firması tarafından çalışma için temin edilmiştir. Uygulamanın son sürümü ise 8,6’dır. Uygulama, temin edilen JTAG adaptör yardımıyla kart üzerinde C/C++ uygulamalar koşturabilmektedir. Codewarrior ile birlikte gelen örnek yazılımlar içerisinde gelen kesme rutinleri tez çalışmasında da kullanılmıştır. Bu rutinler assembler dilinde yazılmış ve kesme anından önceki kütükleri yığına yedekleyip, kesme sonrası geri yerleştirmektedir. Ayrıca MPC8260ADS için bellek modüleri için bellek denetleyici kütüklerine gerekli değerlerli yazan ve SDRAM belleklerini ilk çalışma için hazırlayan örnek assembler kodları da uygulama içerisinde çıkmaktadır.

Embedded Linux Development Kit (ELDK): Denx firması tarafından geliştirilen ve ücretsiz olarak internetten indirilebilen bu yazılım çapraz derleme araçlarını bir araya getirmektedir. Çapraz derleme, derleme işleminin sonucunda çıkacak olan programı derleme yapılan bilgisayarın mimarisinden farklı bir mimaride koşturulacaksa kullanılır. Tez çalışmasında kodlar Intel tabanlı Pentium bilgisayar üzerinde ELDK kullanılarak derlenmiştir. Böylece elde edilen programlar, MPC8260 PowerPC mimarisine sahip işlemci üzerinde de sorunsuz koşturulabilmiştir.

U-Boot: Açık kaynaklı olan “Das U-Boot” çeşitli gömülü işlemcileri destekleyen bir önyükleme programıdır (boot loader). Bir önyükleme programı genellikle işlemciye elektrik verildikten hemen sonra çalışmaya başlayan programdır. Genellikle masa üstü bilgisayarlardan alışık olunan lilo ve grub adlı programlar buna bir örnektir. BIOS, çeşitli kontrol ve ayarları yaptıktan sonra önyükleme programını çalıştırmaya başlar. Bu program da sistem ile ilgili bilgileri çekirdeğe geçirerek onu çalıştırır. Gömülü sistemlerde durum biraz daha karışıktır çünkü bu sistemlerde genellikle BIOS bulunmaz. Gömülü sistemlerde önyükleme programları en azından sistemdeki donanımları hazır hale getirir, bellekleri ayarlar, çekirdeğe açılış parametrelerini verir ve onu çalıştırır. U-boot bu özellikleri barındırmasının haricinde bellek üzerinde kopyalama işlemleri, çekirdek parametrelerini değiştirip Flash bellek üzerinde saklama özelliği, çekirdeği tftp gibi değişik protokoller ile ağ üzerinden yükleme özelliği gibi uygulama geliştiricinin hayatını kolaylaştıran çeşitli yeteneklere sahiptir. Bu çalışmada u-boot’un internet üzerinden son sürümü indirilip ELDK kullanılarak PowerPC mimarisine uygun olarak derlenmiştir.

Linux 2.4.25 Çekirdek: Internet üzerinden çekilen çeşitli Linux çekirdek sürümlerinin arasından 2.4.25 sürümünün MPC8260ADS desteği verdiği görülmüş ve çalışma için bu sürüm seçilmiştir. Çekirdek de yine ELDK çapraz derleyici ile derlenmiştir. Çekirdek derleme ile ilgili detaylar ileriki konularda verilmiştir.

Ağ yardımcı paketleri: Kullanıcı ve çekirdek arasında bir ara yüz oluşturan ve çekirdeğin ağ fonksiyonlarını kullanmamıza olanak veren iptables, brctl gibi çeşitli açık kodlu uygulamalardan tez çalışmasında faydalanılmıştır.

OCD commander: Masaüstü bilgisayara kurulan program hata ayıklama amaçlı olarak kullanabiliyor. Örneğin program işlemcinin o an hangi satırda bulunduğunu yani program sayaç kütüğünü (PC) gösterebiliyor, işlemciyi durdurabiliyor, kaldığı yerden başlatabiliyor. Belleğin herhangi bir bölümüne istenen değer yazılıp yine herhangi bir yerdeki değer okunabiliyor. Çalışma esnasında bu program Flash bellek modülündeki yazma korumalı alanları açabilmek için kullanılmıştır. Her ne kadar çalışmada kullanılan Codewarrior sürümü bütünleşik bir Flash programlayıcı sunsa da yazma koruma alanları ile ilgili yeterli bir ara yüze sahip olmadığı için bu işlemlerin OCD commander gibi bir yazılım kullanılarak Flash belleğe özel komutlarla yapılması gerekmektedir.

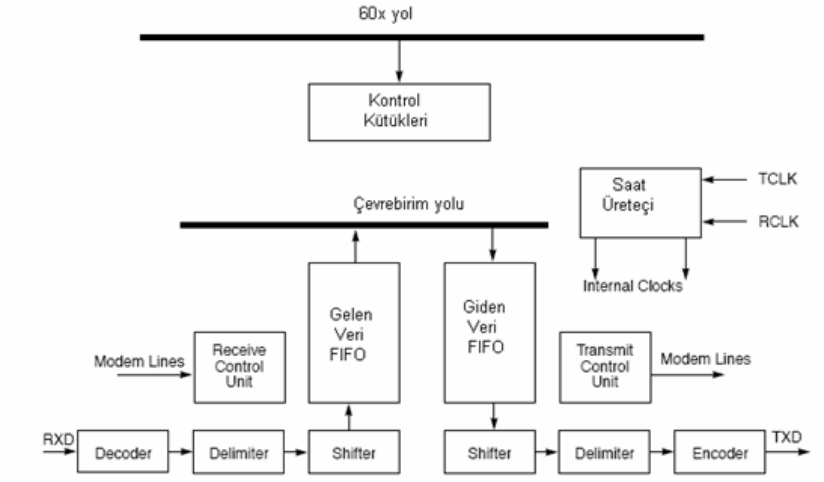
2.3.4 Codewarrior Yazılımı ile Uygulama Geliştirme

Codewarrior ile birlikte gelen ve MPC8260ADS için yazılmış örnek programlar bulunmaktadır. Fakat örnek programlar genellikle fiziksel aygıta ulaşmadan geri

dönüş (loopback) kipinde çalışmaktadır. Gerçek fiziksel aygıtlarla, mesela LXT970 Ethernet denetleyici yongasıyla çalışmak için birçok yeni fonksiyon ve ayarlama yapılmalıdır. Tahmin edilebileceğinin aksine bu ayarların ve fonksiyonların hazırlanması derin bir MPC8260 bilgisi gerektirmektedir. Çalışmanın bu bölümü üç aydan uzun bir zaman almıştır. Örnek programların yararlarının arasında bazı başlık dosyalarının da hazır olması bulunuyor. Bu başlık dosyalarının biri de MPC8260.h isimli dosyadır. MPC8260.h dosyası MPC8260'ın kaynaklarının yapısını içerir. Böylece IMMR tarafından işaret edilen ve kullanıcı el kitabının üçüncü bölümünde detaylı olarak verilen 128Kb'lık alana programcı tarafından düzenli bir şekilde erişilebilir.

2.4 MPC8260ADS Ethernet Programlama

Çalışmanın bu kısmında kartın Ethernet giriş çıkış kapıları üzerinde hâkim olabilmek için iki FCC arasında paket alışverişi yapan bir C programı geliştirilmeye çalışılmıştır. MPC8260 içerisinde Ethernet protokolünün gerçekleştirilmesi ile ilgili görevli modül CPM'dir. FCC'ler aslında birer yüksek hızda çalışmak için optimize edilmiş SCC'lerdir. FCC, 100MHz CPM saat hızında 100Mbps Ethernet, 155Mbps ATM ve 45Mbps DS-3/E3 protokollerini destekleyebilir. Her üç FCC'de birbirinden bağımsız farklı protokolü çalıştırmak üzere programlanıp köprü, yönlendirici ve ağ çıkış kapısı gibi amaçlarda kullanılabilmektedir. FCC'nin genel blok diyagramı Şekil 2.12'de [7] verilmiştir.

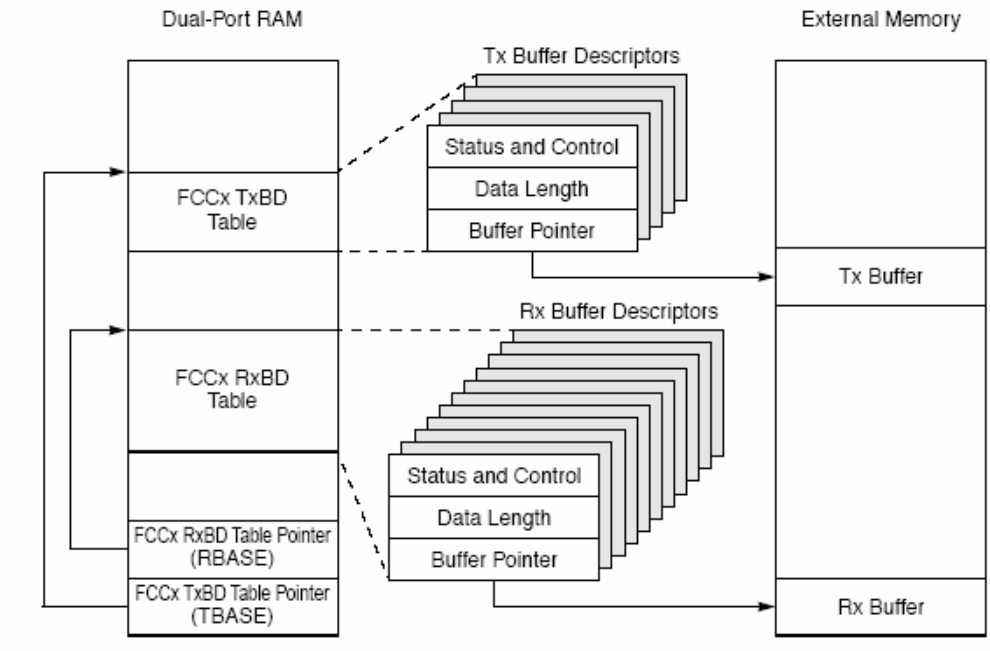


Şekil 2.12: FCC Genel Blok Diyagramı

FCC'nin hangi protokolde çalışacağını ve protokollerden bağımsız parametrelerini genel FCC kip kütüğü (GFMR) belirler. FCC'nin protokole spesifik kip kütükleri (FPSMR) ise seçilen protokole uygun değişik ayarlar yapmaya olanak verir. Ethernet

ile ilgili akış kontrol, CRC, tam dubleks, tüm paketleri alma kipi (promiscuous kip) gibi özel ayarlar FPSMR [7] kütüğü üzerine uygun değerler yazılarak yapılır.

CPM, Ethernet çerçevelerindeki bilgileri bellek üzerine sanal DMA kanallarını kullanarak yazar veya okur. Bu verileri ise tampon bölgelerde tutar. Bölgeleri, tampon bölge tanımlayıcıları belirler ve tanımlayıcılar 24kb'lık CPM'in içerisindeki çift kapılı bellekte yer almaktadır. Tanımlayıcılar, tampon bölgenin durumunu, uzunluğunu ve bellek üzerindeki yerlerini belirler. Birden büyük ve fiziksel kapasite ile sınırlı olmak şartıyla istenilen sayıda tampon bölge tanımlanabilir. Tampon bölge tanımlayıcıları CPM belleğinde dairesel olarak tanımlanmıştır. Bu demektir ki son bölge tanımlayıcısı kullanıldıktan sonra CPM tekrar ilk bölge tanımlayıcısını işler. Dairesel giriş ve çıkış tampon tanımlayıcılarının ilkinin yerini RBASE ve TBASE kütüklerindeki işaretçiler belirlenir. FCC'nin kullandığı bellek yapısı Şekil 2.13'de gösterilmiştir.



Şekil 2.13: FCC Tampon Bellek Yapısı

Tampon bölgeler tanımlayıcıları, FCC ile kullanılan protokole göre değişiklik göstermektedir. GFMR kütüğünde Ethernet protokolü seçilmiş ise RX ve TX tampon bölge tanımlayıcıları Şekil 2.14'deki [7] yapıyı alır.

Şekil 2.14'deki alanların açıklamaları ise Tablo 2.1 [7] ve Tablo 2.2'de [7] verilmiştir.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset + 0	**E**	—	**W**	**I**	L	F	—	M	BC	MC	LG	NO	SH	CR	OV	CL
Offset + 2	**Veri Uzunluğu**															
Offset + 4	**RX veri tampon işaretçisi**															
Offset + 6																

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset + 0	**R**	**PAD**	**W**	**I**	**L**	**TC**	DEF	HB	LC	RL	RC				UN	CSL
Offset + 2	**Veri Uzunluğu**															
Offset + 4	**TX veri tampon işaretçisi**															
Offset + 6																

Şekil 2.14: Tampon Bölge Tanımlayıcıları

Tablo 2.1: RX Tampon Bölge Alan Açıklamaları

Bit	İsim	Açıklama
0	E	0 Tampon bölge dolu veya çerçeve alımı bir hata yüzünden durduruldu. CP, bu tampon alanı 0 olduğu sürece veri alımı için kullanmaz. 1 Tampon bölge boş ve artık CP'ye ait. Ana işlemci bu alan 1 olduğu sürece bu bölge tanımlayıcısının herhangi bir alanına yazma işlemi yapmamalıdır.
2	W	0 RX tampon bölgeleri arasındaki son tanımlayıcı değil 1 RX tampon bölgeleri arasındaki son tanımlayıcı. Ethernet protokolünde birden fazla RX tampon bölge tanımlayıcı olması gerekmektedir.
3	I	0 Bu alan kullanıldığı zaman kesme üretmez. 1 Bu alan kullanıldıktan sonra kesme kütüğündeki FCCE[RXB] ve FCCE[RXF] bitleri 1'e çekilir. Eğer bu alanlar maskeleme kütüğünde açılmış ise ana işlemciye kesme üretilir.
4	L	Bu tampon bölge çerçevedeki son bölge olduğu zaman CP tarafından işaretlenir.
5	F	Bu tampon bölge çerçevedeki ilk bölge olduğu zaman CP tarafından işaretlenir.
7	M	0 Bu çerçeve adres tanımlandığı için alındığını gösterir. 1 Bu çerçeve promiscuous kipinde çalışıldığı için alındı fakat adres tanımlanamadı.
8	BC	Adres Broadcast tipinde olduğu zaman CP tarafından işaretlenir

9	MC	Alınan paketteki adres Multicast tipinde bir adres olduğu zaman CP tarafından işaretlenir
10	LG	Alınan çerçeve MFRL ile belirlenen maksimum çerçeve uzunluğunu geçtiği zaman bu alan CP tarafından işaretlenir
11	NO	Alınan çerçevedeki bit sayısı 8 ile bölünemediği zaman CP tarafından işaretlenir
12	SH	Alınan çerçeve MINFLR ile belirlenen minimum çerçeve uzunluğunun altında kaldığı zaman bu alan CP tarafından işaretlenir
13	CR	CRC hatası durumunda CP tarafından işaretlenir.
14	OV	Alıcı fazla yüklemesi olduğu zaman CP tarafından işaretlenir. Örneğin gelen paketlerin hızı, paketlerin işlenme hızını geçtiği ve bu nedenle boş yer kalmadığı zaman.
15	CL	Çerçeve alımında bir çarpışma yaşandığı zaman CP tarafından işaretlenir.

Tablo 2.2: TX Tampon Bölge Alan Açıklamaları

Bit	İsim	Açıklama
0	R	0 Bu tampon bölge henüz gönderilmeye hazır değil. Kullanıcı bu bölge tanımlayıcısı üzerindeki değerleri değiştirebilir. CP, bu bölgeyi gönderdikten sonra R bitini 0'a çeker. 1 Bu tampon bölge gönderilmeye hazır. Çerçeve ya gönderiliyor ya da gönderilmeyi bekliyor durumdadır. R biti 1'e kullanıcı tarafından bir kez çekildikten sonra bir daha tampon bölge üzerinde değişiklik yapılmamalıdır
1	PAD	0 MINFLR'de belirtilenden kısa çerçevelere dokunma. 1 MINFLR'de belirtilenden kısa çerçevelere MINFLR'ye eşit olana kadar PAD_PTR parametresi ile gösterilen yerdeki PAD baytları ekle.
2	W	0 TX tampon bölgeleri arasındaki son tanımlayıcı değil 1 TX tampon bölgeleri arasındaki son tanımlayıcı. Ethernet protokolünde birden fazla TX tampon bölge tanımlayıcı olması gerekmektedir.
3	I	0 Bu alan kullanıldığı zaman kesme üretmez. 1 Bu alan gönderildikten sonra kesme kütüğündeki FCCE[TXB] ve FCCE[TXF] bitleri 1'e çekilir. Eğer bu alanlar maskeleme kütüğünde açılmış ise ana işlemciye kesme üretilir

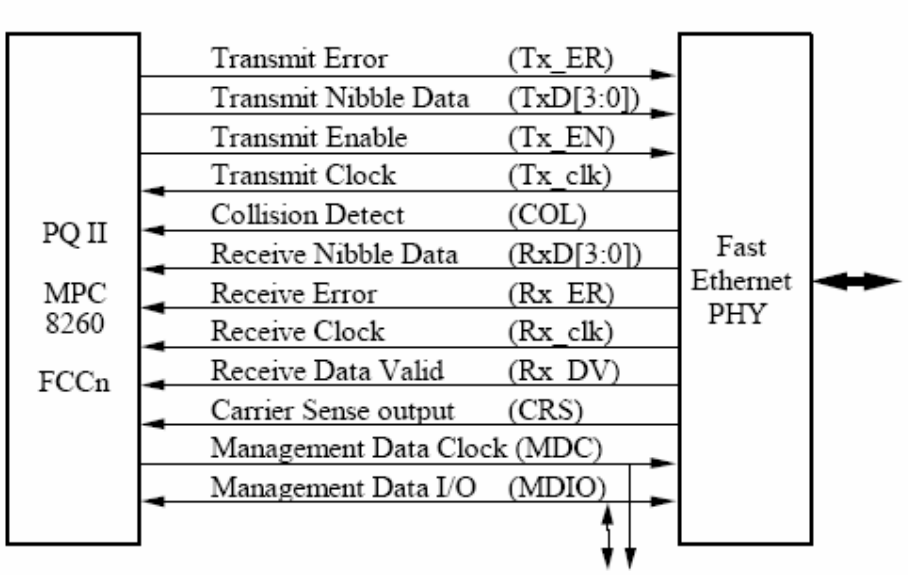
4	L	Bu tampon bölge çerçevedeki son bölge olduğu zaman kullanıcı tarafından işaretlenir.
5	TC	0 Son bayttan sonra gönderimi kes. 1 Son bayttan sonra CRC sekansını gönder.
6	DEF	Bu alan CP tarafından işaretlendiği zaman kullanıcı çarpışma harici bir ertelemeden dolayı tampon bölgedeki bilginin geç gönderildiğini anlar.
7	HB	Gönderimin ardından 40 devir içinde çarpışma girişi uyarılmadığını belirtir ve Ethernet denetleyicisi tarafından yazılır.
8	LC	FPSMR[LCW]'de belirtilen süre kadar sonra (56 veya 64 bayt) geç çarpışma meydana geldiği zaman Ethernet denetleyicisi tarafından 1'e çekilir.
9	RL	Ethernet denetleyicisi her tekrarlanan çarpışma sonrası RET_LIM'de belirtilen sayıya kadar tampon bölgeyi iletmeye çalışır ve her tekrar gönderimde RL'nin değerini bir arttırır.
10-13	RC	Tampon bölgedeki verinin kaç deneme sonrasında başarılı gönderildiğini gösterir. RC değeri 0 ise Ethernet denetleyici tek seferde başarılı gönderim yapmıştır.
14	UN	Gönderilen çerçeve birden çok tampon bölgeye yayılmış olabilir. CP tüm parçaların yüklenmesini beklemez ve eğer bir çerçeve gönderilirken, çerçeveye ait sıradaki gönderilecek tampon bölge tanımlayıcıyı ana işlemci hazırlayamamışsa CP, UN (underrun) bitini 1'e çeker.
15	CSL	Gönderim sırasında taşıyıcı hat kaybedilmişse (Carrier Sense Lost) CP, CSL bitini 1'e çeker.

FCC'yi Ethernet kipinde çalıştırmak için sırasıyla aşağıdaki adımlar izlenmelidir;

1. Paralel G/Ç kütükleri her bir FCC için Şekil 2.15'deki [7] konfigürasyonu destekleyecek şekilde ayarlanmalıdır.
2. LXT970'in saat bacakları MPC8260 üzerindeki belirli paralel G/Ç bacaklarından birine bağlıdır. CMX kütüğüne hangi FCC'nin hangi bacağa yani dolaylı olarak hangi saati kullandığı yazılmalıdır.
3. GFMR kütüğüne uygun değer yazılmalıdır fakat henüz GFMR[ENT] (enable transmit) ve GFMR[ENR] (enable receive) bitlerine dokunulmamalıdır.
4. FPSMR kütüğüne Ethernet'e spesifik ayarlar yazılmalıdır (tam-dubleks gibi)
5. Veri senkronizasyon kütüğüne (FDSR) Ethernet protokolü için 0xD555 yazılmalıdır.

6. FCC parametre RAM içerisine gerekli değerler yazılmalıdır.
7. Kesme kütüğü FCCE temizlenmelidir ve FCCM istenilen kesme değerlerine göre ayarlanmalıdır. SCPRR_H kütüğüne uygun değerler yazılarak FCC'ler arasında kesme önceliği verilebilir. SIMR_L'e uygun değer yazılarak CP'nin kesme denetleyicisine doğru kesmeler açılır.
8. CP'ye INIT TX AND RX PARAMETERS komutu verilir.
9. GFMR[ENT] (enable transmit) ve GFMR[ENR] (enable receive) değerleri 1'e çekilir.

Yukarıdaki adımlardaki uygun değerler Ek.1'deki programın tam kodunda verilmiştir ve kodun her satırının detaylı açıklanması tez konusun dışında kalacağı için bölümün kalanında sadece fonksiyonlar, işlevleri ve gerekli görüldüğü kadar detaylarından bahsedilecektir.



Şekil 2.15: MPC8260 ve Ethernet Fiziksel Sürücü Bağlantıları

Ek.1'deki programda kullanılan fonksiyonlar aşağıdaki gibidir;

void Main(void) : Tüm açılış parametreleri diğer fonksiyonlar çağırılarak ayarlanır. Kesmeler açılır ve sonsuz bir döngü içerisinde kalınarak programın sürmesi sağlanır.

void InterruptVectorInit(UWORD *, UWORD[]) : Kesme rutin kodu verilen kesme adresine kopyalanır. Burada işlemcinin FCC için kesme geldiğinde dallanacağı adres verilir. Kod işaretçisi ise startup.s dosyasında yer alan temel kesme görevini yerine getiren kodu gösterir. Bu kod kütükleri yığında sakladıktan sonra

programımızdaki ExtIntHandler fonksiyonunu çağırır. Fonksiyonun koşturulması bittikten sonra yığından kütükler tekrar geri yüklenir.

void InitBDs(void): Her iki FCC için de tampon bölgeler ayarlanır. Bellekteki yerleri belirlenir. Çıkış tamponları yazılmaya hazır, giriş tamponları ise boş olarak hazırlanır.

void InitParallelPorts(void): Paralel giriş çıkışlar, MPC8260 FCC'ler ile LXT970 fiziksel denetleyicileri birbirine kullanıcı el kitabı kullanılarak [7] ve pinmux yapılandırma seçici programının yardımıyla bağlanır.

void InterruptControlInit(void): FCC maskele kütüklerine kesmeleri geçirmesi için gerekli değerler yazılır.

void FCC1Init(void): TCOM genişleme kartı üzerindeki FCC1 için gerekli tüm parametre değerleri kullanım elkitabından yardım alınarak ayarlanır.

void FCC2Init(void): ADS kartı üzerindeki FCC1 için gerekli tüm parametre değerleri kullanım elkitabından yardım alınarak ayarlanır.

void ExtIntHandler(UWORD): Kesme geldiğinde startup.s deki kod parçasının içinden buraya yönlendirilir. Fonksiyon öncelikle kesmenin bir FCC kesmesi olup olmadığını denetler ardından da hangi FCC için kesme geldiyse ona uygun olarak processFCC1 veya processFCC2 fonksiyonlarından birini çağırır.

void processFCC1(void): Öncelikle kesmeleri temizle ve maskeleri kapatır sonrasında FCC1'in tampon tanımlayıcılarındaki boş tamponu bulur ve FCC2'nin ilk gönderilecek boş tamponuna yazar. Kesmeleri maskelerini tekrar açmadan önce yeni FCC1 için yeni kesme gelip gelmediği kontrol ettikten sonra fonksiyondan çıkar.

void processFCC2(void): processFCC1 fonksiyonun yaptığı işi FCC2 için yapar.

void enableFCC(void): MPC8260ADS kartı üzerinde bulunan BCSR kütüğüne gerekli değerler yazılarak FCC açık konuma getirilir [8].

void GP0led(UHWORD): GP0 ışıklı diyotu kapatıp açmaya yarar. Bu iş için BCSR kütüğü kullanılır [8].

void GP1led(UHWORD): GP1 ışıklı diyotu kapatıp açmaya yarar. Bu iş için BCSR kütüğü kullanılır [8]. Son iki fonksiyon kullanıcıyı bilgilendirmek için kullanılır.

void FlashGP1led(void): GP1 ışıklı diyotunu belirli periyotla kapatıp açmaya yarar. Kod içerisinde hata durumunda çağrılır.

GP0 ve GP1 ile ilgili fonksiyonların yazılımın işleyişi üzerinde direkt katkısı yoktur.

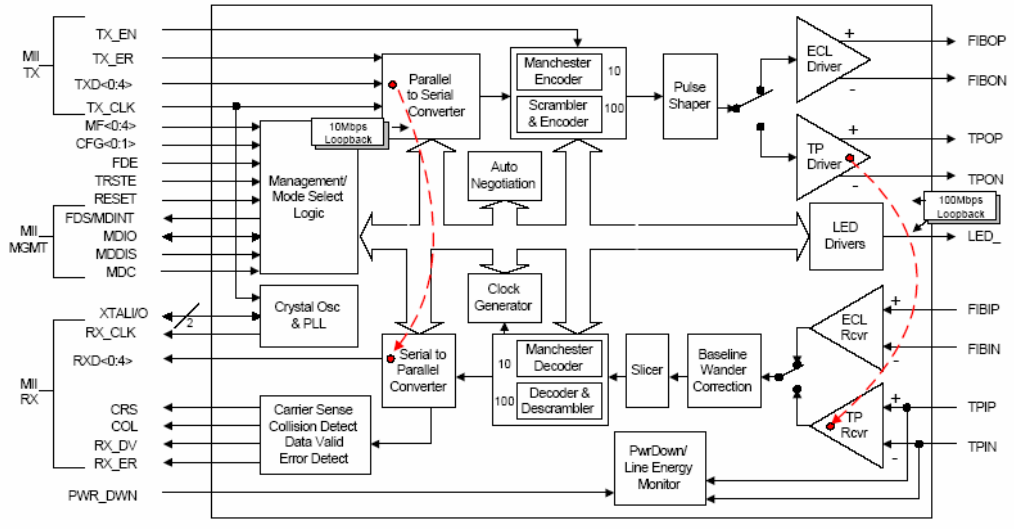
void mdioSend(UBYTE, UWORD, UHWORD): Detayı ileride tanımlanacak olan bu fonksiyon LXT970 ethernet fiziksel denetleyici yongalarının içerisindeki kütüklere veri yazmak için kullanılır. Bu işlem sırasında MII ara yüzü kullanılmaktadır.

void fethReset(void): MPC8260ADS'nin kütüğüne gerekli veriler yazılarak kart üzerindeki Ethernet çıkış sıfırlanarak ilk konuma getirilir.

void SetEEinMSR(void): Bu fonksiyon, MPC8260 işlemcinin Makine Durum Kütüğünün (MSR) ilgili bitine 1 yazılarak kesme desteğini aktif duruma getirir.

Programı geliştirmek için MPC8260ADS ve MPC8260-TCOM kartı üzerinde kullanılan LXT970 yongasını programlayan fonksiyon yazılmıştır. LXT970 yongası bir Ethernet denetleyicidir (Şekil 2.16 [23])ve OSI katmanındaki birinci, yani fiziksel katmandaki işleri yerine getirmekle görevlidir.

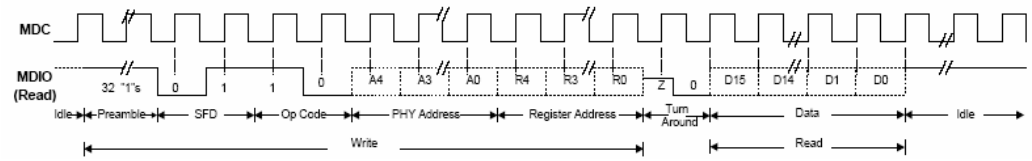
LXT970, hızlı Ethernet alıcı-vericisi olarak adlandırılmaktadır ve IEEE 802.3 standardında 10Mbps ve 100Mbps hızlarında uygulamaları desteklemektedir. LXT970 her iki hızda da tam-dubleks veri alışverişi desteklemektedir. Hız ayarı; otomatik anlaşma, paralel keşif veya el ile ayarlama kiplerini desteklemektedir.



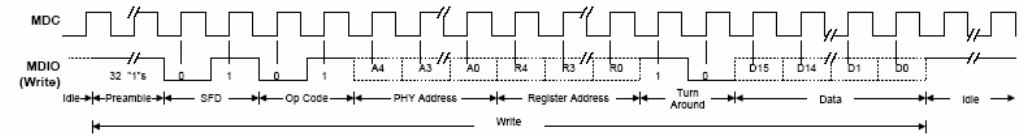
Şekil 2.16: LXT970 Ethernet Fiziksel Sürücü

MPC8260ADS ve MPC8260TCOM üzerindeki LXT970'lere konuşmak için MII ara yüzü kullanılmaktadır. MPC8260ADS üzerinde bir (FCC2) , MPC8260-TCOM üzerinde ise 2 adet (FCC1 ve FCC3) LXT970 vardır.

Tüm LXT970'ler için aynı MII yolu kullanılmaktadır. MII bir MDIO (veri) ve MDC (saat) hattından oluşmaktadır. MII ile LXT970 kontrol veya durum kütüklerinden veri okuma Şekil 2.17'de [23], veri yazma ise Şekil 2.18'de [23] gösterilmiştir.



Şekil 2.17: LXT970 MII Yolu Üzerinden Veri Okuma



Şekil 2.18: LXT970 MII Yolu Üzerinden Veri Yazma

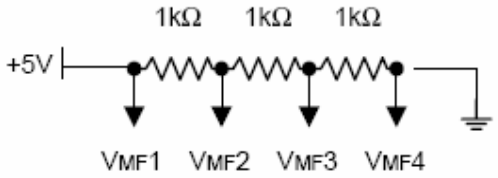
PowerPC işlemci LXT970'e mesaj gönderirken komutun içerisinde 5 bitlik bir de adres gönderir ve sadece o adrese ait LXT970 kendine ait olan veriyi işler. LXT970'lerin adresleri ADS ve TCOM kartlarda birbirinden farklı olarak ayarlanmıştır. Bu ayar LXT970 yongasının MF0, MF1, MF2, MF3 ve MF4 bacaklarına 4 farklı voltaj seviyesi verilerek ayarlanabilmektedir. Bu beş bacağa bağlı olan değerlerle LXT970'nin adresinin haricinde yonga içerisindeki bazı kontrol kütüklerinin de değerleri verilmiş olur.

MPC8260 üzerinde Ethernet programlama yapılacaksa LXT970'in saklayıcı değerleri mutlaka detaylı incelenmelidir. Çalışma sonrasında elde edilen verilerle hazırlanan Tablo 2.1'de ADS ve TCOM kartlar üzerindeki LXT970'ler için minimum düzeyde ayarlanması gereken kütükler ve değerleri verilmiştir.

Tablo 2.3: LXT970 Saklayıcı Değerleri

Saklayıcı no	Değer	Açıklama
0	0x2100	Otomatik hız seçme kipini seçer ve yongayı çalışır konuma getirir
4	0x01E1	Tüm desteklenen hız seçeneklerini sunar.
17	0x0002	LXT970'den gelecek olan kesme isteklerini aktifler
19	0x0000	DTE kipini aktifler. Ethernete bağlı LED'lerin çarpışmalarda yanacağını belirtir

Tablonun daha iyi anlaşılabilmesi için MF (Multi Function) bacaklarının alabilecekleri değerler incelenmelidir; MF, dört farklı voltaj değeri ile birbirinden bağımsız iki parametreyi belirler (Şekil 2.19 [23]). Bunlar daha önce de bahsedildiği gibi MII için fiziksel adresi ve konfigürasyon bilgileridir. Eğer LXT970 üzerindeki MDDIS bacağına yüksek potansiyel uygulanmışsa, MF bacağındaki değerler çalışma esnasında sürekli olarak konfigürasyon bilgilerini etkilemeye devam ederken MDDIS bacağına uygulanan düşük potansiyel MF'den alınan bilgilerin sadece kütüklerin açılış değerleri olacağı anlamına gelmektedir. Tez çalışmasında MDDIS bacağına düşük potansiyel verilerek belirlenen MF değerlerinin kütükler üzerindeki etkisi sadece açılış anında etkin olması sağlanmıştır.



Şekil 2.19: LXT970 MF Bacağının Alabileceği Değerler

Tablo 2.4: MF Voltaj Seviyeleri

Parametre	Sembol	Minimum	Maksimum	Unit
Voltaj Seviyesi 1	VMF1	V_{CC}-0.5	-	V
Voltaj Seviyesi 2	VMF2	(V_{CC} /2)+0.5	V_{CC} -1.2	V
Voltaj Seviyesi 3	VMF3	1.2	V_{CC} /2 -0.5	V
Voltaj Seviyesi 4	VMF4	-	0.5	V

MF'in alabileceği değerlerin sınırları Tablo 2.4'de verilmiştir. MPC8260ADS ve MPC8260-TCOM kartlarının el kitaplarındaki devre şemaları incelenmiş ve voltaj seviyeleri Tablo 2.5'deki gibi çıkarılmıştır. Kartlar üzerinde belirli voltaj seviyelerini yakalayabilmek için voltaj bölücüler kullanılmıştır. Basit bir voltaj bölücü hesabı ile ilgili MF bacağının hangi voltaj seviyesinde olduğu kolayca çıkartılabilir.

Tablo 2.5: MPC8260ADS ve TCOM Üzerinde Tespit Edilen MF Değerleri

MF	**FCC1**	**FCC2**	**FCC3**
MF0	VMF2	VMF3	VMF3
MF1	VMF4	VMF4	VMF1
MF2	VMF4	VMF4	VMF4
MF3	VMF4	VMF3	VMF4
MF4	VMF3	VMF4	VMF3

LXT970 el kitabından içerisinde yer bilgilere göre tez çalışmasında kullanılan konfigürasyonlar özetle Tablo 2.6'daki gibidir.

Tablo 2.6: LXT970 Sürücü Konfigürasyonları

Özellik	**FCC1**	**FCC2**	**FCC3**
Otomatik Hız Seçimi	Açık	Açık	Açık
Çalışma Tarzı (DTE veya Tekrarlayıcı)	DTE	DTE	DTE
TX Tarzı	4B Nibble	4B Nibble	4B Nibble
Karıştırıcı	Açık	Kapalı	Açık
Hız Bildirimleri	10/100 Tam Dubleks	10/100 Tam Dubleks	10/100 Tam Dubleks
Adres	00001	00000	00010

CPM paralel çıkışlarından iki tanesini (MDIO=PC9 ve MDCK=PC10) kullanarak LXT970 fiziksel denetleyicilere veri gönderir. CPM'in bu özelliğini kullanabilmek için daha önce bahsedilen paralel kapıları ayarlarken ilgili çıkışları denetleyicinin MDIO ve MDCK kapılarına yönlendirilmesi gerekir. Burada ilgili çıkış derken MPC8260ADS ve TCOM kartlar üzerinde kullanılan çıkışlardan bahsedilmektedir. Kendi özel kartımızı imal ediyor olsaydık CPM üzerinde MDIO ve MDCK için farklı bir çıkış kullanılabilirdi fakat kart üzerinde işlemcinin PC9 ve PC10 çıkışları sabit yollarla LXT970'in MDIO ve MDCK kapılarına bağlıdır.

MPC8260'ın MDIO ve MDCK için özel bir desteği bulunmamaktadır bu yüzden uygulama geliştiricilerin PC9 ve PC10 çıkışlarını LXT970 el kitabında belirtilen hızlarda ve gereksinim durumuna göre lojik 0 ve lojik 1' çekmeleri gerekmektedir. Saat çıkışı gerekli olan sabit bir kare dalga için belirli bir devirde dönüşümlü olarak lojik 0 ve lojik 1 yazılmıştır. Şekil 2.18'de görüldüğü üzere bir saklayıcı programlamak için önce 32 adet 1'den oluşan ön veri, arkasından 01 gönderilir, arkasından yazmanın operasyonel kodu olan 01, devamında LXT970'in MF'ler ile belirlenen 5 bitlik fiziksel adresi gönderilir. Fiziksel adresin devamında hangi kütüğe veri yazılacağını belirleyen 5 bitlik saklayıcı adresi verilir. Saklayıcı adresinden sonra 01 yazılır ve bunu kütüğe yazılacak veri takip eder. LXT970 üzerindeki kütükler 16bitlik olduğu için veri de 16bit genişliğindedir. Sonuç olarak bir saklayıcı programlamak için 64 bitlik veri gönderilmesi gerekmektedir. Bu işi üstelenecek bir fonksiyon geliştirilmiştir. Fonksiyon giriş değeri olarak fiziksel adres, saklayıcı adresi

ve veri almaktadır. LXT970'in kütüklerden MII ile okuma desteği bu tezde kullanılmamıştır. Yazma fonksiyonu ile ilgili fonksiyon aşağıdaki gibidir;

```
void mdioSend(UBYTE REGADDR,UWORD PHYADDR,UHWORD DATA){

UBYTE i;
UWORD j;
UWORD mask = 1<<15;
UBYTE SFDOP=0x05;
UBYTE TA = 0x02;
UWORD TEMP=0;
UWORD bak=0;
UWORD wait = 0xFF;// Clock Frequency should be lower than 2.5Mhz
UWORD mdio = 0x00400000; //PC9
UWORD mdc  = 0x00200000; //PC10



// 32 adet 1'i arka arkaya gönder

GP1led(ON);

for ( i= 0;i<32 ;i ++){

        IMM->io_regs[PORTC].pdat |= mdio;   // mdio üzerine 1 yaz
                // zaman üzerinde kare dalga yarat
        for( j = 0;j < wait; j++ ); // bekle
        IMM->io_regs[PORTC].pdat |= mdc;  // MDC'yi 1 yap
        for( j = 0;j < wait ; j++ );         // bekle
        IMM->io_regs[PORTC].pdat &= ~mdc;  // MDC'yi 0 yap
}

// geri kalan veriyi hazırla
TEMP=(TA)|(REGADDR<<2)|(PHYADDR<<7)|(SFDOP<<12);

for ( i= 0;i<16 ;i ++){
if(mask & TEMP){    // soldan ilk bit'e bak
 IMM->io_regs[PORTC].pdat |= mdio; //ilgili bit 1 ise mdio üzerine 1 yaz
 }
else {
 IMM->io_regs[PORTC].pdat &= ~mdio; //ilgili bit 0 ise mdio üzerine 0 yaz
 //GP1led(OFF);
 }

// zaman üzerinde kare dalga yarat
for( j = 0;j < wait ; j++ ); //bekle
IMM->io_regs[PORTC].pdat |= mdc; // MDC'yi 1 yap
for( j = 0;j < wait ; j++ );
IMM->io_regs[PORTC].pdat &= ~mdc; // MDC'yi 0 yap

TEMP = (UWORD) TEMP << 1; en soldaki bit'i düşürerek veriyi bir sola kaydır.
        }
```

```
TEMP=DATA;   // göndereceğimiz bilgi artık kütüklere yazacağımız veri
for ( i= 0;i<16 ;i ++){

// en soldaki biti gönder
if(mask & TEMP){
 IMM->io_regs[PORTC].pdat |= mdio;
 }
 else {
 IMM->io_regs[PORTC].pdat &= ~mdio;
 }

// zaman üzerinde kare dalga yarat
for( j = 0;j < wait ; j++ );
IMM->io_regs[PORTC].pdat |= mdc;
for( j = 0;j < wait ; j++ );
IMM->io_regs[PORTC].pdat &= ~mdc;
TEMP = (UWORD) TEMP << 1;  // en soldaki bit'i düşürerek veriyi bir sola kaydır
                                }

 }
```

Tezin bu aşamasının sonunda programın çalıştırılmasıyla birlikte MPC8260ADS saydam bir paket taşıyıcı olarak görev yaptığı gözlemlenmiştir. Bu yapı tekrarlarcılardaki gibi zayıflayan sinyalleri güçlendirmek için kullanılabilir.

3. MPC8260ADS ÜZERİNDE LINUX İŞLETİM SİSTEMİ KURULUMU

MPC8260ADS üzerinde hazır bir işletim sistemi olmadan CPM içerisindeki kendi mikro kodları ile TCP/IP gerçekleştiremez. Kabul edilebilir bir süre içerisinde kartın tüm kaynaklarını kontrol etmenin üzerine sıfırdan bir TCP/IP uygulaması gerçekleştirmek için bir proje grubuna ihtiyaç vardır. Bu süreyi kısaltmak ve açık kodlu diğer tüm yazılımlardan da faydalanabilmek için kart üzerinde tezin bu bölümünde Linux koşturulması amaçlanmıştır.

Öncelikle bir Linux çekirdeğine ihtiyaç vardır ve bazı çekirdek sürümlerinin MPC8260 desteği bulunmaktadır. Birçok sürüm internetten indirilerek denenmiş ve kart için en uygun olarak aralarından 2.4.25 sürümü seçilmiştir. MPC8260, PowerPC™ mimarisinde bir işlemci olduğu için ya aynı yapıda bir işlemci üzerinde çekirdek derlenmelidir ya da çapraz derleme programları kullanılmalıdır. Tezin bu bölümünde ELDK çapraz derleyici kullanılarak PowerPC™ uyumlu bir çekirdek yaratılmıştır. Normal bilgisayarlarda ilk açılışta BIOS devreye girer ve arkasından da lilo ve grub gibi diskin açılış bölümü diye tabir edilen kısmında yer alan açılış programları yer alır. Bu tip programlar çekirdeği çalıştıran ve ona gerekli parametreleri ileten yazılımlardır. MPC8260ADS üzerinde bir BIOS bulunmadığı için bu programlar MPC8260ADS için kullanılamazlar. UBoot adlı program ise her iki görevi de yerine getirmektedir. Internet'ten en son sürüm Uboot çekilip derlenmiş ve daha sonra derlencek çekirdeğe başlangıç parametreleri vererek koşturmak için kullanılmıştır. BusyBox isimli yazılım ise bize Linux'un en temel komutlarını içeren bir ext2 dosya sistemi imajı yaratmamıza yardımcı olmuştur. Temelde bu üç imaj dosya uygun şekilde hazırlandıktan sonra karta uygulanması ile birlikte sistem temel olarak açılıp, konsol üzerinden basit işlemler yerine getirilebilecek konuma gelmektedir. TCP ile ilgili çeşitli yazılımlar yine internet üzerinden indirilip çapraz derleme yöntemiyle dosya sistemi imajına eklenmiştir ve kart bir yönlendirici, trafik düzenleyici gibi amaçlar için kullanılmıştır.

Bu adımlar için basit ve düz bir yol ne yazık ki yer almamaktadır ve özellikle çekirdek üzerindeki sürücüler için bazı yamalar yapmak gerekmektedir. MPC8260ADS ve TCOM genişleme kartı üzerine Linux yüklemek ve çeşitli uygulamaları çalıştırmak için izlenen adımlar bu bölümde detaylı olarak yer almıştır.

3.1.1 Çapraz Derleme

PowerPC™ uyumlu çalıştırabilir kodlar üretmek için uygun derleyicilere ihtiyaç bulunmaktadır. Eğer derleyicinin çalışacağı platform farklı ise bu durumda yapılan işleme çapraz derleme adı verilir. Tez çalışması süresince Intel tabanlı bir bilgisayar ve Linux işletim sistemi üzerinde derleme işlemleri gerçekleştirilmiştir bu yüzden MPC8260ADS için ELDK isimli uygulama geliştirme yazılımı kullanılmıştır. ELDK, gömülü işlemciler için çapraz uyguluma geliştirme araçlarını bir araya getiren bir pakettir. Firmanın sitesinde [9] aşağıdaki sistemler üzerinde sorunsuz çalıştığı belirtilmiştir;

- Fedora Core 1, 2, 3, 4
- Red Hat Linux 7.3, 8.0, 9
- SuSE Linux 8.x, 9.0, 9.1, 9.2, 9.3
- Debian 3.0 (Woody) and testing (Sarge)
- Ubuntu 4.10, 5.04
- FreeBSD 5.0

Bunların dışında ELDK kullanıcıları aşağıdaki sistemlerde ise sorun yaşamadan yazılımdan faydalanabildiklerini belirtmişlerdir;

- Suse Linux 7.2, 7.3
- Mandrake 8.2
- Slackware 8.1beta2
- Gentoo Linux 1.4_rc2

Tez çalışması sırasında ise Red Hat Enterprise Linux 3.0 kullanılmıştır ve herhangi bir sorun yaşanmamıştır. ELDK aşağıdaki mimariler için kod üretebilmektedir [11]

- ppc_4xx = AMCC 4xx processors without FPU
- ppc_4xxFP = AMCC 4xx processors with FPU
- ppc_6xx = PowerPC processors based on the 603e core (This includes support for MPC5xxx, 7xx, and 82xx processors).
- ppc_74xx = 74xx processors

- ppc_8xx = MPC8xx processors
- ppc_85xx = MPC85xx processors

ELDK uygulaması iki farklı mimariye özgü olarak iki ISO imaj dosyasında sunulmaktadır. Biri AMCC 4xx ailesi için diğeri ise Freescale 8xx, 6xx, 74xx ve 85xx aileleri için hazırlanmıştır. Linux komut satırından uygulamayı çekmek için aşağıdaki komut çalıştırılır;

```
wget ftp://ftp.sunet.se/pub/Linux/distributions/eldk/4.0/ppc-linux-x86/iso/ppc-2006-01-11_freescale.iso
```

iso dosyası boş cd'ye yazmadan sabit disk üzerinde aşağıdaki gibi açılabilir;

```
mkdir /cdrom
mount ppc-2006-01-11_freescale.iso /cdrom -t iso9660 -o loop
cd /cdrom
```

Kurulumu gerçekleştirmek için aşağıdaki yapı kullanılır;

./install [-d <dir>] [<cpu_family1>] [<cpu_family2>] ...

MPC8260, ppc_6xx ailesinde bulunduğu için aşağıdaki komut verilerek MPC8260 destekli ELDK, kök dizini üzerindeki eldk dizinine kurulmuş olur;

```
mkdir /eldk
./install –d /eldk ppc_6xx
```

Bundan sonra diğer programların kendilerini derlerken kullanacağı ortam değişkenini atamak gerekmektedir;

```
PATH=$PATH:/eldk/usr/bin:/eldk/bin

export CROSS_COMPILE=ppc_6xx-
```

artık çapraz derleyici hazırdır. Örneğin C ile hello_world.c isimli bir kaynak dosyası hazırlanmış ise bu dosya aşağıdaki komut ile MPC8260 işlemciye uygun kod üretilebilir;

${CROSS_COMPILE}gcc -o hello_world hello_world.c

3.1.2 Boot Loader (Das UBoot) Kurulumu

UBoot, açık kaynak kodlu ve bir çok gömülü işlemciyi destekleyen sistem açılış programıdır. UBoot, MPC8260ADS üzerinde kişisel bilgisayarlarda bulunan BIOS'un ve sabit diskimizin başında bulunan lilo, grub gibi programların işlevlerini bir araya getirerek çalışmaktadır.

Boot Loader uygulamaları sistemi ilk değer atamalarını yaparak hazır hale getirdikten sonra çekirdeği sisteme özgü parametreleri vererek çalıştırır. Örneğin bu parametrelerden biri kök dizininin hangi fiziksel sabit sürücü üzerinde olduğunu belirtebilir.

Gömülü sistemlerde bu programlar kişisel bilgisayarlardaki BIOS'un olmaması nedeniyle daha karmaşık olmaktadır. Çekirdeği çalıştırmadan önce UBoot düşük seviyede bellekleri, işlemciyi ve çeşitli modüllerinin ilk ayarlarını yapar ve bu doğal olarak sistemden sisteme göre de değişmektedir. UBoot bu temel ihtiyaçların dışında kullanıncaya bir komut satırı ve bu komut satırını kullanarak yapılabilecek çeşitli işlemler sunmaktadır. Başlıca işlevleri aşağıdakiler olarak sıralanabilir;

- Bellek üzerindeki herhangi bir adresten veri okuma ve herhangi bir adrese veri yazma.
- Seri port veya Ethernet üzerinden imaj dosya yükleyebilme
- RAM üzerinden Flash bellek üzerine imaj dosyaları programlama

UBoot daha önceleri 8xxROM olarak bilinen ve 8xx PowerPC mimarisindeki sistemler için boot loader uygulamasının geliştirilmiş halidir ve artık PowerPC, ARM, XScale, MIPS, Coldfire, NIOS, Microblaze ve x86 işlemcileri de destekleyebilmektedir [11].

UBoot'u [12] internetten internetten indirdikten sonra ELDK çapraz derleyicinin yardımıyla MPC8260 işlemcinin desteklediği mimariye uygun olarak derlemek gerekmektedir. Bunun için aşağıdaki adımlar uygulanmalıdır;

```
tar –xzvf u-boot.tar.bz2

cd u-boot

make mpc8260ads_config

export CROSS_COMPILE=ppc_6xx_
```

Uboot kullanıcı ile iletişimini RS232 kapısı üzerinden yapmaktadır. Bu iletişim farklı hızlarda sağlanabilir. Tez çalışması sırasında 38400 bps hızı tercih edilmiştir. Bu

ayarı yapabilmek için uboot klasörünün altındaki include/configs/MPC8260ADS.h dosyasında aşağıdaki değişiklikler yapılmalıdır;

```
#define CONFIG_BAUDRATE         38400
#define CFG_BAUDRATE_TABLE      { 9600, 19200, 38400 }
```

Bu değerler için daha sonra değişiklik yapmak istersek uboot konsolu bunlara imkan vermektedir. Son olarak uboot klasöründeyken "make" komutu verilerek PowerPC mimarisine uygun imaj dosyası oluşturulur. Bu dosya bir sonraki adımda codewarrior ile gelen flash programlayıcısını kullanılarak flash belleğe yüklenir. Böylece artık MPC8260ADS kartına elektrik gelmesiyle birlikte uboot uygulaması çalışmaya başlayacak ve bizim yerimize başlangıçtaki bir çok düşük seviyedeki sistem ayarını yapacaktır. Çekirdek derlendikten sonra uboot, çekirdeği flash belleğe kopyalamada, parametreleri ayarlamada kullanılacaktır fakat detayları bu bölümde değil ilgili bölümlerin içerisinde verilecektir.

MPC8260ADS üzerinde UBoot'un ilk çıktıları aşağıdaki gibidir;

```
U-Boot 1.1.4 (Oct 17 2006 - 22:25:52)

MPC8260 Reset Status: Check Stop, External Soft, External Hard

MPC8260 Clock Configuration
 - Bus-to-Core Mult 3x, VCO Div 2, 60x Bus Freq  33-100, Core Freq 100-300
 - dfbrg 1, corecnf 0x08, busdf 3, cpmdf 1, plldf 0, pllmf 1, pcidf 0
 - vco_out  264000000, scc_clk   66000000, brg_clk   16500000
 - cpu_clk  198000000, cpm_clk  132000000, bus_clk   66000000

CPU:   MPC8260 (HiP3 Rev 01, Mask A.1 1K22A-XC) at 198 MHz
Board: Motorola MPC8260ADS
I2C:   ready
DRAM:  16 MB
FLASH:  8 MB
In:    serial
Out:   serial
Err:   serial
Net:   FCC2 ETHERNET
Hit any key to stop autoboot:  0
```

Daha sornaki bölümlerde hazırlanacak bir imaj dosyasını tftp komut ile bellek üzerinde istediğimiz bir alana kopyabiliriz. Örneğin tftp f00000 vmlinux komutu ile

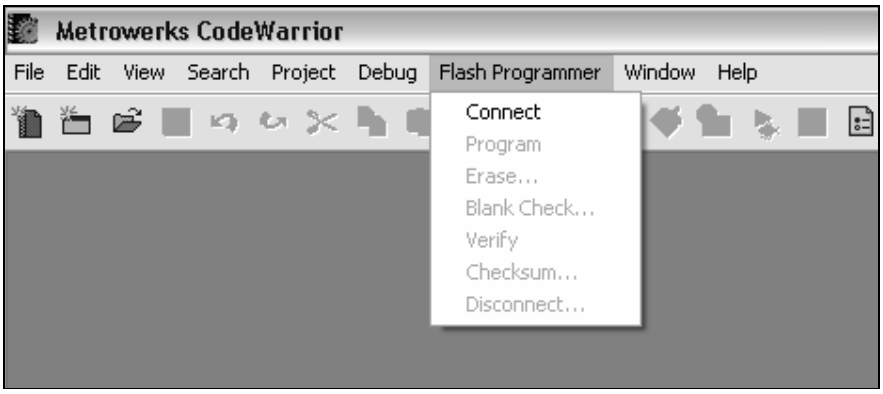
tftp ile paylaştırdığımız vmlinux dosyasını bellek üzerinde 0xf00000 adresine kopyalabilirz. Uboot dosyayı hangi bilgisayar üzerinden çekeceğini ortam değişkenlerinden anlar. Bunun için ipaddr, netmask, ethaddr, serverip değerlerinin ayarlanması yeterlidir. Örnek olarak aşağıdaki komutlar kullanılabilir;

```
setenv ipaddr 192.168.1.10
setenv netmask 255.255.255.0
setenv ethaddr 00:00:11:aa:bb:cc
setenv serverip 192.168.1.2
```

Bu ayarlar ile birlikte mpc8260ads 192.168.1.2 IP adresli bilgisayara bağlantı kurmaya çalışacak ve tftp protokolü ile birlikte vmlinux dosyasını talep edecektir. Çekirdek ve dosya sisteminin imaj dosyaları bu şekilde belleğe kopyalanır. Ana bilgisayar üzerinde tftp sunucusu kurmak için linux.com sitesindeki makeleden yardım alınabilir [13]. Eğer imaj dosyaları son haline getirilmiş ise bunları MPC8260ADS üzerindeki Flash bellek üzerinde kopyalamak sisteme bağımsız bir şekilde çalışma imkanını sağlar. Bunun için "cp.b " komutunu kullanılabilir. Kullanımı gayet basit olan bu komuta parametre olarak önce kaynak adresi, ardından da hedef adresi ve hex formatında, byte cinsinden uzunluk bilgisi verilir.

3.1.3 Codewarrior ile Flash Programlama

Değişik firmalar tarafından Flash bellekleri programlamak için çeşitli uygulamalar geliştirilmiştir fakat bunların çoğu ücretli yazılımlardır. Flash bellek programlama işlevleri ileride uboot içerisinden yapılacağı için ilk etapta temel ve basit bir programlayıcı yeterli olacaktır. Codewarrior içerisinde Raven JTAG adaptörünü destekleyen bir Flash bellek programlayıcı bulunmaktadır (Şekil 3.1).



Şekil 3.1: CodeWarrior Flash Bellek Programlayıcısı

Codewarrior'un programlayıcısı ile Flash belleği programlama, silme, doğrulama gibi işlevleri yerine getirilebilir fakat bu program belleğin yazma korumalı alanları üzerine kilidi açamadığı için bazı alanlara yazamayabilir. Konunun daha net anlaşılması için bu kısımda Flash bellek hakkında daha detaylı bilgi verilmiştir.

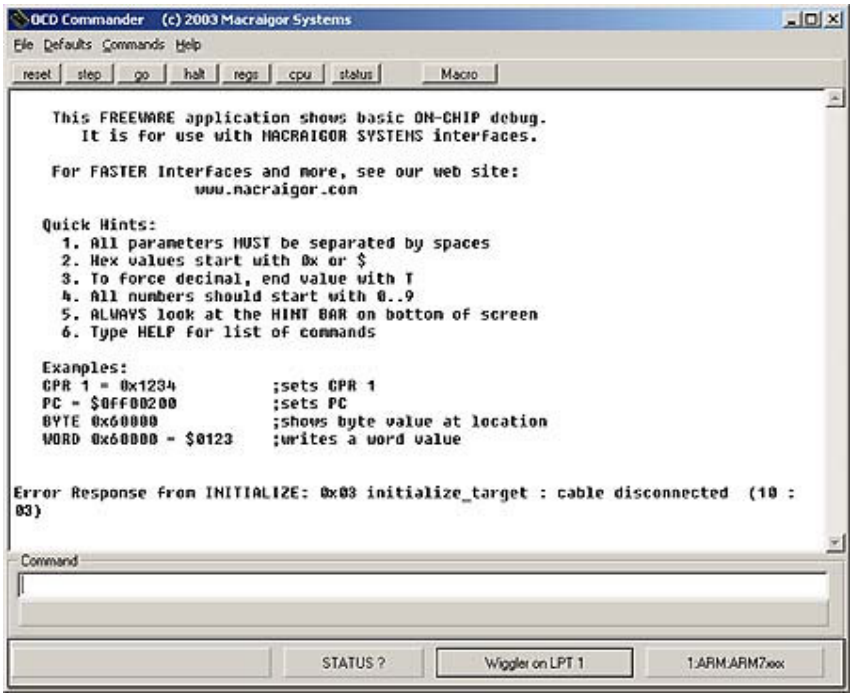
MPC8260ADS üzerinde bir adet Flash bellek modülü yer almaktadır. Modülün üzerinde dört adet, her biri 8 bitlik genişliğe sahip LH28F016SCT-Z4 Sharp Flash bellek yongaları bulunmaktadır. Her biri 16 megabit (2 megabayt) veri saklayabilen yongalar kullanıcıya toplamda 8 megabaytlık bir veri alanı sunmaktadır. Flash bellek lokal yol üzerindedir ve 32bitlik bir veri genişliği sunmaktadır ve bunu da 4 adet LH28F016SCT-Z4 yongayı yan yana getirerek gerçekleştirmektedir. Örneğin Tablo 3.1'de görüldüğü gibi 0xFF000000 adresinden 32bitlik bir veri okuduğumuz zaman her dört yonganın başlangıç adresindeki verilerin yan yana getirilmiş halini elde ederiz.

Tablo 3.1: Flash Bellek Adresleme

	YONGA1 (FF000000)	YONGA2 (FF000001)	YONGA3 (FF000002)	YONGA4 (FF000003)
FF000000	8BIT	8BIT	8BIT	8BIT

Her yonga üzerinde 32 adet 64kilobaytlık alana sahip bloklar bulunmaktadır. Bu bloklar üzerine yazma koruma konulabilmektedir. Uboot kendini ilk yüklediği zaman kendini yüklediği 2 adet alana koruma koymaktadır. Böylece yanlışlıkla imaj dosyaları uboot imajının üzerine yazılması engellenmiş oluyor. Fakat yeni bir uboot imajı atılmak istenirse bu alanların sıfırlanması gerekir ve bu durumda Codewarrior programı yetersiz kalmaktadır. Raven JTAG adaptörümüzün üretici firması Macraigor firmasının ücretsiz bir yazılımı olan OCD commander ile Flash bellek üzerindeki koruma alanlarını kaldırılabilmektedir [14].

OCD commander içinden örnek bir görüntü Şekil 3.2'de verilmiştir.



Şekil 3.2: OCD Commander

LH28F016SCT-Z4 belleğini programlamak için desteklenen komutlar Tablo 3.2'de verilmiştir.

Tablo 3.2: LH28F016SCT-Z4 Komutlar

KOMUT	Gerekli devir	İLK DEVİR			İKİNCİ DEVİR		
		operasyon	adres	veri	operasyon	adres	veri
Sıra oku / sıfırla	1	Yazma	X	FFH			
Tanımlayıcı kodları oku	≥2	Yazma	X	90H	Okuma	IA	ID
Durum kütüğünü oku	2	Yazma	X	70H	Okuma	X	SRD
Durum kütüğünü sıfırlar	1	Yazma	X	50H			
Blok sil	2	Yazma	BA	20H	Yazma	BA	D0H
Bayt yaz	2	Yazma	WA	40H veya 10H	Yazma	WA	WD
Blok sil ve byte yaz duraklat	1	Yazma	X	B0H			
Blok sil ve byte yaz devam ettir	1	Yazma	X	D0H			
Blok kilit bitini 1 yap	2	Yazma	BA	60H	Yazma	BA	01H
Master kilit bitini 1 yap	2	Yazma	X	60H	Yazma	X	F1H
Blok kilit bitlerini temizle	2	Yazma	X	60H	Yazma	X	D0H

X=Aygıt üzerindeki herhangi bir adres
IA=Tanımlayıcı kodu adresi
BA=Silinen veya kilitlenen blok üzerinde bir adres
WA=Yazılacak verinin adresi
SRD=Durum kütüğünden kounan veri
WD=WA adresine yazılacak veri
ID=Tanımlayıcı kodundan okunan veri

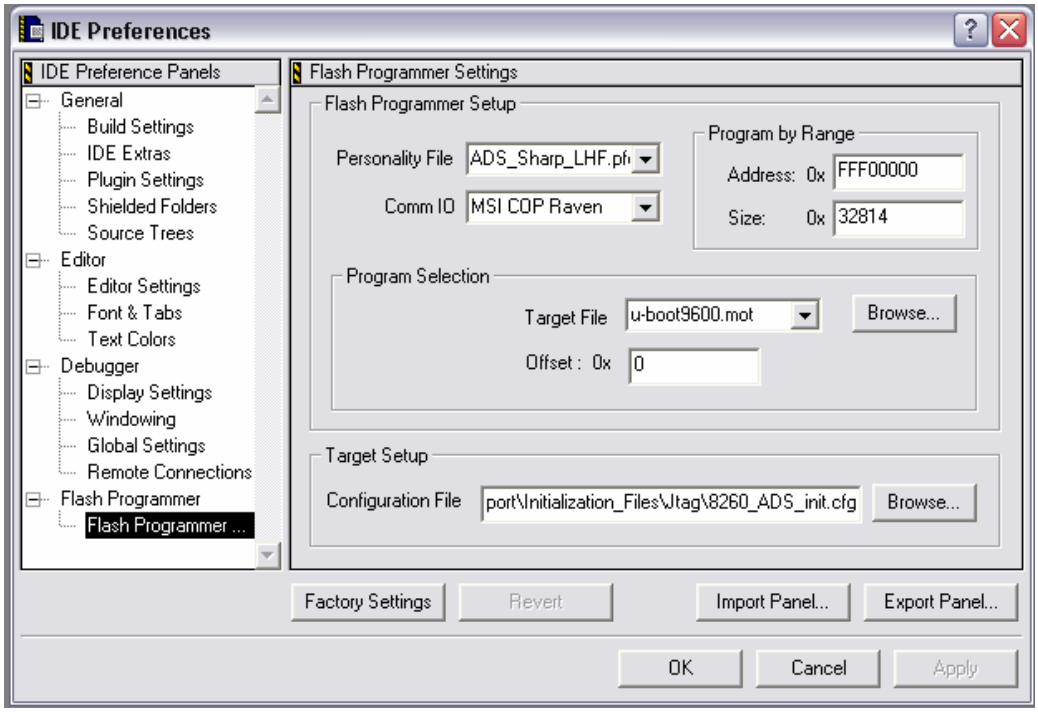
Tablo 3.2'ye göre istenilen bir bloğu kilitlemek için blok içerisinde bir adrese 60H ve 01H değerleri arka arkaya yazılmalıdır. Doğal olarak kullanıcı penceresinden bakıldığında bir adres tek bir blok gibi gözükmesine rağmen Flash bellek modülü üzerindeki dört ayrı yongada birbirinden bağımsız dört ayrı blok mevcuttur. Bu durumda anlamlı olan dört yongadaki aynı adreslerdeki kilit bitlerini senkron tutmaktır. Bunun için ise Tablo 3.2'deki komutlardaki veriler kendini tekrarlayarak dört kez yazılmalıdır. Örneğin bir bloktaki kilit bitini temizlemek için, blok içerisinde bir adrese (genellikle blok başlangıç adresi seçilir) önce 60606060H sonrasında ise D0D0D0D0H yazılır.

Eğer Uboot kurulumu başarısız olduysa veya Flash bellek üzerinde daha önceden programlı ve kilitli bloklar mevcut ise yeni bir uboot imaj programlama işlemi öncesinde bu blokların kilit bitlerinin temizlenmesi gerekir. Sonrasında yine Flash bellek komutları veya Codewarrior uygulaması içerisindeki bütünleşik Flash programlayıcı ile birlikte içerideki veri temizlenebilir.

Daha önceki bölümden hazırlanmış olan uboot.mot doyası Codewarrior ile Flash belleğe programlanması gerekmektedir. Bu dosya S19 formatındadır ve text tabanlıdır. Dosyanın her satırı belirli kolonlara ayrılmıştır. Temel olarak hangi adreste hangi verinin saklanacağı belirtilir. Dosyanın içerisinden kolaylıkla hangi adresler arasında kayıt yapılacağı anlaşılabilir. Tez çalışmasında elde edilen s19 çıktısında ilk satır 0xFFF00000, son satır ise 0xFFF32811 olduğu gözükmektedir. Bu demektir ki yaklaşık 202kb'lık bir alana ihtiyaç duyulmaktadır. Normalde bir blok (4*64Kb) ihtiyaç varken Uboot konfigürasyon bilgileri gibi kendine özel verileri korumak için ilk açılışta kendi bloğu ile birlikte bir sonraki bloğa da kilit koymaktadır. Flash bellek üzerindeki blokları ve kilit durumlarını görmek için uboot açıldıktan sonra "flinfo" komutu verilebilir;

```
Evren=> flinfo
Bank # 1: Sharp 28F016SC (16 Mbit, 32 x 64K)
  Size: 8 MB in 32 Sectors
  Sector Start Addresses:
    FF800000      FF840000      FF880000      FF8C0000      FF900000
    FF940000      FF980000      FF9C0000      FFA00000      FFA40000
    FFA80000      FFAC0000      FFB00000      FFB40000      FFB80000
    FFBC0000      FFC00000      FFC40000      FFC80000      FFCC0000
    FFD00000      FFD40000      FFD80000      FFDC0000      FFE00000
    FFE40000      FFE80000      FFEC0000      FFF00000 (RO) FFF40000 (RO)
    FFF80000      FFFC0000
```

Codewarrior uygulamasında Edit menüsünden Preferences seçildikten sonra Şekil 3.3'deki Flash bellek ayarları yapılır. Burada dosya ismi, adres ve dosya büyüklüğü değerleri girilir.



Şekil 3.3: Flash Programlayıcı Ayar Ekranı

Ayarlar yapıldıktan sonra sırasıyla Flash programmer menüsünden önce Connect ve sonrasında Program düğmesi seçilerek uboot imaj dosyası Flash belleğe programlanmış olur. Bu işlemleri yapmadan önce dikkat edilecek bir nokta ise kart üzerindeki VPP ayarıdır. Bazı Flash bellekler sadece 12 volt ile programlanabilmelerine rağmen tez çalışmasında kullanılan Sharp Flash bellek modülü 5V ile de programlanabilmektedir [16] ve MPC8260ADS kartına herhangi bir 12V üreten kaynak bağlanmamıştır. Programlama esnasında 5V tarafında tutulması gereken ayar, normal çalışma esnasında 12V'a alınırsa ve sisteme 12 voltluk bir üreteç bağlanmaz ise Flash bellek istenmeyen yazma işlemlerine karşı korunmuş olur.

3.1.4 Çekirdek Derleme

Daha önceki bölümlerde çapraz derleme programı kurulmuş ve Power PC için Uboot imaj dosyası Flash bellek üzerine kaydedilmişti. Artık seri port üzerinden kontrol edilebilen MPC8260ADS, Uboot programının olanakları sayesinde çeşitli temel işlemler gerçekleştirilebiliyor durumdadır. Uboot'un asıl görevi olan çekirdeğe parametre göndererek çalıştırma işlevi için MPC8260ADS uyumlu bir çekirdeğe

ihitiyaç bulunmaktadır. Tez çalışmasında yine çapraz derleme programının yardımıyla sistem için uygun olan çekirdek derlenmiştir. Bu bölümde hangi çekirdek sürümünün kullanıldığı ve parametre ayarlarıyla birlikte çekirdeğin nasıl derlendiği özetle verilecektir.

Her Linux çekirdek sürümü MPC8260ADS'yi desteklememektedir, bu yüzden çeşitli sürümler internetten indirilerek araştırmalar yapılmış ve MPC8260ADS için en sorunsuz çalışan sürümün 2.4.25 olduğuna karar verilmiştir. Bu kararda derleme esnasında alınan hatalar, sürümün direkt MPC8260ADS desteğinin olup olmaması gibi ölçütler göz önüne alınmıştır. Linux çekirdeğinin 2.4.25 sürümünün açık kaynaklı kodu "The Linux Kernel Archives" [16] isimli siteden ücretsiz olarak indirilebilmektedir.

Bu çalışmada denx firmasının CVS sunucularından çekilen 2.4.25 sürümü çekirdek kaynağı kullanılmıştır. Bu kaynağa ulaşmak için aşağıdaki komutlar sırasıyla verilir;

```
bash$ cd /opt/eldk/usr/src
bash$ cvs -d :pserver:anonymous@www.denx.de:/cvsroot login
bash$ cvs -z6 -d :pserver:anonymous@www.denx.de:/cvsroot co -P
linuxppc_2_4_devel
bash$ cd linuxppc_2_4_devel
```

Bu aşamadan sonra çekirdek üzerinde parametre ayarlamaları yapılacak ve MPC8260ADS için uygun seçenekler seçilecektir. Öncelikle "make mrproper" komutu verilerek önceki derlemelerden kod ağacında herhangi bir sonuç kalmamasını sağlarız. Parametreleri kendi isteklerimize göre değiştirmeden önce bir başlangıç noktasına ulaşmak için MPC8260ADS'ye uygun hazır yapılandırma dosyası "make ads8260_config" komutu verilirerek yüklenir.

Genişleme kartının üzerindeki ethernet kapılarını da kullanıma açabilmek için "MPC8260 CPM Options" menusunun altında bulunan aşağıdaki ayarlar aktiflenmelidir;

```
  [*] Ethernet on FCC1
  [*] Ethernet on FCC2
  [*] Ethernet on FCC3
  [*] Use MDIO for PHY configuration
  [*] Support LXT970    PHY
```

Çalışmada ihtiyaç duyulacak olan ağ parametreleri ise “Networking Options” menusu altından ayarlanır;

```
<*> Packet socket
[ ]   Packet socket: mmapped IO
<*> Netlink device emulation
[*] Network packet filtering (replaces ipchains)
[ ]   Network packet filtering debugging
[*] Socket Filtering
<*> Unix domain sockets
[*] TCP/IP networking
[*]   IP: multicasting
[*]   IP: advanced router
[*]     IP: policy routing
[*]       IP: use netfilter MARK value as routingkey
[*]       IP: fast network address translation
[*]     IP: equal cost multipath
[*]     IP: use TOS value as routing key
[*]     IP: verbose route monitoring
[*]   IP: kernel level autoconfiguration
[*]     IP: DHCP support
[*]     IP: BOOTP support
[*]     IP: RARP support
< >   IP: tunneling
< >   IP: GRE tunnels over IP
[ ]   IP: multicast routing
[ ]   IP: ARP daemon support (EXPERIMENTAL)
[ ]   IP: TCP Explicit Congestion Notification support
[ ]   IP: TCP syncookie support (disabled per default)
IP: Netfilter Configuration  --->
IP: Virtual Server Configuration  --->
< >   The IPv6 protocol (EXPERIMENTAL)
< >   Kernel httpd acceleration (EXPERIMENTAL)
SCTP Configuration (EXPERIMENTAL)  --->
< > Asynchronous Transfer Mode (ATM) (EXPERIMENTAL)
< > 802.1Q VLAN Support
---
< > The IPX protocol
< > Appletalk protocol support
Appletalk devices  --->
< > DECnet Support
<*> 802.1d Ethernet Bridging
< > CCITT X.25 Packet Layer (EXPERIMENTAL)
< > LAPB Data Link Driver (EXPERIMENTAL)
[ ] 802.2 LLC (EXPERIMENTAL)
[ ] Frame Diverter (EXPERIMENTAL)
< > Acorn Econet/AUN protocols (EXPERIMENTAL)
< > WAN router
[ ] Fast switching (read help!)
[ ] Forwarding between high speed interfaces
QoS and/or fair queueing  --->
Network testing  --->
```

Bu menüde yukarıdaki ayarlar girildikten sonra buradaki “QoS and/or fair queueing” alt menüsüne girilerek aşağıdaki ayarlar yapılır;

```
[*] QoS and/or fair queueing
<*>   CBQ packet scheduler
<*>   HTB packet scheduler
<*>   CSZ packet scheduler
<*>   H-FSC packet scheduler
<*>   The simplest PRIO pseudoscheduler
<*>   RED queue
<*>   SFQ queue
<*>   TEQL queue
<*>   TBF queue
<*>   GRED queue
<*>   Diffserv field marker
<*>   Ingress Qdisc
[*]   QoS support
[*]     Rate estimator
[*]   Packet classifier API
<*>     TC index classifier
<*>     Routing table based classifier
<*>     Firewall based classifier
<*>     U32 classifier
<*>     Special RSVP classifier
<*>     Special RSVP classifier for IPv6
[*]     Traffic policing (needed for in/egress)
```

Tüm programlar “ram disk” adındaki bellekte tutulacaktır. Bu alan için varsayılan genişlik 4096 kilobayt’dır. Bu alan diğer programların da kurulmasıyla birlikte yetersiz kalacaktır. Bu sebeple çekirdek derlenmeden önce bu değerin 8192Kb’ye yükseltilmesinde fayda vardır. Bunun için ana menünün altındaki “Block Devices” menüsünden “Default RAM disk size” değeri 8192 olarak girilmelidir.

Üstte verilen tüm ayarlar tezin ileriki bölümlerinde kurulacak olan uygulamalar için asgari ayarlardır. Diğer çekirdek özellikleri, başka kurulmak istenen uygulamalar varsa ihtiyaca göre açılıp, kapatılabilir. Kart üzerindeki bellek sınırlı olduğu için çekirdek üzerinde kullanılmayacak olan bir özelliğin açılması uygun değildir. Tüm ayarlar yapıldıktan sonra konfigürasyon kaydedilerek çıkılır.

Çekirdeği derlemek için aşağıdaki komutlar sırayla verilir;

“make dep”

“make uImage”

Ortaya çıkan imaj dosyası (vmlinux.gz) çekirdek ağacının bulunduğu klasörün altındaki “arch/ppc/boot/images/” dizininde yer alır. Bu dosyadan sıkıştırılmış yüklenebilir bir imaj dosyası yaratmak için aşağıdaki komut kullanılır;

mkimage -A ppc -O linux -T kernel -C gzip -a 40000 -e 40000 -n evrenulusoy -d arch/ppc/boot/images/vmlinux.gz /tftpboot/deneme

Komut içerisindeki 40000 değerleri imajın açılırken yükleneceği adresi belirler. “-d” parametresi ile birlikte çıktının tftpboot dizini içerisinde yaratılmasını sağlanmış olunur. Böylece MPC8260ADS üzerinden aşağıdaki komutlarla imaj dosyası bilgisayar üzerinden yüklenir ve çekirdek koşmaya başlatılır;

tftp F00000 deneme

bootm F00000

Yukarıda verilen F00000 adresi ise sıkıştırılmış imaj dosyasının tftp ile nereye kopyalanacağının bilgisidir.

3.1.4.1 MPC8260ADS için Özel Yamalar

Hazırlanan çekirdek ile sistemi açmaya çalıştığımızda aşağıdaki alacaktır;

```
Caused by (from SRR1=49030): Transfer error ack signal
Oops: machine check, sig: 7
NIP: C01516A0 XER: 20000000 LR: C0151668 SP: C01BDF60 REGS: c01bdeb0
TRAP: 0200    Not tainted
MSR: 00049030 EE: 1 PR: 0 FP: 0 ME: 1 IR/DR: 11
TASK = c01bc000[1] 'swapper' Last syscall: 120
last math 00000000 last altivec 00000000
GPR00: 38800001 C01BDF60 C01BC000 C01BF600 00000001 00000019
C0123C54 00000000
GPR08: 00000008 FFFFD555 C0144780 F8000004 84000048 10109A3C
00FFE000 00040000
GPR16: 00800000 00FFF7EC C0157B30 00000040 C013DBE8 C01BF400
F00119C0 F0010000
GPR24: C013DBE8 00000000 C01BF400 C01BF600 FFFF9000 00000000
00000010 F0011300
Call backtrace:
C0151668 C01505DC C0151D1C C014FEE4 C00039C4 C00075B8
Kernel panic: Attempted to kill init!
 <0>Rebooting in 180 seconds..
```

“Call backtrace” satırında gözüktüğü üzere çekirdek açılırken C0151668 satırında hata almıştır. System.map dosyasının yardımıyla hatanın nerede olduğu anlaşılır;

c01510ec T fec_enet_init

c0151840 T sock_init

Yukarıda göründüğü gibi hata fec_enet_init fonksiyonuna gibi gözükmektedir. Bu fonksiyon arch/ppc/8260_io/ fcc_enet.c dosyasının içerisinde yer almaktadır. Hatanın yerini tespit etmek için bu dosyanın içine printf fonksiyonu kullanılarak problem tespit edilmeye çalışmıştır ve hatanın BCSR kütüğüne herhangi bir değer yazılmaya çalışıldığında ortaya çıktığı anlaşılmıştır. BCSR kütüğünün baz adresi ./arch/ppc/platforms/ads8260.h dosyasında yer almaktadır. Doğru değer bu dosyaya aşağıdaki şekilde girilir;

```
#define BCSR_ADDR               ((uint)0xf4500000)
```

Bu şekilde çekirdek tekrar derlenip açılmaya çalışınca aşağıdaki hatayı alacaktır;

```
Please append a correct "root=" boot option
Kernel panic: VFS: Unable to mount root fs on 00:00
```

Bu hata beklenen bir hatadır çünkü henüz dosya sisteminin imajı hazırlanmamıştır. Fakat arasatırlarda görülecek başka hatalar da vardır bunlar Ethernet aygıtı ile ilgili hatalardır;

```
eth0: FCC1 ENET Version 0.4, 00:00:11:AA:BB:CC
FCC: No PHY device found.
Can't get MII IRQ 25
eth1: FCC2 ENET Version 0.4, 00:00:11:EA:BB:CC
eth1: PHY id 0x08100003 is not supported!
Can't get MII IRQ 25
eth2: FCC3 ENET Version 0.4, 00:00:11:8A:BB:CC
FCC: No PHY device found.
```

Hatalardan anlaşıldığı gibi sadece FCC2 aygıtı algılanmış fakat onun için ise PHY id değeri yanlış tespit edilmiştir. Bu, fcc_enet.c doyasının içerisinden gelen hatalardır ve hataların ayaklanması gerektiğinin göstergesidir. Tez süresince bu hataların ayıklanmasında ciddi bir zaman harcanmıştır. Öncelikle PHY id ne geldiği ekrana bastırılıp, sürücü içerisinde beklenen değer gözlemlenen değer ile değiştirilmiştir. Buna göre phy_info_t yapısındaki phy_info_lxt970 değişkeninin içerisindeki 0x07810000 değeri 0x810000 ile değiştirilmiştir.

Bu değişiklikle birlikte yeniden derleme işlemi yapılıp çıktı gözlemlenirse aşağıdaki gibi bir çıktı ekrana gelecektir;

```
eth0: FCC1 ENET Version 0.4, 00:00:11:AA:BB:CC
FCC: No PHY device found.
Can't get MII IRQ 25
eth1: FCC2 ENET Version 0.4, 00:00:11:EA:BB:CC
```

eth1: Phy @ 0x0, type LXT970 (0x08100003)
Can't get MII IRQ 25
eth2: FCC3 ENET Version 0.4, 00:00:11:8A:BB:CC
FCC: No PHY device found.

FCC2 tanımlanmıştır fakat FCC1 ve FCC3 için problemler devam etmektedir. Daha önceki bölümlerde bahsedildiği gibi MPC8260 işlemci LXT970 fiziksel sürücüleriyle konuşurken MII kapılarını kullanmaktadır. Öncelikle MII adresileri kontrol edilmesi gerekir. Burada farkedilen bazı hatalar mevcuttur, bunların yanlış değerleri ve doğru değerleri Tablo 3.3'de verilmiştir;

Tablo 3.3: Sürücü Yamaları

DOĞRU	YANLIŞ
#define PC_F1RXCLK ((uint)0x00000400)	#define PC_F1RXCLK ((uint)0x00000800)
#define PC_F1TXCLK ((uint)0x00000800)	#define PC_F1TXCLK ((uint)0x00000400)
#define CMX1_CLK_ROUTE ((uint)0x37000000)	#define CMX1_CLK_ROUTE ((uint)0x3e000000)
#define MII_MDIO ((uint)0x00400000)	#define MII_MDIO ((uint)0x00000004)
#define MII_MDCK ((uint)0x00200000)	#define MII_MDCK ((uint)0x00000100)
#define MII_MDIO3 ((uint)0x00400000)	#define MII_MDIO3 ((uint)0x00000001)
#define MII_MDCK3 ((uint)0x00200000)	#define MII_MDCK3 ((uint)0x00000040)

MDIO ile ilgili değerler Ethernet sürücünün doğru şekilde tespit edilip yüklenmesinde kullanılmaktadır. RXCLK ve TXCLK değerleri yanlış verilmesi halinde giden ve gelen paketler için hatalı frekans değerleri kullanılacak dolayısıyla hatalı paketler alınacaktır. Her FCC için kullanılan CLK girişi değeri MPC8260ADS ve MPC8260TCOM kullanıcı el kitapları kullanılarak doğru değerler bulunmuştur.

MPC8260ADS üzerinde yer alan LXT970 (FCC2) ilk elektrik verildiği andan itibaren açık durumdadır fakat genişleme kartı üzerinde bulunan LXT970 fiziksel Ethernet sürücülerin ilk konumu kapalı durumdur.

Tüm FCC kapılarını 100Mbit hızında açık duruma getirmek için LXT970 için ayar parametrelerini fcc_enet.c dosyası içerisinde aşağıdaki gibi değiştirilmelidir;

```
(const phy_cmd_t []) {  /* startup - enable interrupts */
            { mk_mii_write(MII_LXT970_IER, 0x0002), NULL },
            { mk_mii_write(MII_LXT970_IER | 0x20, 0x0002), NULL },
            { mk_mii_write(MII_LXT970_IER | 0x40, 0x0002), NULL },

            { mk_mii_write(MII_REG_CR, 0x2100), NULL },
            { mk_mii_write(0x20,0x2100), NULL },
            { mk_mii_write(0x40,0x2100), NULL },

            { mk_mii_write(0x13,0x0080), NULL },
            { mk_mii_write(0x33,0x0080), NULL },
            { mk_mii_write(0x53,0x0080), NULL },

            { mk_mii_end, }
      }
```

Daha önceki kısımlarda verilen MII adres açıklamaları kontrol edilirse 0x13 FCC1, 0x33 FCC2 ve 0x53 ise FCC3 için olduğu gözükmektedir. Doğal olarak bu değerler yazılan kütüğe göre değişmektedir fakat her zaman aralarında 0x20 fark bulunmaktadır.

Tez çalışmasında kullanılan karta uygun olarak tüm düzeltmelerin yapıldığı sürücü dosyasının (fcc_enet.c) en güncel hali Ek.2'deki adreste veya tez CD'sinde bulunabilir. Sürücü güncellendikten sonra çekirdek tekrar derlenir ve sisteme yüklenip, açılış komutu verilir. FCC kapıları ile ilgili yüklemelerin yapıldığı bölümlerde çekirdek çıktısı aşağıdaki gibi olacaktır;

```
eth0: FCC1 ENET Version 0.4, 00:00:11:AA:BB:CC
?? = 1 fep->phy_id >> 4== 0x  810000
eth0: Phy @ 0x0, type LXT970 (0x08100003)
eth1: FCC2 ENET Version 0.4, 00:00:11:EA:BB:CC
?? = 1 fep->phy_id >> 4== 0x  810000
eth1: Phy @ 0x0, type LXT970 (0x08100003)
eth2: FCC3 ENET Version 0.4, 00:00:11:8A:BB:CC
?? = 1 fep->phy_id >> 4== 0x  810000
eth2: Phy @ 0x0, type LXT970 (0x08100003)
```

Bu durumdan sonra çekirdek hazır durumdadır. İstendiği takdirde daha önce gösterilen yöntemlerle Flash bellek üzerine kalıcı olarak kopyalanabilir.

3.1.5 Dosya Sistemi Oluşturma (BusyBox)

MPC8260ADS üzerinde programlar koşturulması için gerekli olan çekirdek derlendikten sonra temel programların da içerisinde yer aldığı bir dosya sistem imajı hazırlanması zorunludur. Normal bir bilgisayar üzerinde en temel bir dosya sistemi onlarca megabayt yer tutabilmektedir oysa ki kullanılacak sistemin bellek alanı sınırlıdır. BusyBox uygulaması özellikle gömülü sistemler için kullanılabilecek bir dosya sistemi olarak bilinmektedir. BusyBox bir adet çalıştırılabilir program ve çeşitli kütüphane dosyalarından oluşmaktadır. "ls, mv, cp, uname" gibi bir çok program sembolik bağlar olarak yaratılmaktadır ve bu bağların hepsi busybox çalıştırılabilir dosyasını göstermektedir. Çalıştırılabilir dosya kendini hangi sembolik bağın çağırdığını anlayıp uygun görevleri yerine getirmektedir. Ağ uygulamaları için gerekli olan özelliklerle birlikte derlenen busybox programının imaj dosyası 2 megabayt'ın altında yer almaktadır ki bu sınırlığı belleğe sahip olan MPC8260ADS için kabul edilebilir bir büyüklükte dosyadır.

Kurulum esnasında BusyBox uygulamasının 1.2.2 sürümü kullanılmıştır. İlgili sürüm [17] BusyBox uygulamasının kendi internet adresinden ücretsiz çekilebilmektedir. Kişisel bilgisayara indirilen dosya aşağıdaki komutlar ile açılır;

```
mkdir /busybox
cd /busybox
wget http://busybox.net/downloads/busybox-1.2.2.tar.bz2
tar –xzvf busybox-1.2.2.tar.bz2
cd busybox-1.2.2
make menuconfig
```

Bu menulerde BusyBox içerisinde derlenirken dahil edilmesi gereken özellikler seçilecektir. Burada en önemli seçeneklerden biri çapraz derleme seçeneğidir, bu ayar için sırasıyla "Busybox settings", "Build Options", "Cross Compiler prefix" seçilir. Bu değer "ppc_6xx-" ile değişitirilir. Böylece Busybox derlendiği zaman ortaya çıkan kodlar PowerPC uyumlu olacaktır. "Coreutils" ve "Networking Utils" menulerinden de "ls, mv, ping, ifconfig" gibi temel ağ görevlerini yerine getirecek uygulamalar seçilir. Busybox konfigürasyon dosyası ise Ek.3'deki adres veya tez CD'sinden alınarak bütün gerekli ayarları tekrar seçmemek için yeni bir kurulumda içerisinde kullanılabilir. Busybox için konfigürasyon ayarları kaydedilip menüden çıkıldıktan sonra sırasıyla "make" ve "make install" komutları verilir. Bu komutların sonucunda çalışılan klasörde "_install" isimli bir klasör yer almaktadır. "_install" klasörü

MPC8260ADS'ye yüklenecek dosya sistemi içermektedir fakat aygıt dosyaları burada yer almamaktadır. Bu dosyalar genext2fs adlı program ile birlikte klasör ext2 dosya sistemine çevrilirken otomatik yaratılır. "_install" klasöründen imaj dosyası oluşturmadan önce bu klasörün altına boş bir proc dizini yaratılması gerekir. Ayrıca, dosya sistemi imajının açılabilir olması için Ek.4'deki adreste veya tez CD'sinde yer alan metin formatındaki dosyaların "etc" klasörünün altına eklenmesi gereklidir. Eklenecek dosyalardan biri ise rc.sh dosyasıdır bu dosyanın içerisine sistem açıldıktan sonra çalışması istenen komutlar yazılabilir.

Aşağıdaki script çalıştırılarak uboot için yüklenebilecek imaj dosyası paylaştırılan tftp klasörüne gönderilir;

```
ROOTFS_DIR=disk                 # directory with root file system content
ROOTFS_SIZE=8192                # size of file system image
ROOTFS_FREE=100                 # free space wanted
ROOTFS_INODES=480               # number of inodes
ROOTFS_DEVICES=fs.tab2 # device description file
ROOTFS_IMAGE=ramdisk.img        # generated file system image
genext2fs -U \
      -d ${ROOTFS_DIR} \
      -D ${ROOTFS_DEVICES} \
      -b ${ROOTFS_SIZE} \
      -r ${ROOTFS_FREE} \
      -i ${ROOTFS_INODES} \
      ${ROOTFS_IMAGE}
rm -f ramdisk.img.gz
gzip -9 ramdisk.img
mkimage -T ramdisk -C gzip -a 800000 -e 800000 -n 'busy' -d ramdisk.img.gz busy
cp -v busy /tftpboot
```

Bu aşamdan sonra artık MPC8260ADS üzerinde uboot'a aşağıdaki komutu vererek Linux işletim sistemi çalıştırılabilir;

tftp fd000000 busy;tftp f00000 deneme;bootm f00000 fd000000

Sistem açılışının son satırları aşağıdaki gibi olacaktır;

```
VFS: Mounted root (ext2 filesystem).
eth0: config: auto-negotiation off, 100FDX, 100HDX, 10FDX, 10HDX.
--------------------
Welcome to MPC8260ADS
--------------------


BusyBox v1.2.2 (2007.02.24-12:46+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

/ $
```

3.1.6 Brctl Kurulumu (Köprü Uygulaması)

MPC8260ADS uygulama geliştirme kartı bir köprü cihazı olarak kullanılabilir. Bunun için Linux işletim sistemi üzerinde Brctl adlı uyguluma kurulmalıdır. Öncelikle bu uygulamayı PPC mimarisine göre derlemek gerekmektedir. Brctl uygulaması kolay bir şekilde çapraz derlemeyi desteklemediği için bu uygulama MPC8260ADS üzerinde derlenecektir. Bunun için denx yazılımı ile gelen ppc_6xx klasörünü NFS ile paylaştırdıktan sonra açılış parametrelerinde kök dizin olarak verilecektir. Brctl uygulaması internetten indirildikten sonra ilgili klasöre açılır. Configure.sh dosyasının oluşturulması için aşağıdaki komutlar sırasıyla verilir;

```
autoconf configure.in > configure.sh

chmod +x configure.sh
```

Uboot açıldıktan sonra çekirdek yüklenmeden önce çekirdeğe aşağıdaki açılış parametresi verilerek sistemin kök dizin olarak NFS ile sunucu bilgisayarda paylaşılan dizini kullanması söylenir.

```
setenv bootargs root=/dev/nfs rw nfsroot=192.168.1.2:8260image
nfsroot=192.168.1.2:/deneme/ppc_82xx nfsaddrs=192.168.1.10:192.168.1.2
ip=192.168.1.10:192.168.1.1:192.168.1.1:255.255.255.0:mpc8260:eth1:off


tftp F00000 deneme

bootm F00000
```

Yukarıdaki komutlar ile birlikte sistem NFS ile paylaşılan dizin üzerinden açılır. Burada brctl uygulamasının bulunduğu dizine geçildikten sonra ./configure komutu ardından da make komutu verilir. Böylece bu derlemede sabit sürücü olarak sunucu bir bilgisayar, işlemci olarak ise MPC8260 kullanılmıştır. Dolayısıyla elde edilen çalıştırılabilir köprü uygulaması Power PC uyumlu olacaktır. Fakat bu uygulama bazı kütüphane dosyalarına da ihtiyaç duyacaktır bu yüzden aşağıdaki dosyalar "_install" dizini altında yarıtılan lib klasörüne "ppc_6xx/lib" klasöründen kopyalanır;

ld-2.3.5.so

ld.so.1 (soft link)

libc-2.3.5.so

libc.so.6 (soft link)

Brctl çalıştırılabilir uygulaması busybox dizini altındaki "_install/sbin" klasörüne kopyalanır ve daha sonra 2.5.5 bölümünde anlatıldığı gibi tekrar imaj dosyası oluşturulur. Böylece tekar sistemi "tftp fd000000 busy;tftp f00000 deneme;bootm f00000 fd000000" komutu ile çalıştırdığımız zaman köprü uygulaması da dosya sisteminin içerisinde yer alacaktır. Sistem açıldıktan sonra verilecek brctl komutu ile birlikte ekran çıktısında köprü uygulamasının kendi komutları yer alacaktır;

```
/ $ brctl
Usage: brctl [commands]
commands:
        addbr           <bridge>                add bridge
        delbr           <bridge>                delete bridge
        addif           <bridge> <device>       add interface to bridge
        delif           <bridge> <device>       delete interface from bridge
        setageing       <bridge> <time>         set ageing time
        setbridgeprio   <bridge> <prio>         set bridge priority
        setfd           <bridge> <time>         set bridge forward delay
        sethello        <bridge> <time>         set hello time
        setmaxage       <bridge> <time>         set max message age
        setpathcost     <bridge> <port> <cost>  set path cost
        setportprio     <bridge> <port> <prio>  set port priority
        show                                    show a list of bridges
        showmacs        <bridge>                show a list of mac addrs
        showstp         <bridge>                show bridge stp info
        stp             <bridge> {on|off}       turn stp on/off
```

Artık sistem üzerinde bir köprü cihazı gerçekleştirmek çok basittir. Bunun için üç komut yeterlidir;

```
/ $ brctl addbr kopru
/ $ brctl addif kopru eth0
eth0: Promiscuous mode enabled.
device eth0 entered promiscuous mode
/ $ brctl addif kopru eth1
eth1: Promiscuous mode enabled.
device eth1 entered promiscuous mode
```

Köprü uygulaması IEEE802.1d standardının alt kümesini gerçekleştirecek şekilde kodlanmıştır. Ayrıca "spanning tree" algoritması brctl uygulaması tarafından desteklenmektedir. Bu özelliğini kullanmadan uygulama çalıştırılırsa ağ topolojisinde bulunması muhtemel çevrimler paketlerin sayaçları sıfırlanana kadar ağ içerisinde dönmesine neden olabilir. Bu durumun engellenmesi için "spanning tree protocol (STP)" özelliği aşağıdaki komut ile aktif hale getirilir;

```
/ $ brctl stp kopru on
```

"showstp" komutu ile "spanning tree" algoritması ile ilgili parametreler incelenebilir.

```
/ $ brctl showstp kopru
```

Diğer komutlarla köprünün bazı parametrelerini belirlememize olanak verir;
Bir MAC adresinden gelen paket sonrasında, ilgili MAC adresi iletim veritabanında belirli bir süre saklanır. Belirli bir süre sonra veritabanından silinmesi o MAC adresin ilgili kapıda sonsuza kadar kalmamasını sağlar. “Setageing” komutu bu süreyi elle belirlenmesini sağlar. Normal şartlarda değiştirilmesine gerek olmayan bu parametre aşağıdaki komut ile birlikte saniye cinsinden değiştirilebilir.

/ $ brctl setageing kopru_ismi zaman

Brctl uygulaması “spanning tree protocol” için detaylı ayar yapmamıza imkan verecek şekilde kodlanmıştır. Uygulama bu parametreleri bağlantı hızlarını ve diğer parametreleri kendi keşfedebilir böylece bunların normal şartlarda değiştirilmesine gerek kalmaz.

Eğer ağımızda birden çok köprü kullanıyorsa “setbridgeprio” koprülere öncelik vermek için kullanılır. En düşük öncelik değerli köprü, kök köprü olarak seçilir.

/ $ brctl setbridgeprio kopru_ismi oncelik

Bir köprüye bağlı kapıların farklı hızları olabilir ve bu nedenle kullanıcı her kapıya kendisine özel maliyet değeri atamak isteyebilir. “setpathcost” komutu ile maliyet atamaları yapılabilir.

/ $ brctl setpathcost kopru_ismi kapi maliyet

“setfd” iletim gecikmesini ayarlamak için kullanılır. İletim gecikme süresi iletim aşamasına geçmeden önce harcanan dinleme ve öğrenme sürelerinin toplamına eşittir. Bu demektir ki yeni bir köprü, yoğun bir ağa eklendiği zaman ağda aktif rol almadan önce belirli bir süre trafiği dinler ve elde edilen bilgilere göre iletim aşamasına geçer.

/ $ brctl setfd kopru_ismi zaman

“spanning tree” algoritmasında periyodik olarak köprüler tarafından “hello” paketleri gönderilir. Bu paketlerin hangi sıklıkta gönderileceği “sethello” komutu ile ayarlanabilir.

/ $ brctl sethello kopru_ismi zaman

Ağdaki diğer bir köprü cihazı belirli bir zaman “hello” paketi göndermediği zaman o köprü kaldırılmış sayılır. Bu süre “maxage” parametresi ile ayarlanabilir.

/ $ brctl maxage kopru_ismi zaman

Yaratılan köprü aksi belirtilmedikçe ağ üzerinde bir IP adresi almadan saydam bir işleyiş izler. Böylece köprüye dışardan erişim olamaz fakat köprünün kötü amaçlar için de kullanılması büyük bir olasılıkla engellenmiş olunur. İstenildiği zaman ise köprüye IP ataması da yapılması mümkündür. Bunun için diğer ethernet aygıtlarında kullanıldığı gibi "ifconfig" komutu kullanılır;

/ $ ifconfig kopru 192.168.1.100 netmask 255.255.255.0

3.1.7 IPtables2 ve Kurulumu

Linux işletim sistemi çekirdeği derlenmesi sırasında ağ üzerindeki paketlerinin filtrelenmesi ve NAT desteğinin sağlanması için gerekli modüller seçilip, derleme listesine dahil edilmişti. Bu bölüme kadar hazırlanan işletim sistemi bu özellikleri desteklemektedir fakat bunlar direkt kullanıcı katmanında açık olan özellikler değillerdir. Bu sebeple iptables paketi kullanıcı ile çekirdek modülleri arasında bir arayüz görevi görmektedir.

Tüm bu filtreleme ve NAT altyapısına NetFilter yapısı adı verilmektedir. Netfilter, sistem yöneticisine ağ paketlerini nasıl dağıtılacağını kurallar tanımlayarak belirlemesine olanak verir. Tüm bu kurallar zincirler içerisinde gruplanmıştır. Her zincir sıralı bir kurallar listesidir. Zincirler ise tablolar içerisinde gruplanmıştır ve her bir tablo farklı bir çeşit paket işlemi için kullanılmaktadır. Kurallar bazı tanımlar içerir ve bu tanımlar hangi paketin o kural için eşleneceğini belirler. Belirli bir kural ile eşleşen paketler, o paket için tanımlanan hedef işleme tabi tutulmaktadır. Sisteme gelen veya sistemden çıkan tüm paketler mutlaka en az bir zincire girer ve o zincirdeki kuralların tanımları ile karşılaştırılır. Eğer hiç bir kuralın tanımı ilgili paket ile eşleşmezse alınacak aksiyonu zincirin kendi varsayılan kuralı belirler. Netfilter içerisinde önceden tanımlı olan zincirlerin haricinde kullanıcılar da kendi zincirlerini yaratabilir. Sonrasında farklı kurallara hedef olarak bu zincire gitme işlemi atanabilir. Tüm kullanıcı tanımlı zincirlerin varsayılan kuralı "return" olarak tanımlanmıştır. Bu demektir ki zincir içerisinde hiç bir kuralın tanımı eşleşmez ise o zincire gelmeye sebep olan diğer zincirdeki kurala geri dönülür ve diğer kuralların tanımları karşılaştırılmaya devam edilir.

Üç adet ön tanımlı tablo vardır. Bu tablolar içlerinde yine önceden tanımlı bazı zincirler saklamaktadır. İlk başta tüm zincirler boştur. Boş zincirler, bütün paketlerin hiç değiştirilmeden geçmesine neden olan varsayılan zincir kurallarına sahiptir. Bu tablolar aşağıdaki gibi özetlenebilir;

Filtreleme tablosu: Bu tablo paketlerin filtrelenmesinden sorumludur, yani bir paketin yoluna devam etmesi veya düşürülmesine karar verir. Tüm paketler bu tablodan

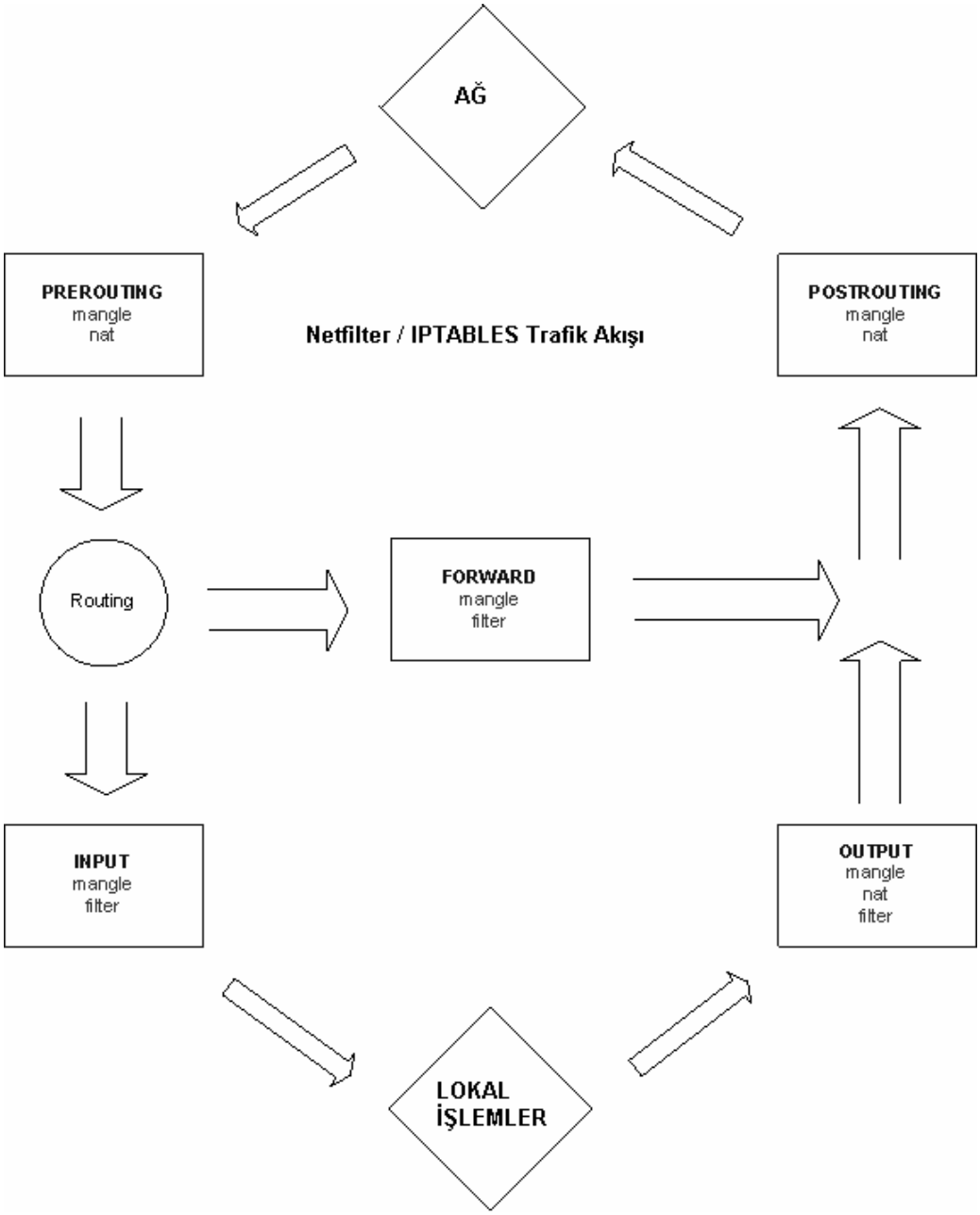
geçerler. Giriş, çıkış ve iletim zincirlerini barındırır. Giriş zinciri dışardan gelen paketlerin yönlendirildiği zincir, çıkış zinciri sistem üzerinde üretilen paketlerin dolaştığı zincirdir. İletim zincirinde ise sistem üzerinden geçen paketlerin işleme tabii tutulur.

NAT tablosu: Bu tablodaki zincirlerde paketlerin içerisinde adreslerin veya kapı değerlerinin tekrar yazılması için belirlenecek kurallar yer alır. Her bir bağlantının ilk paketi bu tablodaki zincirlere girer ve burada alınacak karar o oturum içerisindeki tüm paketler için geçerli olur. Prerouting zinciri gelen paketler için kullanılır ve özellikle hedef NAT (DNAT) uygulaması için yararlıdır. Postrouting zinciri ise paketler dışarı çıkmadan önce paket üzerinde yapılacak değişikliklerin belirlendiği kuralları içerir. Bu zincir kaynak NAT (SNAT) uygulamaları için kullanılmaktadır. NAT tablosundaki çıkış zinciri lokal paketler üzerinde kurallar belirleyerek limitli bir DNAT uygulaması sağlar.

Mangle tablosu: Bu tablo servis kalitesini belirleme gibi paket başlıklarındaki opsiyonlarının belirlenmesinde kullanılır. Tüm paketler bu tablodan geçerler. Olası tüm zincirler bu tablo içerisinde tanımlanmıştır;

- Prerouting(Yönlendirme Öncesi) : Dışarıdan gelen tüm paketler ilk önce bu zincire gelirler böylece paketin iletilmesi veya lokal işlemlere yönlendirilmesi konusunda karar verilir.
- Giriş: Hedefi sistem olan tüm paketler bu zincire tabii tutulur.
- İletim: Sistem üzerinden geçen paketler bu zincirde işlenirler.
- Çıkış: Sistem tarafından yaratılan paketler bu zincirde işlenirler.
- Postrouting(Yönlendirme Sonrası): Sistemi terkeden tüm paketler bu zincirden geçerler.

Netfilter mimarisindeki kurallar genel olarak Şekil 3.4'de [24] gösterilmiştir.



Şekil 3.4: Netfilter/IPTABLES Trafik Akışı

Tez çalışması sırasında Iptables 1.3.6 sürümü en son sürüm olarak seçilmiştir ve bu sürüm üzerinden kurulumlar tamamlanmıştır. Iptables, Netfilter [18] internet üzerinden ücretsiz olarak indirilebilir. Derleme bilgisayar üzerinde yapılmıştır. Öncelikle çapraz derleme programı yani ppc_6xx-gcc Makefile dosyası içerisinde aşağıdaki gibi tanımlanır.

CC=ppc_6xx-gcc

Daha sonra make komutu çekirdek kaynak kodununun lokasyonunu argüman olarak vererek çalıştırılmıştır;

```
make install KERNEL_DIR=/deneme/usr/src/linuxppc_2_4_devel
DESTDIR=/busybox/busybox-1.2.2/_install
```

Çapraz derleme programı kullanılarak derlenen iptables uygulaması Busybox uygulamasının imaj kaynağı klasöre kurulur. "/usr/local/lib/iptables" dizininde bulunan ve ipv6 ile ilgili kütüphane dosyaları yer kazanmak için silinebilir. Kütüphane isimlerinden hangi özellikleri sunduğu anlaşılabilir ve kullanılmayacak olan özellikleri sağlayan dosyaların yer kazanmak adına silinmesinde sakınca yoktur.

Bu kurulum sonrasında tekrar oluşturulan busybox imaj dosyasıyla birlikte sistem tekrar açılır ve iptables komutu basit bir kaç komut ile test edilebilir;

```
/usr/local/sbin $ ./iptables -V
iptables v1.3.6
/usr/local/sbin $ ./iptables -L INPUT
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
           0    --  anywhere             anywhere
/usr/local/sbin $ ./iptables -A INPUT -s 192.168.1.20 -j DROP
/usr/local/sbin $ ./iptables -L INPUT
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
           0    --  anywhere             anywhere
DROP       0    --  192.168.1.20         anywhere
```

Iptables ile birlikte kullanılacak temel komutlar aşağıdaki gibidir;

-A zincir_ismi : Belirli bir zincire kural eklemel için kullanılır.

-D zincir_ismi : Belirli bir zincirden kural silmek için kullanılır.

-L zincir_ismi : Belirli bir zincirdeki kuralları listelemek için kullanılır.

-F [zincir_ismi] : Zincir ismi verildiğinde o zincirdeki tüm kuralları, verilmediği zaman tüm zincirdeki kuralları siler.

Kural tanımlamak ve yukarıda verilen iptables komutlarının detayını görmek için iptables –h komutu kullanılır.

3.1.8 IProute2 ve Kurulumu

Iproute uygulamasının kurulumu ile birlikte MPC8260ADS üzerinde yönlendirme, tünel uygulamaları ve servis kalitesi için trafik kontrol uygulamalarını gerçekleştirmek mümkün olmaktadır. Kurulum sonrası iki adet yeni çalıştırılabilir uygulama kök dizin imaj dosyasında yer alacaktır. Bunlardan ilki yönlendirme, tünel yaratma gibi işlemleri gerçekleştirecek olan "ip" komutu, diğeri ise trafik kontrolünü sağlayacak olan "tc" komutudur.

"ip" komutu ile yapılabilecekler, uygulamayı argümansız çağrıldığında görülebilmektedir;

```
~ $ ip

Usage: ip [ OPTIONS ] OBJECT { COMMAND | help }
       ip [ -force ] [-batch filename
where  OBJECT := { link | addr | route | rule | neigh | ntable | tunnel |
                   maddr | mroute | monitor | xfrm }
       OPTIONS := { -V[ersion] | -s[tatistics] | -r[esolve] |
                    -f[amily] { inet | inet6 | ipx | dnet | link } |
                    -o[neline] | -t[imestamp] }
```

Aşağıda sık kullanılabilir bazı objelerin tanımları verilmiştir.

"link" objesi bir ağ aygıtına karşılık düşmektedir. "ip link" komutu ile aygıtlar özellikleri ile birlikte görüntülenebilir ve durumları değiştirilebilir. Örneğin FCC1 aygıtının çok yöne yayın (multicast) özelliği ip link komutu ile değiştirilebilir. "ip link help" yazılarak aygıtlarla ilgili değiştirilebilecek tüm özellikler ekrana listelenebilmektedir.

"addr" objesi bir aygıta bağlı olan IP veya IPv6 protokolüdür. "ip addr" objesi ile her aygıta bağlı adresler detaylı olarak görüntülenebilir veya değiştirilebilir.

"route" objesi kullanılarak linux çekirdeğinin yönlendirme tabloları görüntülenebilir veya değiştirilebilir. Detaylı bilgi için "ip route help" komutu kullanılır.

"neigh" objesi aynı bağ üzerinde bulunan ağ aygıtlarını görüntülemek ve çekirdekteki ARP tablosunu yönetmek için kullanılır.

Örneğin tez çalışmasındaki sistemde ana bilgisayar için MPC8260ADS üzerinden ping komutu çalıştırıldıktan sonra “ip neigh show” komutu çalıştırılırsa aşağıdaki çıktı elde edilir;

```
~ $ ping -c 1 192.168.1.2
PING 192.168.1.2 (192.168.1.2): 56 data bytes
64 bytes from 192.168.1.2: icmp_seq=0 ttl=64 time=0.7 ms

--- 192.168.1.2 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.7/0.7/0.7 ms
~ $ ip neigh show
192.168.1.2 dev eth1 lladdr 00:08:54:3b:d1:6c REACHABLE
```

“ip” komutu ile yapılabilecek konfigürasyon ayarları çok geniş bir konuyu kapsamaktadır ve tüm komutlar bu çalışmanın konusu dışında kalmaktadır fakat detaylı kullanım bilgisine “Linux Advanced Routing & Traffic Control HOWTO” [19] isimli internet sayfasından ulaşılabilir.

Iproute uygulamasının internet sayfasından [20] kaynak kodu ücretsiz indirilebilmektedir. Uygulamanın makefile isimli dosyası çapraz derlemeye uygun olmadığından ve derleme esnasında çekirdek kütüphane dosyalarına ihtiyaç duyulduğundan iproute MPC8260ADS üzerinde NFS kullanılarak derlenmiştir. Daha önceki bölümlerdeki gibi NFS ile paylaştırılan dizin MPC8260ADS’in kök dizini olarak kullanılmıştır. Kaynak kodu bir klasöre açılarak bu klasörde “make” komutu verilmiştir. Derlemenin yapıldığı klasörün bulunduğu yerde “ip” ve “tc” adlı iki alt klasör yer almaktadır. Bu klasörlerden birincisinden “ip” çalıştırabilir uygulaması, ikincisinden de “tc” çalıştırabilir uyguluması imaj dosyası hazırlarken kullanılan kök dizinin bulunduğu yere kopyalanmıştır. Ayrıca yine tc altklasörünün altında “.o” uzantılı dosyalar kök dizinin altındaki “usr/lib/tc” klasörüne kopyalanması gerekmektedir. Bu uygulamın ihtiyaç duyduğu iki kütüphane dosyası “libresolv.so.2” ve “libm.so.6” /usr/lib altına kopyalanır. Bu iki dosya NFS ile paylaştırılan klasördeki lib alt klasöründen alınır.

Busybox klasörünün altındaki kök dizin klasöründen tekrar imaj dosyası hazırlanmalıdır. Sistem tekrar yeni imaj dosyasıyla açılması halinde artık ip ve tc komutları çalıştırılabilir;

```
/ $ ip help
Usage: ip [ OPTIONS ] OBJECT { COMMAND | help }
       ip [ -force ] [-batch filename
where  OBJECT := { link | addr | route | rule | neigh | ntable | tunnel |
                   maddr | mroute | monitor | xfrm }
       OPTIONS := { -V[ersion] | -s[tatistics] | -r[esolve] |
                    -f[amily] { inet | inet6 | ipx | dnet | link } |
                    -o[neline] | -t[imestamp] }
/ $ tc help
Usage: tc [ OPTIONS ] OBJECT { COMMAND | help }
       tc [-force] -batch file
where  OBJECT := { qdisc | class | filter | action | monitor }
       OPTIONS := { -s[tatistics] | -d[etails] | -r[aw] | -b[atch] [file] }
```

3.1.9 UBoot ile Flash Üzerine Çekirdek ve Dosya Sistemi Programlama

Bu aşamaya kadar hazırlanan imaj dosyaları her sistem açılışı sırasında kaynak bilgisayardan tftp protokolü yardımıyla MPC8260ADS'nin geçici belleğine yüklenmekteydi. Bu yöntem MPC8260ADS'nin kaynak bir bilgisayar olmadan çalışması için engel oluşturmaktadır. Bu sebeple hazırlanan imaj dosyalarının son sürümleri MPC8260ADS ile birlikte gelen Flash bellek üzerine kopyalanmalıdır. Kopyalama işlemi uboot uygulamasının yardımıyla gerçekleşmektedir. İzlenecek adımların başında önce imaj dosyasının SDRAM üzerine kopyalanması, Flash bellek üzerinde silme işlemi ve son olarak da SDRAM üzerinden imaj dosyasının Flash belleğe programlanmadı olarak sayılabilir.

Öncelikle Flash belleğin sektörleri "flinfo" komutu ile listelenir;

```
Bank # 1: Sharp 28F016SC (16 Mbit, 32 x 64K)
  Size: 8 MB in 32 Sectors
  Sector Start Addresses:
    FF800000      FF840000      FF880000      FF8C0000      FF900000
    FF940000      FF980000      FF9C0000      FFA00000      FFA40000
    FFA80000      FFAC0000      FFB00000      FFB40000      FFB80000
    FFBC0000      FFC00000      FFC40000      FFC80000      FFCC0000
    FFD00000      FFD40000      FFD80000      FFDC0000      FFE00000
    FFE40000      FFE80000      FFEC0000      FFF00000 (RO) FFF40000 (RO)
    FFF80000      FFFC0000
```

32 sektörden her biri 256Kb genişliğinde veri alanı saklamaktadır. FFF00000 ve FFF40000 adresleri ile başlayan iki sektör yazma korumalıdır ve içerisinde uboot uygulamasının imajını saklar. Ayrıca, FFF00000 adresi sistem açıldığı zaman ilk okunacak adrestir bu yüzden uboot imajı mutlaka bu adreste yer almalıdır.

Geri kalan sektörlerin hangilerinin dolu olduğunu bulmak için “imls” komutu kullanılabilir. Bu komut Flash bellek üzerinde bulunan imaj dosyalarını listelemektedir. Örneğin bir çekirdek ve bir kök dizin imajı bulunan Flash bellek üzerinde bu komut koşturulduğunda aşağıdaki liste ekrana gelmektedir;

```
Image at FF800000:
  Image Name:   EVREN_22:49:16
  Image Type:   PowerPC Linux Kernel Image (gzip compressed)
  Data Size:    763024 Bytes = 745.1 kB
  Load Address: 00040000
  Entry Point:  00040000
  Verifying Checksum ... OK

Image at FF900000:
  Image Name:   Mpc8260_QoS_FS
  Image Type:   PowerPC Linux RAMDisk Image (gzip compressed)
  Data Size:    1972314 Bytes =  1.9 MB
  Load Address: 00800000
  Entry Point:  00800000
  Verifying Checksum ... OK
```

Bu aşamada listede ilgilenilen parametreler imajın adresi ve büyüklüğüdür. Örneğin ilk imaj FF800000 adresinde başlamış ve üç sektöre yayılmıştır. Kök dizin imajı ise FF900000 adresinden başlayıp 8 sektörde yer almaktadır. “flinfo” ile çıkan listede bu sektörler dolu olarak kabul edildikten sonra ardışık olarak yeni yükleyeceğimiz imaj dosyasına uygun büyüklükte sektörler seçilir. Eğer hazırladığımız çekirdek üç sektöre sığacak bir büyüklüğe sahip ise örneğin FFB00000 adresi uygun bir hedef olabilir. Öncelikle bu üç sektör aşağıdaki komut ile temizlenmelidir;

```
erase FFB00000 FFB7FFFF
```

“erase” komutu ile birlikte iki argüman verilir. İlk argüman silinmeye başlayacak sektörün başlangıç adresi, ikinci argüman ise silinecek son sektördeki son adres olarak tanımlanır. “tftp” komutu ile bilgisayar üzerindeki imaj dosyası SDRAM üzerine kopyalanır. Ancak burada dikkat edilmesi gereken bir nokta uboot kendini açılışta SDRAM’in en düşük adresine yüklediği için uboot uygulamasından uzak bir adrese imaj dosyası kopyalanmalıdır, örneğin 0x40000 adresi bu işlem için uygundur;

```
# tftp 40000 deneme
Using FCC2 ETHERNET device
TFTP from server 192.168.1.2; our IP address is 192.168.1.10
Filename 'deneme'.
Load address: 0x40000
Loading:
done
Bytes transferred = 721620 (b02d4 hex)
```

Artık “cp” komutunun yardımı ile birlikte Flash bellek programlanabilir. “cp” komutu üç farklı şekilde çağırılabilir “cp.b, cp.w, cp.l”. Bunlar byte, word ve long şeklinde adlandırılan veri tipleri için kısaltmalardır. Argüman olarak verilmesi gerekenler ise kaynak adres, hedef adres ve uzunluktur. Tüm argümanlar onaltılık tabanda verilmesi gerekir. ”tftp” komutunun çıktısında kopyalanacak uzunluk bayt cinsinden ve onaltılık tabanda verildiği için “cp.b” komutunun kullanılması “cp.w” ve “cp.l” komutlarına göre daha kolay olacaktır;

cp.b 40000 FFB00000 b02d4

Kök dizin imajı için de aynı işlem tekrarlandıktan sonra bootcmd ortam değişkeni aşağıdaki şekilde değiştirilir;

bootcmd=bootm FFB00000 FFBC0000

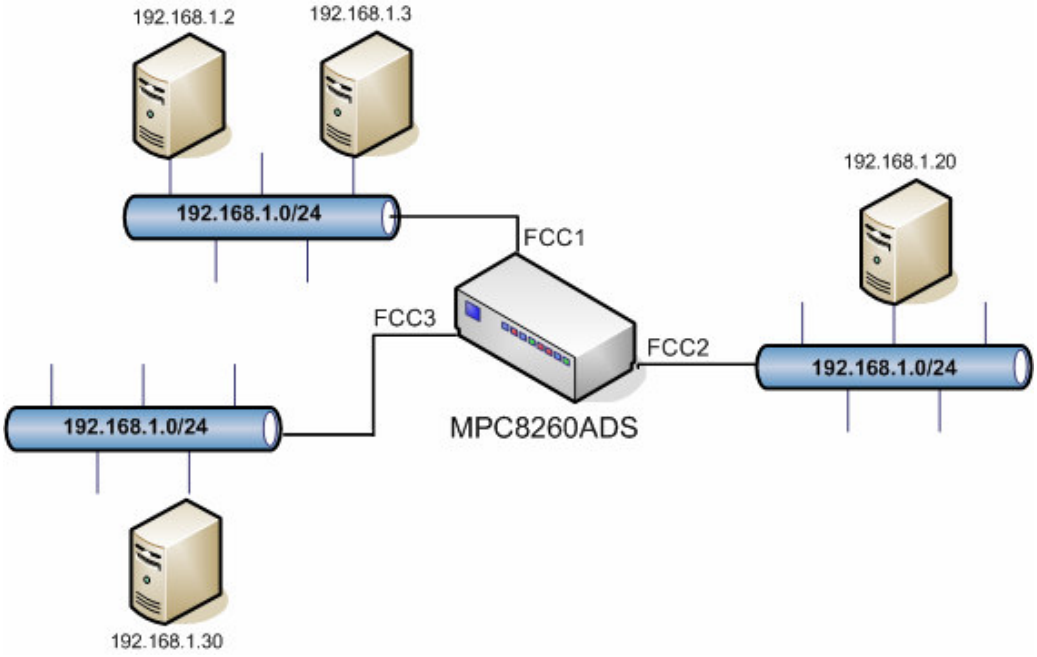
saveenv komutu ile birlikte uboot ortam değişikliği Flash belleğe programlanır. Artık sistem açıldığı zaman her seferinde tftp protokolü ile birlikte imaj dosyası ağ üzerinden çekilmeyecektir, bunun yerine Flash bellekte saklanan imajlar geçici bellekler üzerine açılıp, burada koşturulacaktır. Bu olanak kullanıcıya MPC8260ADS sisteminin imaj dosyalarının hazırlandığı bilgisayarın da bulunduğu geliştirme ortamından taşınıp farklı ortamlarda da hizmet verebilmesini sağlar.

4. MPC8260ADS'NİN AĞ UYGULAMARINDA KULLANIMI

Bu bölümde MPC8260ADS'nin farklı ağ görevlerini pratikte nasıl yerine getirebileceği örneklerle verilmiştir.

4.1 MPC8260ADS'nin Köprü Cihazı Olarak Kullanılması

Gerçek hayatta kullanılabilecek bir köprü uygulaması Şekil 4.1'de verilmiştir.



Şekil 4.1: MPC8260ADS ile Köprü Uygulaması

Ağ üzerindeki konakların sayısı çoğaldıkça hat üzerindeki çarpışmaların sayısı artabilir. Ağı, köprü cihazı kullanarak iki veya daha çok segmente bölmek paketlerin gereksiz yere tüm konaklara iletilmesini engelleyeceği için çarpışma sayısını büyük ölçüde azaltacaktır. Şekil 4.1'de MPC8260ADS 192.168.1.0/24 ağı üzerinde bir köprü görevi görmektedir. Konaklar ağı yine tek bir parça olarak algılayacakları için konaklar üzerinde herhangi bir yapılandırma değişikliğine ihtiyaç duyulmayacaktır. Brctl uygulamasında bir köprüden beklediğimiz gibi eğer paketlerin kaynak ve hedef adresi aynı segment içerisinde kalıyorsa bu paketin diğer segmentlere iletilmesini engellemektedir. Eğer kaynak ve hedef bilgisayar farklı segmentlerde kalıyorsa bu paket brctl uygulaması tarafından ilgili Ethernet kapıları arasında iletilmektedir.

Örneğin eğer FCC1 üzerinden gelen bir paketin hedefi FCC2 kapısı üzerinde tespit edilmiş ise bu paket FCC1 üzerinden sadece FCC2'ye iletilecektir. Eğer tabloda herhangi bir eşleştirme yapılamadıysa uygulama gelen paketi kaynak kapı haricideki tüm kapılara yönlendirir. Kaynakların hangi kapılara bağlı olduğunu öğrenmek için brctl uygulaması bir kaynak adres tablosu tutar ve bu tabloda her kaynak ilk paket gönderdiği anda paketin geldiği kapı ile kaynak eşleştirilir. Konaklar ağ üzerinde yer değiştirebilir, dolayısıyla aynı MAC adresi bir süre sonra farklı bir kapı üzerinden de paket göndermeye başlayabilir bu yüzden tablonun güncel tutulması önemlidir. Bu nedenle "aging" yani yaşlandırma kavramı brctl uygulamasında kullanılmaktadır. Böylece belirli bir süre içerisinde paket göndermeyen bir kaynak var ise o kaynak tablodan silinmektedir. Bu süre "brctl setageing" ile değiştirilebilmektedir. Örneğin "brctl setageing 30" komutu verildiği zaman 30 saniyeden daha uzun süre paket iletmeyen konaklar tablodan otomatik olarak silineceklerdir.

Ağlar büyüdükçe ağ üzerindeki köprü sayıları da artabilir ve çevrimler oluşabilir. Bu bir paketin eğer TTL değeri de düşürülmüyor ise sonsuza kadar ağ içerisinde dönmesine neden olabilir. Bu probleme çözüm olarak "Spanning Tree" protokolü sunulmuştur ve brctl uygulamasının desteklediği bu protokol "brctl stp" komutu ile aktif hale getirebilmektedir. Fakat Şekil 4.1'deki uygulamada "Spanning Tree" protokolüne döngü yer almadığı için ihtiyaç yoktur ve bu yüzden yapılandırma komutlarında bu özellik aktif hale getirilmeyecektir.

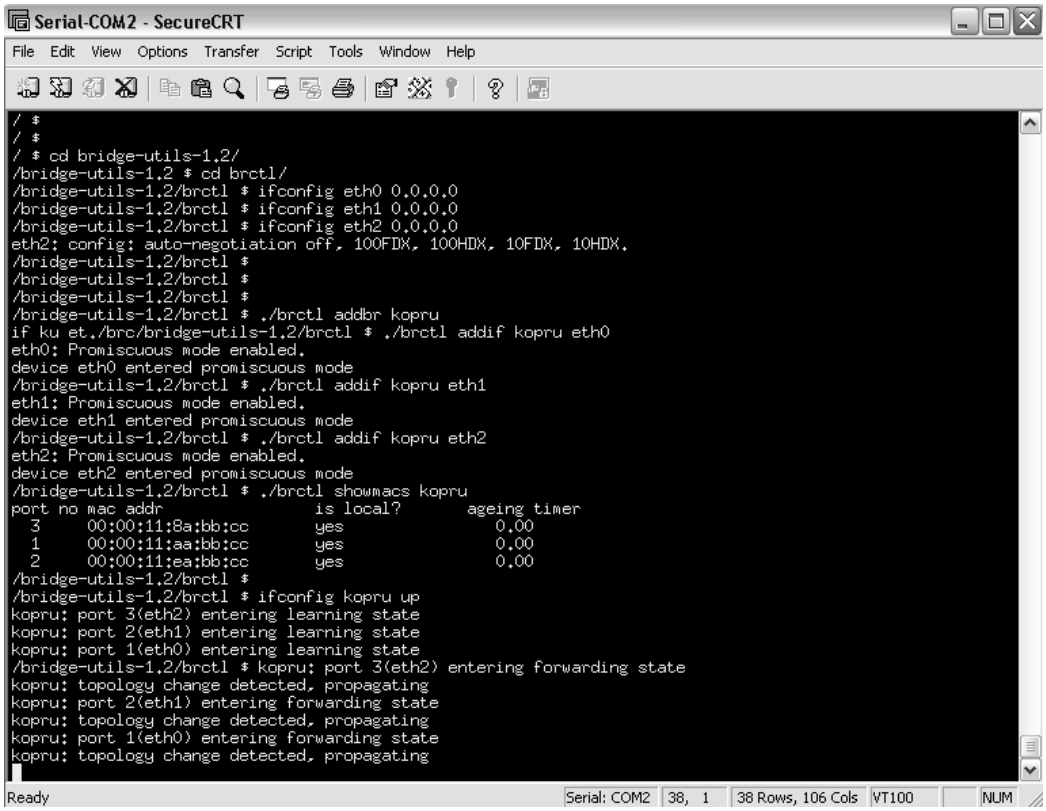
Yukarıdaki köprü uygulamasının MPC8260ADS yapılandırma komutlarına yansıması ise aşağıdaki gibi olacaktır;

```
ifconfig eth0 0.0.0.0
ifconfig eth1 0.0.0.0
ifconfig eth2 0.0.0.0
brctl addbr kopru
brctl addif kopru eth0
brctl addif kopru eth1
brctl addif kopru eth2
ifconfig kopru up
```

Yukarıdaki komutlar ile birlikte köprü aktif olarak çalışmaya başlayacaktır fakat köprüye herhangi bir IP adresinden ulaşmak mümkün olmayacaktır. Eğer sisteme seri iletişim kanalı yerine Ethernet üzerinden erişilmek istenirse köprüye bir IP adresi atamak gerekir. Seri iletişim kanalı üzerinden erişim genellikle geliştirme yapılırken kullanılmaktadır. Uygulamanın gerçek kullanım alanında bu tip cihazlara IP adresi kullanılarak erişilmektedir. MPC8260ADS'ye IP adresi atamak için aşağıdaki komut kullanılır;

ifconfig kopru 192.168.1.100 netmask 255.255.255.0

Yapılandırmayı test etmek için FCC1 kapısına Masaüstü bilgisayar (PC1 - 00:08:54:3b:d1:6c - 192.168.1.20), FCC3 kapısına dizüstü bilgisayar (PC2 - 00:12:79:bf:77:c8 - 192.168.1.30) ve FCC2 kapısına ise ADSL modem (00:14:c1:04:65:5c - 192.168.1.1) bağlanmıştır. Sırasıyla aşağıdaki komutlar çalıştırılmış ve köprü uygulamasının yönettiği kaynak adres tablonun nasıl değiştiği gözlemlenmiş ve yorumlanmıştır. Sistem açıldıktan sonra köprü yapılandırması Şekil 4.2'de görüldüğü gibi gerçekleştirilmiştir.



Şekil 4.2: Köprü Uygulaması Çalışırken

İlk önce PC1 ağ üzerinde olmayan bir IP adresine ICMP ping paketleri göndermiştir ve kendi MAC adresinin MPC8260ADS üzerinde kayıt altına alınmasını sağlamıştır;

PC1 : ping 192.168.1.200
MPC8260ADS : /bridge-utils-1.2/brctl $ ./brctl showmacs kopru

port no	mac addr	is local?	ageing timer
3	00:00:11:8a:bb:cc	yes	0.00
1	00:00:11:aa:bb:cc	yes	0.00
2	00:00:11:ea:bb:cc	yes	0.00
1	00:08:54:3b:d1:6c	no	2.53

PC1 kaynağından gelen paketin başlığı incelenerek kaynağın MAC adresi 1 nolu kapı ile eşleştirilmiş ve tabloya kaydedilmiştir. Modemin MAC adresi halen kaynak

tabloda yer almamaktadır. PC1 üzerinden Modeme paket gönderilmek istenirse bu 1 nolu yani paketin geldiği kaynak kapı hariç diğer kapılara yönlendirilecektir. Bunu gözlemlemek için PC2 üzerinde ethereal [21] uygulaması açılmış ve tüm gelen paketler gözlenmiştir. İlk gelen ICMP ping paketi tüm kapılara yönlendirilmiş fakat Modem'in verdiği cevap sonrasında artık PC2 üzerinde bu paket gözlemlenememiştir. Çünkü brctl uygulaması artık paketi sadece Modem'in bağlı olduğu kapıya yönlendirmektedir.

```
PC1 : ping 192.168.1.1
MPC8260ADS :  /bridge-utils-1.2/brctl $ ./brctl showmacs kopru
port no mac addr              is local?       ageing timer
  3     00:00:11:8a:bb:cc       yes              0.00
  1     00:00:11:aa:bb:cc       yes              0.00
  2     00:00:11:ea:bb:cc       yes              0.00
  1     00:08:54:3b:d1:6c       no               0.56
  2     00:14:c1:04:65:5c       no               0.56
```

PC2 konağının da MAC adresinin tabloya kaydedilmesi için PC2 üzerinden herhangi bir paket göndermek yeterli olacaktır;

```
PC2 : ping 192.168.1.20
MPC8260ADS :  /bridge-utils-1.2/brctl $ ./brctl showmacs kopru
port no mac addr              is local?       ageing timer
  3     00:00:11:8a:bb:cc       yes              0.00
  1     00:00:11:aa:bb:cc       yes              0.00
  2     00:00:11:ea:bb:cc       yes              0.00
  1     00:08:54:3b:d1:6c       no               7.85
  3     00:12:79:bf:77:c8       no               1.00
  2     00:14:c1:04:65:5c       no               7.85
```

Artık tüm MAC adresleri bağlı oldukları kapılarla eşleştirilmiştir ve ileride gönderilecek olan tüm paketler sadece hedef konakların bağlı oldukları kapıya yönlendirilecektir.

Sonuç olarak MPC8260ADS bu yapılandırma ile farklı bir ağ üzerinde de kurulabilir. Bulunduğu ağ üzerindeki tüm konakların yerlerini kendisi otomatik olarak güncelleyebilir. Dolayısıyla yeni ağ üzerinde de tekrar yapılandırma gerektirmeden bir köprü görevini yerine getirebilir.

4.2 MPC8260ADS'nin Yönlendirici Olarak Kullanılması

MPC8260ADS, Linux işletim sistemi sayesinde yönlendirici olarak kullanılabilir. İşletim sisteminin yönlendirici özelliklerinden faydalanabilmek için çekirdek, "IP: advanced router" ve "IP: policy routing" özellikleri seçili konumdayken derlenmesi gerekmektedir.

Çekirdek paketleri nasıl yönlendireceğine karar vermek istediği zaman öncellikle hangi yönlendirme tablosunu kullanacağını belirler. Yönlendirme politikası veritabanı (RPDB), çekirdeğin hangi sırayla yönlendirme tablolarını tarayacağını belirler. Çekirdek kurulumu ile birlikte gelen ön tanımlı üç adet tabloyu işaret eden üç kural vardır. Bu tablolar; lokal tablo, ana tablo ve varsayılan tablodur. RPDB üzerindeki kuralları listelemek için "ip rule ls" komutu kullanılır;

```
/ $ ip rule ls
0:      from all lookup local
32766:  from all lookup main
32767:  from all lookup default
```

Kurallar 0 ile 32767 arasında bir tanım numarası alır ve 0 numaralı tanıma sahip olan kural en yüksek önceliklidir. Yönlendirme tablolarındaki girdileri "ip route ls tablo_ismi" komutu ile birlikte listeleyebiliriz. Örneğin "default" isimli tablodaki girdileri ekranda görebilmek için aşağıdaki komut çalıştırılmaktadır;

```
/ $ ip route ls default
10.0.0.0/24 dev eth0  proto kernel  scope link  src 10.0.0.1
192.168.1.0/24 dev eth1  proto kernel  scope link  src 192.168.1.10
default via 192.168.1.1 dev eth1
```

Çekirdek, yeni bir paket geldiği zaman çekirdek yolu belirlemek için öncelikle bir hash tablosundan oluşan ön belleği tarar. Paket ön bellekte bulunmaz ise RPDB ve yönlendirme tabloları aşağıdaki sözde kod ile taranır;

```
Eğer (Paket.Önbellek = Doğru)
        { Yön = ÖnBellek[Paket.Hash_Degeri]; }
Değilse
        RPDB üzerindeki her kural için
        Eğer Paket.rpdb_arama_degeri içeriliyorsa seçilen_Kural
                yönlendirmeTablosu = rule [ seçilen_Tablo ]
                Eğer Paket.yönbulma_arama_degeri içeriliyorsa yönlendirmeTablosu
                        Yön = yönlendirmeTablosu = [Paket.yönbulma_arama_degeri]
```

Yönlendirme ön belleği “ip route show cache” komutu ile ekrana getirilebilir. “ip route” ile yönlendirme tabloları, “ip rule” komutu ile rpdb üzerindeki kurallar değiştirilebilir.

Daha önceki bölümlerdeki belirtildiği gibi gerekli paketler kullanılarak derlenmiş bir çekirdek hazırlanmış ise MPC8260ADS cihazını yönlendirici olarak kullanmak mümkündür. Bunun için aşağıdaki komutlar yeterli olacaktır;

ifconfig eth0 192.168.1.2 netmask 255.255.255.0

ifconfig eth1 10.0.0.5 netmask 255.255.255.0

ifconfig eth2 100.200.0.30 netmask 255.255.255.0

echo “1” > /proc/sys/net/ipv4/ip_forward

Yukarıdaki ilk üç komut MPC8260ADS üzerindeki her bir Ethernet çıkışına bağlı olduğu ağ üzerinde bir IP adresi atamaktadır. Son komut ise çekirdeğin yönlendirme özelliğini açabilmek için ip_forward parametresine 1 değeri atamaktadır. Ağ üzerindeki diğer konaklar “gateway” adresini MPC8260ADS üzerindeki IP adresi olarak verebilir. Bu durumda örneğin, 10.0.0.6 nolu konak 192.268.1.5 nolu konağa paket göndereceği zaman hedef MAC adresi olarak MPC8260ADS cihazının MAC adresini, hedef IP adresi olarak 192.168.1.5 adresini kullanacaktır. Çekirdek FCC1 kapısına gelen bu paketi 10.0.0.0/24 ağı üzerinde olduğunu anlayıp, paketi FCC2 kapısına yönlendirecektir. Eğer kartın direkt bağlı olduğu bir adres değil ise MPC8260ADS üzerinde kullanılan varsayılan çıkış kapısı kullanılacaktır. Varsayılan çıkış kapısı MPC8260ADS üzerinde “ip route default gateway xxx.xxx.xxx.xxx” komutu ile ayarlanmaktadır.

4.3 MPC8260ADS Üzerinde Ağ İletişim Hizmet Kalitesi Belirlenmesi (QoS)

Çekirdeğin ağ aygıtı üzerinde göndermek istediği birden çok paket bulunduğu zaman hangi paketin önce gönderilmesi, hangisinin bekletmesi veya hangisinin düşürmesi gerektiğini seçmek paket düzenleyicinin görevidir. Aynı zamanda paket düzenleyici dışarıdan gelen paketleri içeride lokal uygulamalara dağıtırken de benzer kuralları paketlere uygulayabilmektedir. Paket düzenleyici bu işi gerçekleştirmek için çeşitli algoritmalar kullanmaktadır. Çekirdeğin hangi algoritmaları destekleyeceği çekirdek derlenirken QoS menüsünden seçilebilmektedir. Yeni algoritmalar ise Linux modülleriyle birlikte çekirdek derlendikten sonra da sisteme entegre edilebilmektedir.

TC, Linux içerisinde trafiği kontrol etmek için kullanıcı katmanında kullanılan uygulamadır. Trafik kontrolü aşağıdaki kavramlardan oluşmaktadır;

Şekillendirmek: Trafik şekillendirilerek transmisyon hızı kontrol altına alınmaktadır. Kontrol altına almak sadece mevcut bandı kısıtlamanın dışında farklı düzenlemeler de yapmak olabilir. Örneğin trafikteki anlık artışları yumuşatmak da trafiği şekillendirmek olarak değerlendirilmektedir.

Düzenleme: Trafiği düzenleyerek yoğun trafiği etkilemeden etkileşimli trafiğe öncelik verilerek servis kalitesini yüksek tutulabilir. Örneğin, Telnet uygulamasının az sayıda paketi kuyrukta en yüksek öncellikli beklerken ftp uygulamasının yüzlerce paketi kuyrukta daha düşük öncelik ile işlenebilir.

Denetleme (Policing): Şekillendirme sadece dışarıya giden trafik ile ilgilenirken, denetleme ağdan içeriye gelen trafik üzerinde trafik düzenlemesi gerçekleştirmektedir.

Düşürme: Belirlenen bant genişliğini aşan paketler, hem gelen hem de gönderilen trafik için, kuyruklarda bekletilmeden düşürülebilmektedir.

TC trafiği işlemek için üç tip obje kullanmaktadır. Bu objeler kuyruklama disiplinleri (Qdisc), sınıflar ve filtrelerdir

Çekirdek bir ağ ara yüzü için paket işlemek istediğinde o paketi ilgili ara yüze bağlı olan Qdisc objesinin kuyruğuna ekler. Hemen arkasından çekirdek, aynı qdisc objesinden çekebildiği kadar paketi çekip ağ aygıtının sürücüsüne verir. Varsayılan yapılandırmada pfifo qdisc objesi ağ aygıtına bağlıdır. Pfifo, limiti paket veya bayt olarak belirlenebilen bir FIFO kuyruktur ve sistemdeki en temel qdisc objesidir.

Bazı qdisc objeleri sınıf veya sınıflar içerebilir. Bu sınıflar da yine kendi içerilerinde farklı qdisc objelerine yer vermektedir. Böylece gelen bir paket herhangi bir sınıf içerisindeki qdisc objesine kuyruklanabilmektedir. Çekirdek, sınıflı bir qdisc objesinden paket çekerken içerideki sınıflar arasındaki öncelliğe önem vermektedir.

Filtreler ise sınıflı qdisc objeleri tarafından kullanılmaktadır. Bir sınıflı qdisc objesine yeni bir paket girişi olduğu zaman o paket filtrelere tabii tutulur. Filtrelerden birinin vereceği karara göre o paketin sınıflı qdisc objesindeki hangi sınıfa yönlendirileceği belirlenir.

Sınıfsız qdisc objeleri şunlardır;

bfifo ve pfifo : FIFO algoritmasını gerçekleştirirler. Pfifo, paket sayısını limitlerken bfifo bayt sayısı ile limitler.

pfifo_fast : Çekirdek derlenirken “Advanced Router” özelliği açılmış sistemler için varsayılan qdisc objesidir. Üç adet farklı öncellikle FIFO kuyruktan oluşmaktadır. Pfifo_fast qdisc objesinde paketlerin hangi kuyrukta yer alacağı servis tipi bayrağına (ToS) göre karar verilmektedir.

RED: Bu qdisc objesi “Random Early Detection” algoritmasını gerçekler. RED algoritmasında mevcut bant genişliği tamamen doldurulmadan kuyruktan rastgele paketler düşürülmektedir.

SFQ: ‘‘Stochastic Fairness Queueing” algoritması için hazırlanmış qdisc objesidir. Bu algoritmada her oturuma sırayla paket gönderme hakkı verilmektedir. SFQ, işlemciyi en az yoran algoritmalardan biridir.

TBF: “Token Bucket Fitler” algoritmasını gerçekleyen bu qdisc trafiği belirli bir seviyeye azaltmak ve trafikteki ani artışları kısıtlamak için kullanılmaktadır.

Sistemde, sınıfsız qdisc objeleri aşağıdaki komut ile yapılandırılır;

“tc qdisc add dev DEV root QDISC QDISC_PARAMETRELERI”

Yukarıdaki komut ile eklenen qdisc objesi “tc qdisc del dev DEV root” ile kaldırılabilmektedir. Örneğin, varsayılan qdisc olarak pfifo_fast olarak ayarlanmış bir sistemde sfq qdisc objesi eth0 ağ aygıtına atamak istenirse aşağıdaki komutlar girilir;

```
/ $ tc qdisc show dev eth0
qdisc pfifo_fast 0: bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
/ $ tc qdisc add dev eth0 root sfq perturb 10
/ $ tc qdisc show dev eth0
qdisc sfq 8002: limit 128p quantum 1514b perturb 10sec
```

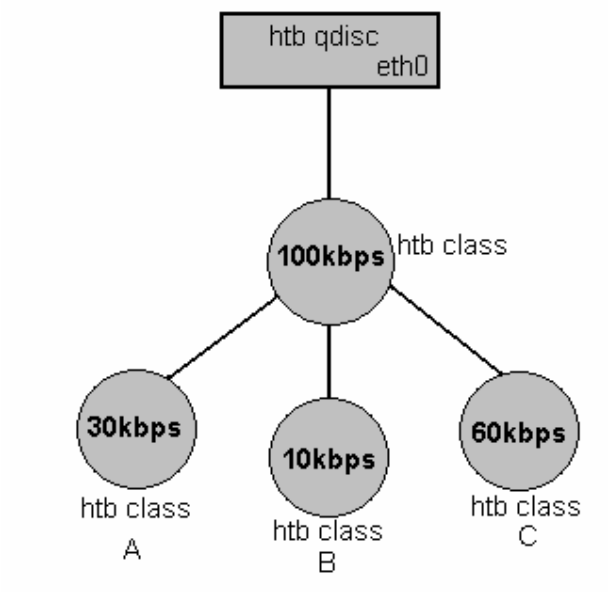
İlk ve son komut aygıtın önceki ve sonraki qdisc objesini ekrana getirirken ikinci komut eth0 aygıtına sfq qdisc objesini atamaktadır. Bu objeye perturb parametresi verilebilmektedir. Linux çekirdeğinde kullanılan SFQ algoritması paketleri 1024 kuyruktan birine atar. Paketlerin hangi kuyruğa gireceği başlık değerlerinden bazılarının bir hash algoritmasına sokulmasından sonra belirlenmektedir. Her ne kadar yeterli sayıda kuyruk olsa da farklı oturumlar aynı kuyrukta beklemeye başlayabilirler. Komut satırında verilen 10 perturb değeri ile birlikte hash algoritması her 10 saniyede bir değişmektedir. Böylece algoritmanın oturumlara daha eşit davranması sağlanmıştır.

Sınıflı qdisc objelerinden bazıları aşağıdaki gibidir;

PRIO: PRIO qdisc objesi trafiği şekillendirmez fakat gelen trafiği belirlenen filtrelere doğrultusunda farklı kuyruklara gidecek şekilde böler. PRIO qdisc objesi varsayılan olarak farklı önceliklerde 3 adet sınıf içerisinde her birinde FIFO kuyruğu içermektedir. Bu nedenle ilk bakışta pfifo_fast qdisc objesine benzemektedir. Objenin farkı ise kuyruk sayısı arttırılabilmesi ve her sınıfına farklı bir qdisc objesi atanabilmesidir. Sınıf ":1" en öncelikli sınıftır. Ancak sınıf "1:" içerisinde bekleyen paket yoksa çekirdek sınıf ":2" içerisinde bulunan kuyruğu sorgulamaktadır. Aynı kural diğer ardışık sınıflar için de geçerlidir.

CBQ: PRIO qdisc objesine benzemektedir ve trafiği şekillendirme yeteneğine sahiptir. Örneğin 100Mbit hızındaki bir bandı 10Mbit hızına düşürebilmektedir. Bunun için paketlerin geliş süresinden, hat için bir boş zaman hesaplar ve bu hesaba göre hattı belirli bir süre kullanmaz. CBQ kodunun karmaşıklığı ve kararsızlığı nedeniyle Linux ortamında çok sık kullanılan bir qdisc objesi olamamıştır.

HTB: Hiyerarşik jetonlu kova algoritmasını gerçekleştirmektedir. CBQ gibi davranan bu sınıflı qdisc objesi trafiği şekillendirmek için hattın boş zamanını hesaplamak yerine bilinen jetonlu kova (token bucket) algoritmasını kullanmaktadır. CBQ qdisc objesine göre daha yaygın kullanılmaktadır.



Şekil 4.3: Hiyerarjik Jetonlu Kova Yapısı

Örneğin Şekil 4.3'de gösterildiği gibi 100kbps hızındaki bir hat üç müşteriye paylaştırılmak istenebilir. A müşterisi 30kbps, B müşterisi 10kbps ve C müşterisi de 60kbps bant genişliğine sahip olacağı varsayılmaktadır. Bunun için aşağıdaki komutlar koşturularak eth0 ağ aygıtına bağlı bir htb sınıflı qdisc objesi ve yine bu kök objeye bağlı başka htb sınıf objeleri yaratılır.

```
tc qdisc add dev eth0 root handle 1: htb default 12
tc class add dev eth0 parent 1: classid 1:1 htb rate 100kbps ceil 100kbps
tc class add dev eth0 parent 1:1 classid 1:10 htb rate 30kbps
tc class add dev eth0 parent 1:1 classid 1:11 htb rate 10kbps
tc class add dev eth0 parent 1:1 classid 1:12 htb rate 60kbps
```

Eğer A, B veya C müşterisi kendilerine ayrılan bant genişliğini kullanmıyorlarsa bu kaynak israf ediliyor olacaktır. HTB'de bir ana sınıfın altında bulunan çocuk sınıf kendine ayrılan genişlikten fazlasını kullanabilir. Fakat bunun için öncelikle bağlı olduğu ana sınıfın altında diğer sınıflar tarafından kullanılmayan bant genişliği olmalı ve bu sınıf eklenirken maksimum kullanacağı genişlik de belirtilmelidir. Örneğin A ve C müşterisin tüm bant genişliğini kullanmadığı bir senaryoda B müşterisin gönderim hızının 10kbps'nin üzerine çıkması için aşağıdaki komut verilebilir. Burada B müşterisi eğer toplam bant genişliği uygunsa gönderim hızı 50kbps'ye kadar çıkabilmektedir;

```
tc class add dev eth0 parent 1:1 classid 1:11 htb rate 10kbps ceil 50kbps
```

A, B ve C müşterisinin paketlerini doğru sınıflara yönlendirmek için filtrelerden yararlanılmaktadır. Filtreler paket başlıklarının istenilen alanlarına bakabilirler. Örneğin Şekil 4.3'deki yapılandırma için kaynak adrese bakarak paketlerin hangi müşteriye gittiği belirlenmektedir;

```
tc filter add dev eth0 parent 10:0 protocol ip prio 1 u32 \
  match ip dst 10.100.2.1/32 flowid 1:10
tc filter add dev eth0 parent 10:0 protocol ip prio 1 u32 \
  match ip dst 10.100.2.2/32 flowid 1:11
```

C müşterisi için ayrıca bir filtre tanımlamaya gerek yoktur. Çünkü geriye kalan tüm paketler qdisc tanımlanırken belirlenmiş varsayılan sınıf 1:12'ye iletilecektir.

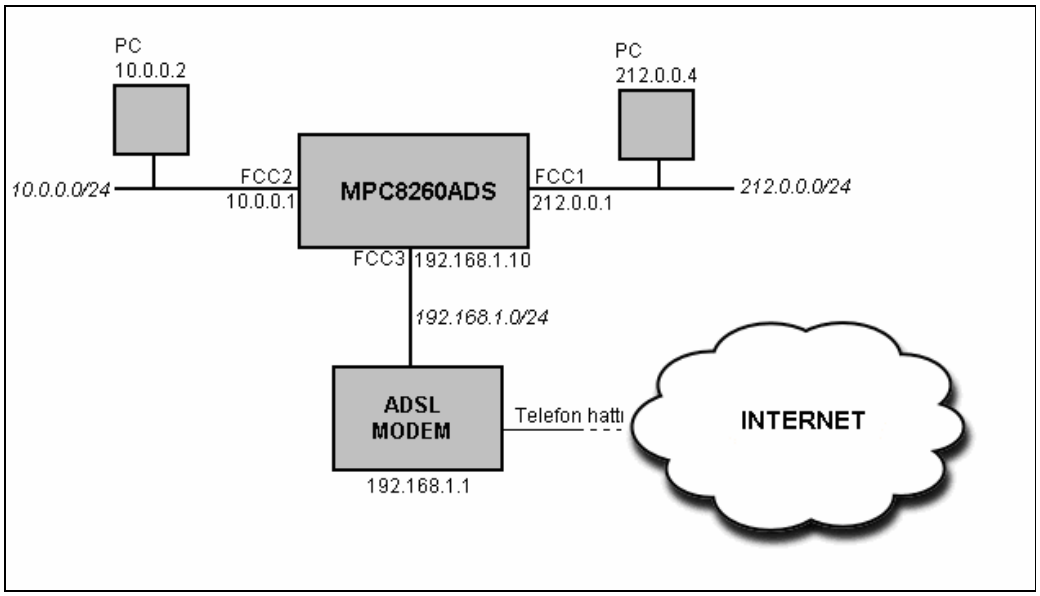
Bu bölümde verilmiş olan örneğin haricinde sınırsız sayıda uygulama gerçekleştirilebilir. Bu esnekliği sağlamak için tc komutu yeterince detaylı olarak

geliştirilmiştir. TC komutu ile yapılabileceklerin bir kısmı da “Linux Advanced Routing & Traffic Control HOWTO” isimli internet sayfasında örneklerle gösterilmiştir [19].

Yapılan çalışmalar sonrasında TC uygulaması ve Linux çekirdeğinin sağlamış olduğu trafik şekillendirme ve servis kalitesi yetenekleri kullanılmış ve bunların sayesinde MPC8260ADS, bir trafik şekillendirici ağ aygıtına dönüştürülebildiği gösterilmiştir.

4.4 MPC8260ADS Üzerinde Global Bir Örnek Uygulama

Bu bölümde MPC8260ADS, aynı anda yönlendirici, servis kalitesi politikası yöneticisi ve NAT sunucu olarak yapılandırılacaktır. MPC8260ADS’ye bağlı kişisel bilgisayarların IP adreslerini otomatik olarak alabilmesi için önceki bölümlere ek olarak DHCP sunucusu kurulacaktır. Bu bölüme kadar sistem seri bağlantı üzerinden kontrol edilmekteydi. Bu bağlantı yönteminin dezavantajı sistemin uzaktan kontrol edilebilmesinin mümkün olmamasıdır. Bu yüzden güvenlik de göz önüne alınarak telnet sunucusu yerine SSH sunucu kurulmuştur. SSH sunucusunun getirdiği olanak ile birlikte MPC8260ADS’ye istenilen herhangi bir noktadan gerekli izinler verildiği sürece bağlanılabilinmektedir. MPC8260ADS’nin bağlı olacağı ağların topolojisi Şekil 4.4’de verilmiştir.



Şekil 4.4: Örnek Uygulama Topolojisi

Topoloji üzerinde üç farklı ağ bulunmaktadır. MPC8260ADS üzerinde bulunan Ethernet kapılarının bağlı oldukları ağ adresleri tablo 4.1'de verilmiştir.

Tablo 4.1: Ethernet Kapı Adresleri

Ethernet Kapısı	Kapı IP Adresi	Bağlı olunan Ağ adresi
FCC1 (eth0)	212.0.0.1	212.0.0.0/24
FCC2 (eth1)	10.0.0.1	10.0.0.0/24
FCC3 (eth2)	192.168.1.10	192.168.1.0/24

Hedeflenen sistemde ADSL modem üzerinden erişilen internet, 10.0.0.0/24 ve 212.0.0.0./24 ağları arasında paylaştırılmaktadır. Bu iki ağın internete erişebilmesi için MPC8260ADS üzerinde iptables kullanılarak NAT sunucu yapılandırılacaktır. Her üç ağ üzerindeki konakların birbirlerine erişebilmesi için ise Linux çekirdeğinin paket yönlendirme özelliğinden yararlanılacaktır.

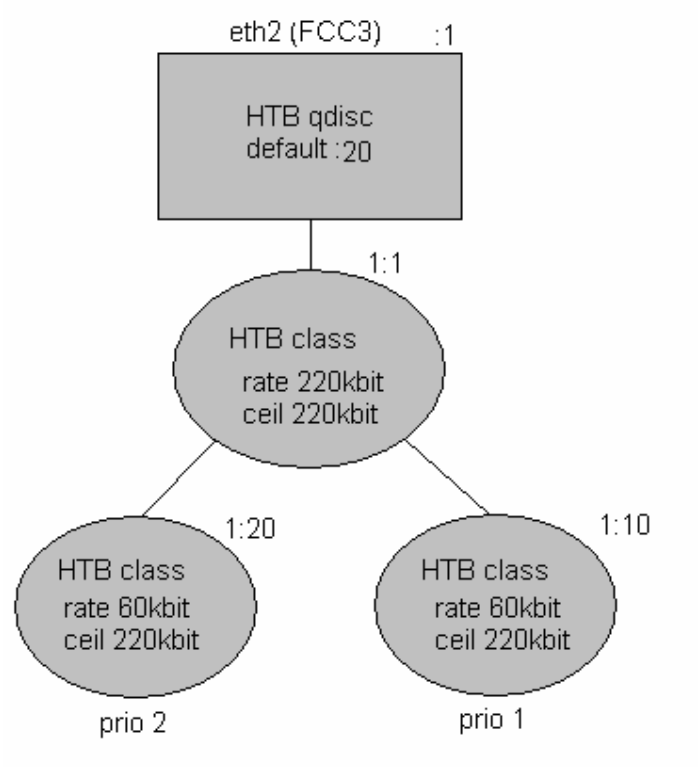
Linux çekirdeğinin özelliklerini kullanarak internet'e doğru giden bazı akışlar sınıflandırılacak ve bu sınıflara farklı öncellikler verilecek. İnternet'e doğru gerçekleştirilen dosya aktarım işlemleri, aynı anda yapılan dosya indirme işlemlerini dramatik bir şekilde yavaşlatmaktadır. İnternet'e doğru olan akış yönündeki kuyrukta ACK paketlerinin beklemesi, dosya indirme işleminin akışını yavaşlatmaktadır. Bu yüzden ACK paketleri gerçek zamanlı servis sınıfına sokulmalıdır. ACK paketleriyle birlikte gerçek zamanlı servis sınıfına eklenmesi gereken SSH, ICMP ve DNS paketleri de yer almaktadır. Böylece internet'e doğru yoğun bir trafik akışının olumsuz etkisi bu servisler üzerinde en aza indirgenecektir.

Internet'e çıkış noktasındaki darboğaz ADSL modem olduğu için bu yöne doğru olan akışlarda aktif kuyruk modem üzerindedir. Eğer bazı akışlara farklı davranmak isteniyorsa bu kuyruk kontrol edilebilen MPC8260ADS üzerine kaydırılmalıdır. Kuyruğun modem üzerinden MPC8260ADS üzerine kaydırmak için FCC2 çıkışının hızı modemin internet çıkış hızından daha düşük bir hız ile sınırlandırlır. Böylece modem üzerindeki kuyrukta birikme yaşanmayacak, paketler FCC2 çıkış kapısında bekleyeceklerdir.

Internet üzerinden gelen paketlerin ise maksimum hızı saniyede 1Mbit olarak varsayılmaktadır. Modem'den MPC8260ADS'ye olan bağlantı ise zaten saniyede 1Mbit'den daha hızlı (100Mbit) olacağı için bu yöndeki FIFO kuyrukta birikme yaşanmaz.

Kuyruğu, ADSL modemden MPC8260ADS üzerine kaydırmaktaki trafik şekillendirmenin haricindeki bir amaç ise, dışarıya giden paketlerin kuyruktaki sıralarını değiştirebilmektir. Burada asıl amaç ACK ve diğer öncellikli servislerin

paketlerine kuyrukta en yüksek önceliğe sahip olmalarını sağlamaktır. Internet üzerinden yoğun paket gelişi olduğu zaman, örneğin istemci bir çoklu medya dosyasını bilgisayara indirirken, dosya akışının devam edebilmesi için kaynak sunucuya istemciden sürekli ACK paketleri gönderilir. Böylece dolu olan TCP penceresi gönderilen ACK paketleriyle açılmış olur. Eğer ACK paketleri kuyruklarda bekletilirse, sunucudan da istemciye doğru olan akışın hızı yavaşlayacaktır veya mevcut genişliği doldurabilecek hıza ulaşamayacaktır. MPC8260ADS üzerinde bahsedilen paketlere öncelik verebilmek için şunlar yapılacaktır; iki adet delikli kova yaratılacak ve bu kovaların çıkış hızı sınırlandırılacaktır. İki kovanın da çıkışında birer FIFO kuyruk yaratılacaktır. Yüksek önceliğe sahip olan kovaya ACK ve diğer öncelikli paketleri, düşük önceliğe sahip olan kuyruğa ise geri kalan tüm paketler gönderilecektir. Şekil 4.5 üzerinde bu uygulamada MPC8260ADS üzerinde kullanılacak trafik kontrol sınıflarını ve kuyruklama disiplinlerini gösterilmektedir. Bu sınıflar “rate” parametresi ile verilmiş olan bant genişliğini garanti etmektedir. Alt sınıflarda kullanılan toplam bant genişliği bir üst sınıftaki bant genişliğinin altında kalırsa alt sınıflar “ceil” parametresi ile belirlenen hıza kadar çıkabilirler. Eğer birden çok alt sınıf boş bant genişliğini kullanmak isterse kullanılmayan bant genişliğini hangi sınıfın kullanılımına sunulacağı “prio” parametresi ile belirlenir. Düşük “prio” değerine sahip olan boş bant genişliğini kullanmakta önceliklidir. Akışları ayırıp onlara farklı öncelik verebilmek için bölüm 4.3’deki tekniklerden faydalınacaktır.



Şekil 4.5: Örnek Uygulamada Servis Kalitesi Yapısı

Şekil 4.4 ve Şekil 4.5'deki yapının kurulabilmesi için MPC8260ADS üzerinde koşturulacak komutlar açıklamalarıyla birlikte aşağıda verilmiştir.

```
# Bellekteki yönlendirme tabloları siler
ip route flush all

# eth0 aygıtına 212.0.0.0/24 ağı üzerinde 212.0.0.1 IP adresini atar
ifconfig eth0 212.0.0.1 netmask 255.255.255.0 \
broadcast 212.0.0.255

# eth1 aygıtına 10.0.0.0/24 ağı üzerinde 10.0.0.1 IP adresini atar
ifconfig eth1 10.0.0.1 netmask 255.255.255.0 \
broadcast 10.0.0.255

# eth2 aygıtına 192.168.0.0/24 ağı üzerinde 192.168.1.10 IP adresini atar
ifconfig eth2 192.168.1.10 netmask 255.255.255.0 \
broadcast 192.168.1.255

# Ağ üzerinde varsayılan çıkış kapısını modemin IP adresi yani 192.168.1.1 olarak ayarlar
ip route add default via 192.168.1.1

# Çekirdeğin yönlendirici özelliklerinin kullanılabilmesi için aygıtlar arasında
# paket alışverişini aktifleştirir.
echo "1" > /proc/sys/net/ipv4/ip_forward
```

```
# eth2 aygıtından çıkan paketlerin bağlantı durumlarını saklar ve yönlendirildikten
# sonra IP adresini eth2'nin IP adresi olarak değiştirir.
/usr/local/sbin/iptables --table nat --append POSTROUTING --out-interface eth2 -j
MASQUERADE

# eth2 aygıtına varsayılan hedefi 1:20 olan bir kök htb qdisc objesi atar
tc qdisc add dev eth2 root handle 1: htb default 20

# Ağ çıkış hızını sınırlandırmak için kök qdisc objesine bağlı bir htb sınıfı yaratır
tc class add dev eth2 parent 1: classid 1:1 htb rate 220kbit

# Yüksek öncelikli paketleri için 1:1 sınıfına bağlı 1:10 sınıf numarasına bağlı
# htb sınıfı yaratır. Yaratılan sınıflar içerisinde en düşük PRIO değerine sahip
# olmalıdır. Hızını aşmadığı durumlarda anlık 3kb,
# aştığı durumlarda 1kb gönderebilir
tc class add dev eth2 parent 1:1 classid 1:10 htb rate 60kbit ceil 220kbit prio 1 burst
3k cburst 1k

# Sınıflandırılmayan tüm paketlerin geleceği jetonlu kova yaratılır. Anlık çıkış hızı
diğer sınıfa göre
# sınırlandırılır. Hızını aşmadığı durumlarda anlık 2kb,
# aştığı durumlarda 1kb gönderebilir
tc class add dev eth2 parent 1:1 classid 1:20 htb rate 60kbit ceil 220kbit prio 2 burst
2k  cburst 1k

# Kovaların çıkışlarına fifo kuyruklar atanır.
tc qdisc add dev eth2 parent 1:10 pfifo limit 10
tc qdisc add dev eth2 parent 1:20 pfifo limit 30

tc filter add dev eth2 parent 1:0 protocol ip prio 100 u32 match ip  protocol 1 0xFF
flowid 1:10

# ACK paketlerini öncelikli sınıfa atar.
tc filter add dev eth2 parent 1: protocol ip prio 10 u32 \
  match ip protocol 6 0xff \
  match u8 0x05 0x0f at 0 \
  match u16 0x0000 0xffc0 at 2 \
  match u8 0x10 0xff at 33 \
  flowid 1:10

# SSH paketlerini öncelikli sınıfa atar.
tc filter add dev eth2 parent 1: protocol ip prio 1 u32 match ip sport 22 0xffff flowid
1:10

# DNS paketlerini öncelikli sınıfa atar
tc filter add dev eth2 parent 1: protocol ip prio 1 u32 match ip sport 53 0xffff flowid
1:10
```

Şekil 4.4'deki yapının gerçekleşmesi için bilgisayarlar üzerinde de ayarlar yapılması gerekmektedir;

- 10.0.0.0/24 ağı üzerindeki bilgisayarda IP adresi 10.0.0.2, ağ çıkışı ve dns sunucu adresleri ise 192.168.1.1 olarak ayarlanmıştır.
- 212.0.0.0/24 ağı üzerindeki bilgisayarda IP adresi 212.0.0.4, ağ çıkışı ve dns sunucu adresleri ise 192.168.1.1 olarak ayarlanmıştır.

Bu ayarların otomatik olarak yapılabilmesi için ise Busybox ile gelen DHCP sunucusu kullanılmıştır. Bunun için aşağıdaki yapılandırma dosyaları dosyası udhcpd programına parametre olarak verilerek DHCP sunucusu aktiflenmiştir.

```
start          212.0.0.4
end            212.0.0.254
interface      eth0
opt    dns     192.168.1.1
option  subnet  255.255.255.0
opt    router  212.0.0.1
option  domain  third
option  lease   864000         # 10 gun
```

```
start          10.0.0.2
end            20.0.0.254
interface      eth1
opt    dns     192.168.1.1
option  subnet  255.255.255.0
opt    router  10.0.0.1
option  domain  third
option  lease   864000         # 10 gun
```

Böylece bilgisayarlar ağa bağlandıkları zaman gerekli IP adresleri DHCP protokolünün yardımıyla otomatik olarak MPC8260ADS üzerinden almaktadırlar.

Erişimleri kontrol edebilmek için her iki bilgisayarda internet üzerinde bir sunucunun IP adresi sorgulanmış ve o sunucuya icmp ping paketi gönderilerek erişebilirlik test edilmiştir;

```
C:\Documents and Settings\EVREN>ipconfig

Windows IP Yapılandırması


Ethernet bağdaştırıcı Yerel Ağ Bağlantısı 3:

      Bağlantıya özgü DNS Soneki .  . . :
      IP Adres. . . . . . . . . . . . . : 10.0.0.2
      Alt Ağ Maskesi. . . . . . . . . . : 255.255.255.0
      Varsayılan Ağ Geçidi. . . . . . . : 10.0.0.1
```

```
C:\Documents and Settings\EVREN>nslookup
*** 192.168.1.1 adresinin sunucu ad_ bulunam_yor: Non-existent domain
*** Varsay_lan sunucular kullan_lam_yor
Varsayılan Sunucu:  UnKnown
Address:  192.168.1.1

> www.google.com
Sunucu:  UnKnown
Address:  192.168.1.1

G_venilir olmayan yan_t:
Ad:      www.l.google.com
Addresses:  209.85.129.147, 209.85.129.99, 209.85.129.104
Aliases:  www.google.com

> exit

C:\Documents and Settings\EVREN>ping 209.85.129.147 -n 1

32 bayt veri ile 209.85.129.147 'ping' ediliyor:

209.85.129.147 cevabı: bayt=32 süre=72ms TTL=243

209.85.129.147 için Ping istatistiği:
    Paket: Giden = 1, Gelen = 1, Kaybolan = 0 (0% kayıp),
Mili saniye türünden yaklaşık tur süreleri:
    En Az = 72ms, En Çok = 72ms, Ortalama = 72ms
```

```
C:\Documents and Settings\TCEULUSOY>ipconfig

Windows IP Configuration

Ethernet adapter Local Area Connection:

        Connection-specific DNS Suffix  . :
        IP Address. . . . . . . . . . . . : 212.0.0.4
        Subnet Mask . . . . . . . . . . . : 255.255.255.0
        Default Gateway . . . . . . . . . : 212.0.0.1

C:\Documents and Settings\TCEULUSOY>nslookup
*** Can't find server name for address 192.168.1.1: Non-existent domain
*** Default servers are not available
Default Server:  UnKnown
Address:  192.168.1.1

> www.google.com
Server:  UnKnown
Address:  192.168.1.1

DNS request timed out.
    timeout was 2 seconds.
```

```
Non-authoritative answer:
Name:    www.l.google.com
Addresses:  209.85.135.99, 209.85.135.103, 209.85.135.104, 209.85.135.147
Aliases:  www.google.com

> exit

C:\Documents and Settings\TCEULUSOY>ping 209.85.135.99 -n 1

Pinging 209.85.135.99 with 32 bytes of data:

Reply from 209.85.135.99: bytes=32 time=80ms TTL=242

Ping statistics for 209.85.135.99:
    Packets: Sent = 1, Received = 1, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 80ms, Maximum = 80ms, Average = 80ms
```

MPC8260ADS'nin yönlendirici özellikleri ise aşağıdaki komutlar ile birlikte test edilmiştir;

10.0.0.2 IP adresine sahip bilgisayardan;

```
C:\Documents and Settings\EVREN>ping 212.0.0.4 -n 1

32 bayt veri ile 212.0.0.4 'ping' ediliyor:

212.0.0.4 cevabı: bayt=32 süre<1ms TTL=127

212.0.0.4 için Ping istatistiği:
    Paket: Giden = 1, Gelen = 1, Kaybolan = 0 (0% kayıp),
Mili saniye türünden yaklaşık tur süreleri:
    En Az = 0ms, En Çok = 0ms, Ortalama = 0ms
```

212.0.0.4 IP adresine sahip bilgisayardan;

```
C:\Documents and Settings\TCEULUSOY>ping 10.0.0.2 -n 1

Pinging 10.0.0.2 with 32 bytes of data:

Reply from 10.0.0.2: bytes=32 time<1ms TTL=127

Ping statistics for 10.0.0.2:
    Packets: Sent = 1, Received = 1, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

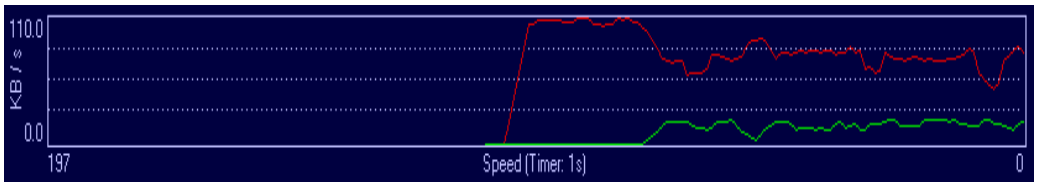
Servis kalitesindeki iyileştirmeyi görebilmek için bir pc üzerinden internet üzerindeki bir sunucuya dosya gönderilirken yine internet üzerindeki başka bir sunucudan dosya çekilmiştir. Bu işlem önce "tc" komutları çalıştırılmadan yapılmış ve çıktılar Şekil 4.6'da verilmiştir. Daha sonra aynı işlemler "tc" çalıştırılmış ve çıktılar Şekil 4.7'de verilmiştir.

Testler sırasında önce internetten dosya indirme işlemi başlatılmış ve iletişim sabit bir hıza ulaştığı anda başka bir sunucuya dosya aktarımı yapılmaya başlanmıştır. Dosya indirmek için "ftp.kernel.org", dosya aktarmak için ise "ftp.ulusoy.net" adresi kullanılmıştır. FTP istemci programı olarak ise Windows işletim sistemi üzerinde çalışan SmartFTP uygulaması seçilmiştir.

Şekil 4.6: "tc" Komutları Çalıştırılmadan Ölçülen Trafik

İlk test sırasında Şekil 4.6'da görüldüğü gibi lokal ağdan internet yönüne doğru bir trafik akışı başladığı zaman diğer yöndeki trafik ack paketlerinin dışarıya giderken bekletilmesinden dolayı olumsuz etkilenmektedir. Dosya indirme hızı saniyede 110kb değerinden 50kb değerinin altına kadar düşmüşütr.

İkinci test, paketleri sınıflandırıp onları farklı öncelikte ve bant genişliğine sahip kuyruklara sokarak yapılmıştır. Bunun için Şekil 4.5'deki yapıyı oluşturan "tc" komutları MPC8260ADS üzerinde koşturulmuştur. Böylece bu test sırasında ACK paketleri en hızlı şekilde MPC8260ADS üzerinden Modem'e verilmiştir. Bunun sonucunda internet üzerindeki sunucu, dosyayı gönderirken TCP penceresi ACK paketlerini beklerken dolmamaktadır ve böylece hızını bir önceki testteki kadar yavaşlatmamaktadır. İkinci testte dosya aktarım hızı ortalama saniyede 80Kb değerlerinde kalmıştır.



Şekil 4.7: "tc" Komutları Çalıştırıldıktan Sonra Ölçülen Trafik

Özet olarak Şekil 4.7'da de görüldüğü gibi servis kalitesi yöntemlerinin MPC8260ADS üzerinde kullanılmasıyla internete çıkış yönündeki düşük öncelikli trafik sınırlandırılmış ve böylece diğer yöndeki trafiği Şekil 4.6'daki kadar olumsuz etkilememiştir.

MPC8260ADS üzerinde ayarların değiştirilebilmesi, istatistiklerin görülmesi gibi yönetimsel işleri uzaktaki bir bilgisayar yardımıyla yapabilmek için dropbear [22] isimli açık kodlu SSH sunucu yazılım kullanılmıştır. Bu yazılımın kurulumu

MPC8260ADS NFS ile açıldıktan sonra kart üzerinde derlenmiştir. Dropbear derlendikten sonra üç adet çalıştırılabilir dosya çıkmaktadır. Bunlardan ilki sunucunun kendisi, diğerleri ise anahtar yaratmakla ilgili olan dosyalardır. SSH için gerekli olan anahtarlar PC üzerinde yaratılmış ve uygulamanın kendisiyle birlikte dosya sistemi imaj dosyasına eklenmiştir. Uygulamanın ihtiyaç duyduğu aşağıdaki dosyalar da /usr/lib dizinine imaj dosyası güncellenmeden önce kopyalanmıştır.

libcrypt.so.1
libz.so.1
libutil.so.1
libnss_files.so

SSH sunucusu, /etc/rc.sh dosyasının içerisinden aşağıdaki komut ile birlikte başlatılmaktadır.

/dropbear/dropbear -d /dropbear/dss_file -r /dropbear/rsa_file –E >/dev/null

Tüm komutlar busybox imaj dosyasında bir dosyada (config.sh) toplanmıştır. Sistemin bu dosyayı açılışta çalıştırması için ismi "/etc/rc.sh" dosyasına eklenmiştir. Dosya sisteminin son durumunun imajı oluşturulup Flash belleğe programlanmıştır.

Artık sistem açıldığı anda hiçbir komut girilmeden otomatik olarak istenilen yapılandırmayı destekleyecek hale gelmektedir. Böylece MPC8260ADS, seri ve jtag bağlantı yöntemleriyle bir bilgisayara bağlı olmadan tek başına belirli görevleri yerine getirebilen bir ağ cihazı konuma getirilmiştir.

5. SONUÇLAR VE TARTIŞMA

Bu tez çalışması süresince Motorola firmasının bir ürünü olan MPC8260ADS ağ uygulamaları geliştirme kartı kullanılmıştır. Tezin ilk aşamasında kart üzerinde kullanılan MPC8260 ağ işlemcisi incelenmiştir. Bu işlemcinin yetenekleri detaylı olarak araştırılmış ve bölüm 2.2 içerisinde anlatılmıştır. MPC8260 işlemci genel olarak iki çekirdekten oluşmaktadır. Ana işlemci PowerPC mimarisinde üretilmiş ve daha çok kullanıcı katmanındaki işleri yürütmektedir. RISC mimarisinde üretilmiş ikinci işlemci ise CPM adında bir modül içerisinde yer almaktadır. Bu modül, içerisinde özel olarak komünikasyon kanallarını kontrol eden çevre birim denetleyecilerini de barındırmaktadır. CPM kullanıcı tarafından uygun şekilde yapılandırıldıktan sonra bellek ile fiziksel ağ aygıtları arasındaki iletişimi CPM yönetir. Kullanıcı ise ağ aygıtından gelen veriyi işler, gönderilmek üzere yeni verileri tampon belleğe yazar.

Tezin ikinci aşamasında pratikte MPC8260ADS'nin Ethernet kapılarının işleyişi hakkında daha fazla bilgi sahibi olmak için Codewarrior üzerinde uygulama geliştirilmiştir. Codewarrior, Raven JTAG adaptörünü kullanarak direkt MPC8260ADS üzerinde C programlama dile uygulama geliştirmeye olanak vermektedir. Tez aşamasında geliştirilen uygulamada MPC8260ADS'nin iki Ethernet kapısı kullanılmıştır. Uygulamanın amacı herhangi bir kapıdan gelen paketi diğer kapıya iletmektir. Bunun için her gelen paket bir kesme üretmektedir ve bir fonksiyon yardımıyla bellek üzerinde paket diğer kapının çıkış işaretçisi tarafından gösterilmektedir. Böylece gelen paketler diğer kapı tarafından gönderilmeye hazır hale getirilmiş olmaktadır. İlk bakışta basit bir uygulama olarak gözükmesine rağmen düşük seviyede birçok kütüğün yapılandırılması gerekmektedir. Bu yapılandırmayla birlikte kartın çalışma prensibi hakkında detaylı bilgi birikimi elde edilmiştir. MPC8260 işlemcinin, LXT970 Ethernet sürücü aygıtındaki kütükleri yapılandırabilmesi için MII isimli ara yüz kullanılmaktadır. MPC8260 bu ara yüzü direkt olarak desteklememektedir. Bu nedenle geliştirilen uygulama içerisinde bu ara yüz üzerinden üç LXT970 üzerinde de işlem yapabilen bir fonksiyon geliştirilmiştir. Uygulamanın bu haliyle kart basit bir tekrarlayıcı görevi görmektedir. Daha karmaşık uygulamalar geliştirmek için kart üzerine Linux işletim sistemi kurulmuştur. Kurulum esnasında yaşanan birçok problem daha önceki uygulamada edinilen bilgiler

sayesinde aşılmış, Linux çekirdeği üzerinde gerekli değişiklikler yapılmıştır. Tezin bu aşamasının ilk etabında Linux çekirdek MPC8260ADS'ye uygun olarak değiştirilmiştir. Bu değişikler kaynak veya kütüphane dosyalarına bazı saklayıcı değerlerinin tekrar girilmesini, Ethernet sürücüsünde bazı yamaların yapılmasını kapsamaktadır. Hazırlanan çekirdek imaj dosyası kart üzerindeki Flash belleğe programlanmıştır. Sistemin açılması için gerekli olan kök dizin imajını en az yer kaplayacak şekilde hazırlayabilmek için BusyBox uygulamasından faydalanılmıştır. Bu uygulama Linux işletim sistemi üzerinde bulunan birçok aracı tek bir çalıştırılabilir dosya ve birkaç kütüphane dosyasında toplamaktadır. Çekirdek ve kök dizin imaj dosyaları ile birlikte MPC8260ADS üzerinde Linux işletim sistemi koşturulabilmiştir fakat kartı ağ uygulamalarında kullanabilmek için yardımcı programlar PowerPC mimarisine uygun olarak derlenip, kök dizin imaj dosyasına eklenmiştir. Bu uygulamalar sayesinde Linux çekirdeğinin ağ yetenekleri kullanıcı katmanından da erişebilmektedir. Böylece MPC8260ADS yönlendirici, ikinci katman köprüsü, güvenlik sağlayıcı, NAT sunucusu, servis kalitesi düzenleyicisi gibi uygulamaları gerçekleştirebilmektedir.

5.1 İleriki Çalışmalar

Codewarrior ile geliştirilen uygulama temel fonksiyonları ile birlikte daha gelişmiş bir ağ uygulaması için bir alt yapı oluşturmaktadır. Geliştirilebilecek uygulamada gelen paketlerin başlık bilgilerine göre farklı davranışlar sergilenebilir veya iletilen paketlerin başlık bilgileri istenildiği gibi değiştirilebilir. Örneğin sadece belirli bir TCP kapısına giden paketleri geçirebilir diğer kapılara geçen paketleri düşürebilir. Bu bir web sunucusu ise MPC8260ADS sadece 80 numaralı TCP kapısına giden paketlere izin vererek güvenlik sağlayabilir. Güvenlik için belirli bir süre içerisinde bir kaynaktan gelen paketler de sınırlanabilir. Bu alt yapı sayesinde bahsedilen örnek geliştirmeler, MPC8260ADS uygulama geliştirme kartına özel olmayan programlama teknikleriyle yapılabilir. Yapılması gerekenler sadece gelen ve giden tampon bölgelerdeki verileri işlemektir.

Kart üzerinde çalıştırılan Linux çekirdeği kart üzerindeki ağ aygıtlarından sadece Ethernet aygıtını desteklemektedir. MPC8260ADS, ATM, T1 gibi farklı yapıda ağ giriş çıkışlarına da sahip olduğu için bu aygıtları kullanamamak kartın birçok özelliğinden de faydalanmamak anlamına gelmektedir. Bu yüzden ileriki çalışmalarda ihtiyaca göre diğer fiziksel aygıtlar için Linux sürücüsü yazılabilir. Örneğin yazılacak bir ATM sürücüsü sayesinde MPC8260ADS, Ethernet ve ATM ağları arasında bir köprü görevi görebilir.

Tez sırasında hazırlanan örnek uygulamada servis kalitesi sağlamak için Ethernet kapılarının çıkış yönlerindeki trafik şekillendirilmiştir. Bunun nedeni Linux çekirdeği ile birlikte gelen trafik şekillendirme objelerinin sadece çıkış yönünü desteklemesindendir. Fakat internet üzerinde Linux çekirdeğe uygulanabilecek farklı yamalar sayesinde giriş yönündeki trafik de şekillendirilebilir. Bunlar için bir örnek IMQ objesidir. IMQ sayesinde htb, tbf gibi çıkış yönündeki trafiği şekillendiren objeler, giriş yönündeki trafiği şekillendirmek için de kullanılabilir. MPC8260ADS uygulama geliştirme kartı ile ileriki çalışmalarda bu yöndeki trafik de şekillendirilmek istenirse çekirdek IMQ yaması ile birlikte tekrar derlenip Flash bellek yeni imaj ile birlikte güncellenebilir.

KAYNAKLAR

[1] **IBM**, Power Architecture: A High-Performance *Architecture with a History*, http://www-03.ibm.com/servers/eserver/pseries/hardware/whitepapers/power/ppc_arch.html, Ziyaret Tarihi 19.04.2007

[2] **Wikipedia**. *Power PC: History*, http://en.wikipedia.org/wiki/PowerPC, Ziyaret Tarihi 19.04.2007

[3] **Motorola**. 1997, PowerPC Microprocessor Familiy: *The Programming Environments for 32Bit Microprocessors*. Motorola

[4] **Freescale Semiconductor, Inc**., *MPC68360 Product Summary,* http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=MC68360, Ziyaret Tarihi 19.04.2007

[5] **Freescale Semiconductor, Inc**., 2005. *Introducing Freescale's QUICC Engine*, Freescale

[6] **Macraigor Software**, *Raven Tester*, http://www.macraigor.com/downloads/Rav_tstr.exe, Ziyaret Tarihi 19.04.2007

[7] **Motorola**, 1999. *MPC8260 PowerQuicc II User's Manual*, Motorola

[8] **Motorola**, 1999. *MPC8260ADS User's Manual*, Motorola

[9] **DENX Software Engineering**, *ELDK Supported Host Systems*, http://www.denx.de/wiki/view/DULG/ELDKSupportedHostSystems, Ziyaret Tarihi 19.04.2007

[10] **DENX Software Engineering,** *Supported Target Architectures*, http://www.denx.de/wiki/view/DULG/ELDKSupportedTargetArchitectures, Ziyaret Tarihi 19.04.2007

[11] **Brune C.**, 2004. Das U-Boot: *The Universal Boot Loader, Das U-Boot*, http://linuxdevices.com/articles/AT5085702347.html, Ziyaret Tarihi 19.04.2007

[12] **DENX Software Engineering**, *Das U-Boot Universal Boot Loader*, ftp://ftp.denx.de/pub/u-boot/, Ziyaret Tarihi 19.04.2007

[13] **Linux Home Networking**, *Quick HowTo Ch16 Telnet,TFTP and xinetd,* http://www.linuxhomenetworking.com/wiki/index.php/Quick_HOWTO_:_Ch16_:_Telnet,_TFTP,_and_xinetd#Installing_The_TFTP_Server_Software, Ziyaret Tarihi 19.04.2007

[14] **Macraigor Software**, OCD Commander, http://www.macraigor.com/ocd_cmd.htm, Ziyaret Tarihi 19.04.2007

[15] **SHARP Integrated Circuits Group**, 1998. *LH28F016SCT-Z4 Flash Memory User's Manual*, Sharp.

[16] **Linux Kernel Organization**, *The Linux Kernel Archives*, http://www.kernel.org, Ziyaret Tarihi 19.04.2007

[17] **Erik Andersen**, *BusyBox*, http://busybox.net/, Ziyaret Tarihi 19.04.2007

[18] **Noris Network**, *Netfilter/Iptables*, http://www.netfilter.org/downloads.html, Ziyaret Tarihi 19.04.2007

[19] **Bert Hubert**, *Linux Advanced Routing & Traffic Control HOWTO*, http://lartc.org/howto, Ziyaret Tarihi 19.04.2007

[20] **LinuxNet**, *Iproute2*, http://linux-net.osdl.org/index.php/Iproute2, Ziyaret Tarihi 19.04.2007

[21] **Ethereal Software**, *Ethereal Network Protocol Analyzer*, http://www.ethereal.com/, Ziyaret Tarihi 19.04.2007

[22] **Matt Johnston**, *Dropbear SSH server and client*, http://matt.ucc.asn.au/dropbear/dropbear.html, Ziyaret Tarihi 29.04.2007

[23] **Level One**, 1997. *LXT970 Fast Ethernet Transceiver Datasheet*, Level One

[24] **Daniel Miessler**, 2007. *Iptables*, http://dmiessler.com/study/iptables/, Ziyaret Tarihi 19.04.2007

EKLER

Ek1:

```
#include <string.h>
#include <stdlib.h>
#include "netcomm.h"       /* global defines */
#include "mpc8260.h"       /* IMM definitions and declarations */
#include "ethernet.h"      /* Local header file */


t_PQ2IMM  *IMM;         /* Internal Memory Map base pointer */

BDRINGS *BD_FCC2;       /* buffer descriptors base pointer */
BDRINGS *BD_FCC1;

LOG     *LOGS[LOG_SIZE];

BUFFER *fcc1buffer[NUM_RXBDS+NUM_TXBDS];
BUFFER *fcc2buffer[NUM_RXBDS+NUM_TXBDS];

VUBYTE NotDone;        /* Termination of Rx flag */
VUBYTE RxProcIndex;    /* keeps track of next BD to process */
VUWORD RxGood;         /* Successful RX flag */
UWORD  *POINT;
UBYTE  turn;
UWORD c_total_1;
UWORD c_total_2;
VUHWORD wF1=0;
VUHWORD wF2=0;
UWORD  logptr;
VUHWORD tF1=0;
VUHWORD tF2=0;
WORD evren = 0x69868269;
WORD evren2= 0x78857685;
WORD evren3= 0x837989FF;
WORD evren4;



/*-------------------------------------------------------*/
/* Interrupt Handler Code to be moved to Address 0x500 */
/*-------------------------------------------------------*/

extern UWORD ExtIntTable[];



void     Main(void);
void     InterruptVectorInit(UWORD *, UWORD[]);
void     InitBDs(void);
void     InitParallelPorts(void);
```

```
void    InterruptControlInit(void);
void    FCC1Init(void);
void    FCC2Init(void);
void      processFCC1(void);
void      processFCC2(void);
void      enableFCC(void);
void    ExtIntHandler(UWORD);
UHWORD   BDRxError(UHWORD);
UHWORD   BDEmpty(UHWORD);
UHWORD   LastBD(UHWORD);
void    GP0led(UHWORD);
void    GP1led(UHWORD);
void    FlashGP1led(void);
void    mdioSend(UBYTE,UWORD,UHWORD);
void    fethReset(void);
void    SetEEinMSR(void);

void main()

{

   UWORD check;
   UWORD *yer;
         UWORD j;

/*evren4=evren;
evren4=evren2;
evren4=evren3;*/

c_total_1=0;
c_total_2=0;

   IMM = (t_PQ2IMM *)(BASE_ADDR);  /* pointer to MPC8260 internal memory
                              map. */
   GP0led(OFF);       /* turn off signal LEDs */
   GP1led(OFF);


   InterruptVectorInit((UWORD *) EXT_INT_VECTOR, ExtIntTable);
      InterruptVectorInit((UWORD *)0x0900, ExtIntTable);

   enableFCC();

   BD_FCC2 = (BDRINGS *)(BDRING_BASE);

   BD_FCC1 = (BDRINGS *)(BDRING_BASE+0x1000);
   logptr=0;
   tF1=0;
   tF2=0;
   wF1=0;
   wF2=0;


         yer=(UWORD *)0xFFF00000;



   for (check=0;check<0xFFF;check++) {
   *yer=0x00000000;
   yer++;
```

```
    }


          yer=(UWORD *)POOL_BASE;


    for (check=0;check<(ENET_MRBLR+32)*NUM_TXBDS;check++) {
    *yer=0x00000000;
    yer++;
    }



          yer=(UWORD *)POOL_BASE;
    for (check=0;check<(ENET_MRBLR+32)*NUM_TXBDS;check++) {
    if(*yer!=0x00000000)FlashGP1led();;
    yer++;
    }



    for(j=0;j<LOG_SIZE;j++)
             LOGS[j] = (LOG *)LOG_BASE+j;

                for(j=0;j<LOG_SIZE;j++)

                LOGS[j]->fcc=0;



    for(j=0;j<NUM_RXBDS+NUM_TXBDS;j++){
    fcc1buffer[j]=(BUFFER *)(POOL_BASE+j*(ENET_MRBLR+32));
    fcc2buffer[j]=(BUFFER
*)(POOL_BASE+(j+NUM_RXBDS+NUM_TXBDS)*(ENET_MRBLR+32));
    }

   GP1led(ON);
    for(j=0;j<0x100000;j+=1){
    POINT=(UWORD *)0x04000000+j;
    *POINT=0;
    }

    POINT = (UWORD *)POOL_BASE;
    for(j=0;j<0x1000;j+=1){
    POINT=(UWORD *)POOL_BASE+j;
    *POINT=0x00000000;
    }
    GP1led(OFF);

    InitParallelPorts();

          InterruptControlInit();
//    while (1);
          fethReset();

    mdioSend(0,0,0x2100);
    mdioSend(4,0,0x01E1);
    mdioSend(19,0,0x0000);
    mdioSend(17,0,0x0002);
```

```
    mdioSend(0,1,0x2100);
    mdioSend(4,1,0x01E1);
    mdioSend(19,1,0x0000);
    mdioSend(17,1,0x0002);

    InitBDs();    /* Initialize RX and TX BDs */
    FCC2Init();
    FCC1Init();

    while (1){processFCC1();processFCC2();};
}



void InterruptVectorInit(UWORD *interrupt_vector,
                UWORD interrupt_code[])

{

UHWORD rxIndex;
UWORD *instruction;
UWORD *next_vector;


   next_vector = (interrupt_vector + VECTOR_BLOCK_LEN); /* next vector entry */

   for(instruction = interrupt_vector, rxIndex = 0; instruction < next_vector;
       instruction++, rxIndex++)

      *instruction = interrupt_code[rxIndex];

} /* end InterruptInit */







void InitBDs()

{

UHWORD i;

   IMM->fcc_regs[FCC2].gfmr &= !(GFMR_ENT | GFMR_ENR);
   for (i = 0; i < NUM_RXBDS; i++)

   {

      BD_FCC2->RxBD[i].bd_addr = (UWORD
*)(POOL_BASE+((NUM_RXBDS+NUM_TXBDS+i)*(ENET_MRBLR+32)));
      BD_FCC2->RxBD[i].bd_length = 0;

      BD_FCC1->RxBD[i].bd_addr = (UWORD *)(POOL_BASE+(i*(ENET_MRBLR+32)));
      BD_FCC1->RxBD[i].bd_length = 0;
```

```
    if( i != (NUM_RXBDS-1) )

    {  BD_FCC2->RxBD[i].bd_cstatus = 0x9000;    /* Empty */
       BD_FCC1->RxBD[i].bd_cstatus = 0x9000;
    }

    else {
       BD_FCC2->RxBD[i].bd_cstatus = 0xB000;
       BD_FCC1->RxBD[i].bd_cstatus = 0xB000;
    }
  }



  for (i=0; i < NUM_TXBDS; i++)

  {


    BD_FCC2->TxBD[i].bd_length = 0;
    BD_FCC1->TxBD[i].bd_length = 0;

         if ( i != (NUM_TXBDS-1) ){
    BD_FCC2->TxBD[i].bd_cstatus = 0x4400;
    BD_FCC1->TxBD[i].bd_cstatus = 0x4400;
    }
    else {
    BD_FCC2->TxBD[i].bd_cstatus = 0x6400;
    BD_FCC1->TxBD[i].bd_cstatus = 0x6400;
    }

    BD_FCC2->TxBD[i].bd_addr = (UWORD
*)(POOL_BASE+(i+NUM_RXBDS*2+NUM_TXBDS)*(ENET_MRBLR+32));
    BD_FCC1->TxBD[i].bd_addr = (UWORD
*)(POOL_BASE+(i+NUM_RXBDS)*(ENET_MRBLR+32));

                    }



  for (i=0; i < NUM_TXBDS; i++)

  {

  fcc2buffer[i]->INFO[0]=0xFF;
  fcc2buffer[i]->INFO[1]=0xFF;
  fcc2buffer[i]->INFO[2]=0xFF;
  fcc2buffer[i]->INFO[3]=0xFF;
  fcc2buffer[i]->INFO[4]=0xFF;
  fcc2buffer[i]->INFO[5]=0xFF;
  fcc2buffer[i]->INFO[6]=0xAF;
  fcc2buffer[i]->INFO[7]=0xAF;
  fcc2buffer[i]->INFO[8]=0xAF;
  fcc2buffer[i]->INFO[9]=0xAF;
  fcc2buffer[i]->INFO[10]=0xAF;
  fcc2buffer[i]->INFO[11]=0xAF;
  fcc2buffer[i]->INFO[6]=(UBYTE)i;
```

```
  }
}


void InitParallelPorts()

{
   IMM->io_regs[PORTA].ppar = 0x00000000;
   IMM->io_regs[PORTB].ppar = 0x00000000;
   IMM->io_regs[PORTC].ppar = 0x00000000;
   IMM->io_regs[PORTD].ppar = 0x00000000;


   IMM->io_regs[PORTA].pdir = 0x00000000;
   IMM->io_regs[PORTB].pdir = 0x00000000;
   IMM->io_regs[PORTC].pdir = 0x00000000;
   IMM->io_regs[PORTD].pdir = 0x00000000;


   IMM->io_regs[PORTA].psor = 0x0000003f;
   IMM->io_regs[PORTB].psor = 0x00000004;
   IMM->io_regs[PORTC].psor = 0x00000000;
   IMM->io_regs[PORTD].psor = 0x00000000;


   IMM->io_regs[PORTA].pdir = 0x00003c0c;
   IMM->io_regs[PORTB].pdir = 0x000003c5;
   IMM->io_regs[PORTC].pdir = 0x00600000;
   IMM->io_regs[PORTD].pdir = 0x00000000;


   IMM->io_regs[PORTA].podr = 0x00000000;
   IMM->io_regs[PORTB].podr = 0x00000000;
   IMM->io_regs[PORTC].podr = 0x00000000;
   IMM->io_regs[PORTD].podr = 0x00000000;


   IMM->io_regs[PORTA].ppar = 0x0003fc3f;
   IMM->io_regs[PORTB].ppar = 0x00003fff;
   IMM->io_regs[PORTC].ppar = 0x00003c00;
   IMM->io_regs[PORTD].ppar = 0x00000000;


}



void InterruptControlInit()

{
  IMM->ic_simr_l = ALL_ZEROS;
//   IMM->ic_simr_l = 0xC0000000;
  IMM->fcc_regs[FCC1].fccm = 0x000C0000;
  IMM->fcc_regs[FCC2].fccm = 0x000C0000;
  SetEEinMSR();

}
```

```
void FCC1Init()

{

        t_EnetFcc_Pram* FCC1Ethernet;

//    IMM->pram.serials.fcc_pram[FCC1].riptr = 0x3900;
  //  IMM->pram.serials.fcc_pram[FCC1].tiptr = 0x3A00;
    IMM->pram.serials.fcc_pram[FCC1].riptr = 0x3000;
    IMM->pram.serials.fcc_pram[FCC1].tiptr = 0x3500;

    IMM->pram.serials.fcc_pram[FCC1].reserved0= 0x0000;
    IMM->pram.serials.fcc_pram[FCC1].rbase = (UWORD)&BD_FCC1->RxBD[0];
    IMM->pram.serials.fcc_pram[FCC1].tbase = (UWORD)&BD_FCC1->TxBD[0];
    IMM->pram.serials.fcc_pram[FCC1].rstate &= 0x00FFFFFF;
    IMM->pram.serials.fcc_pram[FCC1].rstate |= 0x30000000;
    IMM->pram.serials.fcc_pram[FCC1].tstate &= 0x00FFFFFF;
    IMM->pram.serials.fcc_pram[FCC1].tstate |= 0x30000000;
    IMM->pram.serials.fcc_pram[FCC1].mrblr = ENET_MRBLR;

    FCC1Ethernet =
    (t_EnetFcc_Pram * ) &(IMM->pram.serials.fcc_pram[FCC1].SpecificProtocol.e);

    FCC1Ethernet->c_mask   = ENET_C_MASK;
    FCC1Ethernet->c_pres   = ENET_C_PRES;
    FCC1Ethernet->crcec    = ALL_ZEROS;
    FCC1Ethernet->alec     = ALL_ZEROS;
    FCC1Ethernet->disfc    = ALL_ZEROS;
    FCC1Ethernet->ret_lim  = ENET_RET_LIM;
    FCC1Ethernet->p_per    = ALL_ZEROS;
    FCC1Ethernet->gaddr_h  = ALL_ZEROS;
    FCC1Ethernet->gaddr_l  = ALL_ZEROS;
    FCC1Ethernet->tfcstat  = ALL_ZEROS;
    FCC1Ethernet->mflr     = ENET_MFLR;
    FCC1Ethernet->paddr1_h = ENET_PADDR_H;
    FCC1Ethernet->paddr1_m = 0x79;
    FCC1Ethernet->paddr1_l = ENET_PADDR_L;
    FCC1Ethernet->iaddr_h  = ALL_ZEROS;
    FCC1Ethernet->iaddr_l  = ALL_ZEROS;
    FCC1Ethernet->minflr   = ENET_MINFLR;
    FCC1Ethernet->taddr_h  = ALL_ZEROS;
    FCC1Ethernet->taddr_m  = ALL_ZEROS;
    FCC1Ethernet->taddr_l  = ALL_ZEROS;
    FCC1Ethernet->pad_ptr  = IMM->pram.serials.fcc_pram[FCC1].tiptr;
    FCC1Ethernet->cf_type = ALL_ZEROS;
    FCC1Ethernet->maxd1   = ENET_MDMA;
    FCC1Ethernet->maxd2   = ENET_MDMA;
    FCC1Ethernet->octc=0;          /* received octets counter. */
    FCC1Ethernet->colc=0;          /* estimated number of collisions */
    FCC1Ethernet->broc=0;          /* received good packets of broadcast address */
    FCC1Ethernet->mulc=0;          /* received good packets of multicast address */
    FCC1Ethernet->uspc=0;          /* received packets shorter then 64 octets. */
    FCC1Ethernet->frgc=0;          /* as uspc + bad packets */
    FCC1Ethernet->ospc=0;          /* received packets longer then 1518 octets. */
    FCC1Ethernet->jbrc=0;          /* as ospc + bad packets  */
    FCC1Ethernet->p64c=0;          /* received packets of 64 octets. */
    FCC1Ethernet->p65c=0;          /* received packets of 65-128 octets. */
    FCC1Ethernet->p128c=0;         /* received packets of 128-255 octets. */
```

```
    FCC1Ethernet->p256c=0;          /* received packets of 256-511 octets. */
    FCC1Ethernet->p512c=0;          /* received packets of 512-1023 octets. */
    FCC1Ethernet->p1024c=0;         /* received packets of 1024-1518 octets. */
    FCC1Ethernet->rfthr=0;         /* received frames threshold */
    FCC1Ethernet->rfcnt=0;          /* received frames count */

    IMM->fcc_regs[FCC1].gfmr = 0x0000000C;
    IMM->fcc_regs[FCC1].dsr = 0xD555;
    IMM->fcc_regs[FCC1].psmr = 0x16400080;
    IMM->fcc_regs[FCC1].fcce = 0xFFFF0000;


   while ((IMM->cpm_cpcr & CPCR_FLG) != READY_TO_RX_CMD);

    IMM->cpm_cpcr = CPCR_INIT_TX_RX_PARAMS |
                CPCR_FCC1_CH |
                CPCR_MCN_FEC |
                CPCR_FLG;

    while ((IMM->cpm_cpcr & CPCR_FLG) != READY_TO_RX_CMD);


    IMM->fcc_regs[FCC1].gfmr |= GFMR_ENR | GFMR_ENT;

}



void FCC2Init()

{

          t_EnetFcc_Pram* FCC2Ethernet;

//    IMM->cpm_mux_cmxfcr = 0x3E250000;

    IMM->cpm_mux_cmxfcr = 0x37253700;
    IMM->cpm_mux_cmxuar = ALL_ZEROS;
    IMM->clocks_sccr   &= 0xFFFFFFF8;


    IMM->pram.serials.fcc_pram[FCC2].riptr = 0x3900;
    IMM->pram.serials.fcc_pram[FCC2].tiptr = 0x3A00;
    IMM->pram.serials.fcc_pram[FCC2].reserved0= 0x0000;
    IMM->pram.serials.fcc_pram[FCC2].rbase = (UWORD)&BD_FCC2->RxBD[0];
    IMM->pram.serials.fcc_pram[FCC2].tbase = (UWORD)&BD_FCC2->TxBD[0];
    IMM->pram.serials.fcc_pram[FCC2].rstate &= 0x00FFFFFF;
    IMM->pram.serials.fcc_pram[FCC2].rstate |= 0x30000000;
    IMM->pram.serials.fcc_pram[FCC2].tstate &= 0x00FFFFFF;
    IMM->pram.serials.fcc_pram[FCC2].tstate |= 0x30000000;
    IMM->pram.serials.fcc_pram[FCC2].mrblr = ENET_MRBLR;

    FCC2Ethernet =
    (t_EnetFcc_Pram * ) &(IMM->pram.serials.fcc_pram[FCC2].SpecificProtocol.e);

    FCC2Ethernet->c_mask   = ENET_C_MASK;   /* Constant MASK for CRC */
    FCC2Ethernet->c_pres   = ENET_C_PRES;   /* CRC Preset */
    FCC2Ethernet->crcec    = ALL_ZEROS;     /* CRC Error Counter */
    FCC2Ethernet->alec     = ALL_ZEROS;     /* Align. Error Counter */
    FCC2Ethernet->disfc    = ALL_ZEROS;     /* Discard Frame Counter */
```

```
    FCC2Ethernet->ret_lim  = ENET_RET_LIM;  /* Retry Limit Threshold */
    FCC2Ethernet->p_per    = ALL_ZEROS;     /* Persistence */
    FCC2Ethernet->gaddr_h  = ALL_ZEROS;     /* Group Addr. Filter 1 */
    FCC2Ethernet->gaddr_l  = ALL_ZEROS;     /* Group Addr. Filter 2 */
    FCC2Ethernet->tfcstat  = ALL_ZEROS;     /* temp BD holder */
    FCC2Ethernet->mflr     = ENET_MFLR;     /* Max Frame Length Reg. */
    FCC2Ethernet->paddr1_h = ENET_PADDR_H;  /* Phys. Addr. 1 (MSB) */
    FCC2Ethernet->paddr1_m = ENET_PADDR;    /* Phys. Addr. 1 */
    FCC2Ethernet->paddr1_l = ENET_PADDR_L;  /* Phys. Addr. 1 (LSB) */
    FCC2Ethernet->iaddr_h  = ALL_ZEROS;     /* Ind. Addr. Filter 1 */
    FCC2Ethernet->iaddr_l  = ALL_ZEROS;     /* Ind. Addr. Filter 2 */
    FCC2Ethernet->minflr   = ENET_MINFLR;   /* Min Frame Length Reg. */
    FCC2Ethernet->taddr_h  = ALL_ZEROS;     /* Temp Address (MSB) */
    FCC2Ethernet->taddr_m  = ALL_ZEROS;     /* Temp Address */
    FCC2Ethernet->taddr_l  = ALL_ZEROS;     /* Temp Address (LSB) */
    FCC2Ethernet->pad_ptr  = IMM->pram.serials.fcc_pram[FCC2].tiptr;/* internal pad pointer,
can be
                                    same as TIPTR if no specific character needed. */
    FCC2Ethernet->cf_type = ALL_ZEROS;      /* reserved, cleared */
    FCC2Ethernet->maxd1   = ENET_MDMA;      /* Max DMA1 Length Reg. */
    FCC2Ethernet->maxd2   = ENET_MDMA;      /* Max DMA2 Length Reg. */
          FCC2Ethernet->octc=0;          /* received octets counter. */
    FCC2Ethernet->colc=0;          /* estimated number of collisions */
    FCC2Ethernet->broc=0;          /* received good packets of broadcast address */
    FCC2Ethernet->mulc=0;          /* received good packets of multicast address */
    FCC2Ethernet->uspc=0;          /* received packets shorter then 64 octets. */
    FCC2Ethernet->frgc=0;          /* as uspc + bad packets */
    FCC2Ethernet->ospc=0;          /* received packets longer then 1518 octets. */
    FCC2Ethernet->jbrc=0;          /* as ospc + bad packets  */
    FCC2Ethernet->p64c=0;          /* received packets of 64 octets. */
    FCC2Ethernet->p65c=0;          /* received packets of 65-128 octets. */
    FCC2Ethernet->p128c=0;         /* received packets of 128-255 octets. */
    FCC2Ethernet->p256c=0;         /* received packets of 256-511 octets. */
    FCC2Ethernet->p512c=0;         /* received packets of 512-1023 octets. */
    FCC2Ethernet->p1024c=0;        /* received packets of 1024-1518 octets. */
    FCC2Ethernet->rfthr=0;         /* received frames threshold */
    FCC2Ethernet->rfcnt=0;         /* received frames count */


    IMM->fcc_regs[FCC2].gfmr = 0x0000000C;
    IMM->fcc_regs[FCC2].dsr = ENET_DSR;
    IMM->fcc_regs[FCC2].psmr = 0x16400080;
    IMM->fcc_regs[FCC2].fcce = 0xFFFF0000;

    while ((IMM->cpm_cpcr & CPCR_FLG) != READY_TO_RX_CMD);

    IMM->cpm_cpcr = CPCR_INIT_TX_RX_PARAMS |
              CPCR_FCC2_CH |
              CPCR_MCN_FEC |
              CPCR_FLG;             /* ISSUE COMMAND */

    while ((IMM->cpm_cpcr & CPCR_FLG) != READY_TO_RX_CMD);
    IMM->fcc_regs[FCC2].gfmr |= GFMR_ENT | GFMR_ENR;
}




void ExtIntHandler(UWORD vector)
```

```
{

  UWORD    ic_sivec;
  UWORD  fcce;
  UBYTE j;


  ic_sivec = IMM->ic_sivec >> 26;  /* sivec interrupt code */
  if (vector != EXT_INT_VECTOR)    /* interrupt NOT external to core */

  {   while (1)
     FlashGP1led();            /* spin here if error is flagged */  };

  if ((ic_sivec != SIVEC_FCC2) && (ic_sivec != SIVEC_FCC1) )      /* interrupt NOT from
FCC 2 */

  {
  return;
   while (1)
      FlashGP1led();            /* spin here if error is flagged */   };

 IMM->fcc_regs[FCC1].fccm = 0x00000000;   // DISABLE INTERRUPTS BY MASKING
 IMM->fcc_regs[FCC2].fccm = 0x00000000;   // DISABLE INTERRUPTS BY MASKING

   if(ic_sivec == SIVEC_FCC2 ) {
                    fcce = IMM->fcc_regs[FCC2].fcce; /* Save off scce */
                    if (fcce & 0x00080000){
                     IMM->fcc_regs[FCC2].fcce = 0x00080000; // CLEAR INTERRUPT
//                           GP1led(ON);   for(j=0;j<0xFF;j++);        GP1led(OFF);
                     processFCC2();
                     IMM->fcc_regs[FCC2].fcce = 0x00080000; // CLEAR INTERRUPT

                     }
/* if (fcce & 0x00040000 ) {
                    FlashGP1led();
//                           for(j=0;j<NUM_RXBDS;j++)BD_FCC2-
>RxBD[j].bd_cstatus&0x7FFF;
                      IMM->fcc_regs[FCC2].fcce = 0x00040000; // CLEAR INTERRUPT
                                                          } */
         }
   else
   {
          fcce = IMM->fcc_regs[FCC1].fcce; /* Save off scce */
                    if (fcce & 0x00080000){
          IMM->fcc_regs[FCC1].fcce = 0x00080000; // CLEAR INTERRUPT
                    processFCC1();
                                    //          GP0led(ON);   for(j=0;j<0xFF;j++);
          GP0led(OFF);

                           // IMM->fcc_regs[FCC1].fcce = 0x00080000; // CLEAR INTERRUPT


          }

                   /*f (fcce & 0x00040000 ) {
                    FlashGP1led();
                    //         for(j=0;j<NUM_RXBDS;j++)BD_FCC1-
>RxBD[j].bd_cstatus&0x7FFF;
                      IMM->fcc_regs[FCC1].fcce = 0x00040000; // CLEAR INTERRUPT
```

```
                                                    } */

            }

   IMM->fcc_regs[FCC1].fccm = 0x000C0000;   // ENABLE INTERRUPTS
   IMM->fcc_regs[FCC2].fccm = 0x000C0000;   // ENABLE INTERRUPTS

} /* end ExtIntHandler */



void processFCC1(){

UWORD wait;
UBYTE  size;
UHWORD txIndex;
UHWORD rxIndex ;
UHWORD i;
UHWORD t,r;

 //GP1led(ON);
          fcc1:
   IMM->fcc_regs[FCC1].fcce = 0x00080000; // CLEAR INTERRUPT
   size=1;
          i=0;
          t=0;
          r=0;
   while(IMM->fcc_regs[FCC2].fcce & 0x00100000)FlashGP1led();
   rxIndex= wF1;

   if((BD_FCC1->RxBD[rxIndex%NUM_RXBDS].bd_cstatus & 0x8400)==0x0400){  //CHECK
IF THERE IS SOMETHING

     while (!(BD_FCC1->RxBD[rxIndex%NUM_RXBDS].bd_cstatus&0x0800))
               {rxIndex++;size++;if(rxIndex==NUM_RXBDS)rxIndex=0;}

               rxIndex=rxIndex%NUM_RXBDS;

                    txIndex=tF2;

                                    if(rxIndex<(size-1))
            LOGS[logptr]->from=NUM_TXBDS-size+1+rxIndex;
            else
            LOGS[logptr]->from=rxIndex-size+1;

//            LOGS[logptr]->to=txIndex;
            LOGS[logptr]->to=fcc1buffer[rxIndex]->INFO[33];
//            LOGS[logptr]->length=size;

                                    LOGS[logptr]->length=BD_FCC1-
>RxBD[rxIndex].bd_length;
                                    LOGS[logptr]->sequence=((fcc1buffer[rxIndex]-
>INFO[39])<<24)|((fcc1buffer[rxIndex]->INFO[40])<<8)|((fcc1buffer[rxIndex]-
>INFO[41])<<12)|((fcc1buffer[rxIndex]->INFO[42]));



            LOGS[logptr]->fcc=1;
            logptr++;
```

```
            logptr=logptr%LOG_SIZE;
            txIndex+=(size-1)%NUM_TXBDS;
            tF2=(tF2+size)%NUM_TXBDS;
            wF1=(wF1+size)%NUM_RXBDS;


          for(i=0;i<size;i++)    {

          if(txIndex<i)
          t=NUM_TXBDS-(i-txIndex);
          else
          t=txIndex-i;

          if(rxIndex<i)
          r=NUM_RXBDS-(i-rxIndex);
          else
          r=rxIndex-i;


              BD_FCC2->TxBD[t].bd_addr  =BD_FCC1->RxBD[r].bd_addr;

            if(i!=0)
  //          if(BD_FCC1->RxBD[r].bd_length!=1518)
                      BD_FCC2->TxBD[t].bd_length=BD_FCC1->RxBD[r].bd_length;
                  //       else
                    //      BD_FCC2->TxBD[t].bd_length=1500;
                      else
                      BD_FCC2->TxBD[t].bd_length=BD_FCC1->RxBD[r].bd_length-
(ENET_MRBLR*(size-1));

                     if(i==0)
                      if(t!=NUM_TXBDS-1)
                                      BD_FCC2->TxBD[t].bd_cstatus=0xCC00;
                                      else
                                BD_FCC2->TxBD[t].bd_cstatus=0xEC00;
           else

            if(t!=NUM_TXBDS-1)
                                BD_FCC2->TxBD[t].bd_cstatus=0xC400;
                                   else
                                BD_FCC2->TxBD[t].bd_cstatus=0xE400;



                        if(r==NUM_RXBDS-1)
                 BD_FCC1->RxBD[r].bd_cstatus = 0xB000;
                  else
                    BD_FCC1->RxBD[r].bd_cstatus = 0x9000;

                }
                c_total_1++;

                                if(IMM->fcc_regs[FCC1].fcce & 0x00080000)
                                goto fcc1;

                                }

                        //GP1led(OFF);
                 if(IMM->fcc_regs[FCC2].fcce & 0x00080000) processFCC2();
```

```
 }



void processFCC2()
 {

UWORD wait;
UBYTE  size;
UHWORD txIndex;
UHWORD rxIndex ;
UHWORD i;
UHWORD t,r;

 //GP1led(ON);
          fcc1:
   IMM->fcc_regs[FCC2].fcce = 0x00080000; // CLEAR INTERRUPT
   size=1;
          i=0;
          t=0;
          r=0;
   while(IMM->fcc_regs[FCC1].fcce & 0x00100000)FlashGP1led();
   rxIndex= wF2;

   if((BD_FCC2->RxBD[rxIndex%NUM_RXBDS].bd_cstatus & 0x8400)==0x0400){  //CHECK
IF THERE IS SOMETHING

     while (!(BD_FCC2->RxBD[rxIndex].bd_cstatus&0x0800))
               {rxIndex++;size++;if(rxIndex==NUM_RXBDS)rxIndex=0;}


                    txIndex=tF1;

              if(rxIndex<(size-1))
               LOGS[logptr]->from=NUM_TXBDS-size+1+rxIndex;
               else
               LOGS[logptr]->from=rxIndex-size+1;

//             LOGS[logptr]->to=txIndex;
               LOGS[logptr]->to=fcc2buffer[rxIndex]->INFO[33];
//              LOGS[logptr]->length=size;
                                    LOGS[logptr]->length=BD_FCC2-
>RxBD[rxIndex].bd_length;
               LOGS[logptr]->fcc=2;
               logptr++;
               logptr=logptr%LOG_SIZE;
               txIndex+=(size-1)%NUM_TXBDS;
               tF1=(tF1+size)%NUM_TXBDS;
               wF2=(wF2+size)%NUM_RXBDS;


            for(i=0;i<size;i++)    {



            if(txIndex<i)
            t=NUM_TXBDS-(i-txIndex);
```

```
        else
        t=txIndex-i;

        if(rxIndex<i)
        r=NUM_RXBDS-(i-rxIndex);
        else
        r=rxIndex-i;


      /*    if(size>rxIndex){

                        LOGS[logptr]->fcc=2;


        }*/


            BD_FCC1->TxBD[t].bd_addr  =BD_FCC2->RxBD[r].bd_addr;

          if(i!=0)
                    BD_FCC1->TxBD[t].bd_length=BD_FCC2->RxBD[r].bd_length;
                    else
                    BD_FCC1->TxBD[t].bd_length=BD_FCC2->RxBD[r].bd_length-
(ENET_MRBLR*(size-1));

                   if(i==0)
                    if(t!=NUM_TXBDS-1)
                                BD_FCC1->TxBD[t].bd_cstatus=0xCC00;
                                else
                            BD_FCC1->TxBD[t].bd_cstatus=0xEC00;
          else

           if(t!=NUM_TXBDS-1)
                            BD_FCC1->TxBD[t].bd_cstatus=0xC400;
                                else
                           BD_FCC1->TxBD[t].bd_cstatus=0xE400;



                    if(r==NUM_RXBDS-1)
            BD_FCC2->RxBD[r].bd_cstatus = 0xB000;
             else
               BD_FCC2->RxBD[r].bd_cstatus = 0x9000;

            }


            c_total_2++;
                            if(IMM->fcc_regs[FCC2].fcce & 0x00080000)
                            goto fcc1;

                            }

            //          GP1led(OFF);
            if(IMM->fcc_regs[FCC1].fcce & 0x00080000) processFCC1();

        }
```

```
/// VERY LOW LEVEL!!!! ////

//void restartTX(


UHWORD BDRxError(UHWORD bd_cstatus)
{
  if (bd_cstatus & BD_RX_ERROR)
        return TRUE;
  else
    return FALSE;
} /* end BDRxError */


UHWORD LastBD(UHWORD bd_cstatus)
{
  if (bd_cstatus & 0x0800)
    return TRUE;
  else
    return FALSE;
} /* end LastBD */



UHWORD BDEmpty(UHWORD bd_cstatus)
{
  if (bd_cstatus & 0x8000)
     return TRUE;
  else
    return FALSE;
} /* end BDEmpty */


void enableFCC( void ) {

BCSR *bcsr2_;
UWORD evren;

  bcsr2_ = (BCSR *)(IMM->memc_regs[1].br & 0xFFFF8000);
  bcsr2_->bcsr1 &= ~(1<<27);
  bcsr2_->bcsr1 |= (1<<25);
  bcsr2_->bcsr1 |= (1<<24);

}

void GP0led(UHWORD setting)
{

BCSR *csr;

  csr = (BCSR *)(IMM->memc_regs[1].br & 0xFFFF8000);
  if (setting){
    csr->bcsr0 &= ~GP0_LED;   /* turn on LED, active low */
}
  else
    csr->bcsr0 |= GP0_LED;    /* turn off LED, active high */
} /* end GP0led */
```

```
void GP1led(UHWORD setting)
{

BCSR *csr;
  csr = (BCSR *)(IMM->memc_regs[1].br & 0xFFFF8000);
  if (setting)
    csr->bcsr0 &= ~GP1_LED;   /* turn on LED, active low */
  else
    csr->bcsr0 |= GP1_LED;    /* turn off LED, active high */
} /* end GP1led */


void FlashGP1led()

{

UBYTE  ii;
UWORD jj;

  for (ii = 0; ii<20; ii++)
  {
    GP1led(ii%2);  /* Turn on every other time through the loop */
     for (jj=0; jj < 100000; jj++);    /* Wait */
  }

  GP1led(0);  /* LED off */
} /* end FlashGP1led */




void fethReset(){
UWORD j;
BCSR *bcsr2_;
 bcsr2_ = (BCSR *)(IMM->memc_regs[1].br & 0xFFFF8000);
 GP0led(ON);
 bcsr2_->bcsr1 &= ~(1<<26); //FETHRST
 for( j = 0;j < 0xFF ; j++ );
 bcsr2_->bcsr1 |= (1<<26); //FETHRST
 for( j = 0;j < 0xFFFFF ; j++ ); // wait 50ms at least
 GP0led(OFF);
}


void mdioSend(UBYTE REGADDR,UWORD PHYADDR,UHWORD DATA){

UBYTE i;
UWORD j;
UWORD mask = 1<<15;
UBYTE SFDOP=0x05;
UBYTE TA = 0x02;
UWORD TEMP=0;
UWORD bak=0;
UWORD wait = 0xFF;// Clock Frequency should be lower than 2.5Mhz
UWORD mdio = 0x00400000; //PC9
UWORD mdc  = 0x00200000; //PC10
```

```
//PREAMBLE

GP1led(ON);
for ( i= 0;i<32 ;i ++){
IMM->io_regs[PORTC].pdat |= mdio;
        for( j = 0;j < wait; j++ );


        IMM->io_regs[PORTC].pdat |= mdc;


        for( j = 0;j < wait ; j++ );
        IMM->io_regs[PORTC].pdat &= ~mdc;


}


TEMP=(TA)|(REGADDR<<2)|(PHYADDR<<7)|(SFDOP<<12);
for ( i= 0;i<16 ;i ++){
if(mask & TEMP){
 IMM->io_regs[PORTC].pdat |= mdio;

 }
else {
 IMM->io_regs[PORTC].pdat &= ~mdio;
 //GP1led(OFF);
 }

for( j = 0;j < wait ; j++ );
IMM->io_regs[PORTC].pdat |= mdc;
//GP0led(ON);


for( j = 0;j < wait ; j++ );
IMM->io_regs[PORTC].pdat &= ~mdc;
//GP0led(OFF);

TEMP = (UWORD) TEMP << 1;
        }

TEMP=DATA;
for ( i= 0;i<16 ;i ++){
if(mask & TEMP){
 IMM->io_regs[PORTC].pdat |= mdio;
 }
 else {
 IMM->io_regs[PORTC].pdat &= ~mdio;
 }

for( j = 0;j < wait ; j++ );
IMM->io_regs[PORTC].pdat |= mdc;


for( j = 0;j < wait ; j++ );
IMM->io_regs[PORTC].pdat &= ~mdc;
GP0led(OFF);

TEMP = (UWORD) TEMP << 1;
```

```
                        }
GP1led(OFF);
 }

void SetEEinMSR(void)
{

/* Kesme Özelliğini Açmak için */
    asm(mfmsr  r3);
    asm(andi.r3,r3,0xFFFFFFBF);
    asm(ori  r3,r3,0xA000);
    asm(mtmsr  r3);

} /* end SetEEinMSR */
```

tez cd : ek1.zip veya klasör EK1
http://www.ulusoy.net/evren/mpc8260/ek1.zip

Ek2:

tez cd: ek2.zip veya klasör EK2
http://www.ulusoy.net/evren/mpc8260/fcc_enet.c

Ek3:

tez cd: ek3.zip veya klasör EK3
http://www.ulusoy.net/evren/mpc8260/busybox.config

Ek4:

tez cd: ek4.zip veya klasör EK4
http://www.ulusoy.net/evren/mpc8260/etc_files.tar

ÖZGEÇMİŞ

2004 yılında Işık Üniversitesi Elektronik Mühendisliği Bölümünden ikincilik derecesi ile mezun olmuştur. Aynı üniversitenin çift anadal programı altında Bilgisayar Mühendisliği Bölümünü tamamlamıştır. 2002-2003 eğitim yılında Işık Üniversitesinin sağladığı burs ile London SouthBank Üniversitesi Telekomünikasyon ve Bilgisayar Ağları Bölümünde iki dönem eğitim görmüştür. 2004 yılından itibaren Turkcell İletişim Hizmetlerinde sistem yöneticisi olarak görev yapmaktadır.