

YILDIZ TEKNİK ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

Çevresel Mod., Bul., Ver., Bak. ve
Sist. İlg. Dia. Testler

YÜKSEK LİSANS TEZİ

Esin Bingöl

1990

YILDIZ TEKNİK ÜNİVERSİTESİ
KÜTÜPHANE DOKÜMANTASYON
DAİRE BAŞKANLIĞI

Kot	: R 368 57
Alındığı Yer	: Fen Bil. Ens.
Tarih	: 13.4.95
Fatura	:
Fiyatı	: 50.000.-
Ayniyat No	: 1-4
Kayıt No	: 51035
UDC	:
Ek	:

YILDIZ ÜNİVERSİTESİ

D.E. No 49895

YILDIZ ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR BİLİMLERİ ANA BİLİM DALI

REKLAM ANONSU PROJESİ

ÇEVRESEL MODÜLLERDE BULUNAN VERİNİN BAKIMI
VE SİSTEME İLİŞKİN DİAGNOSTİK TESTLER

YÜKSEK LİSANS TEZİ

HAZIRLAYAN
ESİN BİNGÖL
ELEKTRONİK ve HABERLEŞME MÜH.

YÖNETEN
PROF. DR. M. YAHYA KARSLIĞIL



İSTANBUL, 1990

İÇİNDEKİLER

ÖNSÖZ

ÖZET

1.0	REKLAM ANONSU PROJESİNİN TANITIMI	2
1.1	Reklam Anonsu Projesinin Amacı	2
1.2	Türkiye'deki Ring Back Tone Yapısı	3
1.3	Reklam Anonsu Projesinin Kapsamı	5
2.0	DMS (DIGITAL MULTIPLEX SYSTEM) AİLESİ	8
2.1	Merkezi Kontrol Birimi (CCC)	12
2.1.1	Merkezi İşlem Birimi (CPU)	12
2.1.2	Merkezi Mesaj Denetleyicisi (CMC)	14
2.1.3	Program Belleği (PS)	14
2.1.4	Veri Belleği (DS)	14
2.2	Ağ Modülleri (NM's)	15
2.3	Giriş/Çıkış Denetleyicisi (IOC)	16
2.4	Çevresel Modüller (PM's)	17
2.4.1	Çevresel Modüllerin Tanıtımı	17
2.4.2	XPM'de Yazılım	18
2.4.2.1	XPM Yazılımının Genel Yapısı	18
2.4.2.2	XPM Yazılım Modülleri	19
2.4.2.3	XPM Pascal Modülünün Formatı	20
2.4.2.4	USES Listesi	21
2.4.2.5	XPM Yazılımında Task Yapısı	23
2.4.2.6	XPM Yükünün Derlenmesi, Bağlanması ve Yüklenmesi	23
2.4.3	XPM'de Güvenilirlik	31
3.0	REKLAM ANONSU PROJESİ DONANIM YAPISI	34
4.0	REKLAM ANONSU PROJESİ YAZILIM YAPISI	42
4.1	Kullanıcıya Bakım Amacı İle Sağlanan Komutlar	42
4.1.1	Disable Komutu	42
4.1.2	Enable Komutu	43
4.1.3	Dump Komutu	44
4.1.4	Test Komutu	45
4.1.5	Query Komutu	45
4.2	XPM'deki Verinin Bakımı	46
4.2.1	XPM'in CC Ve Eş Birimle Mesajlaşması	46
4.2.1.1	XPM'den CC'ye Mesaj Gönderimi	46
4.2.1.2	CC Mesajlarının Değerlendirilmesi	51
4.2.1.3	Eş Birime Mesaj Gönderimi	54
4.2.2	CC'den Gelen İsteklerin Yerine Getirilmesi	54
4.2.2.1	CA'nın Disable Edilmesi	54
4.2.2.2	CA'nın Enable Edilmesi	55
4.2.2.3	CA'nın Dump Edilmesi	55
4.2.2.4	CA Bölgesinin Test Edilmesi	56
4.2.2.5	CA'nın Dump Sonrası Test Edilmesi	56
4.2.2.6	CA'nın Query Edilmesi	57
4.3	Diagnostik Testler	57
4.3.1	CAM Kartının Test Edilmesi	58
4.3.1.1	Yapılan Testin Amacı	58

4.3.1.2	Testin Yapılışı	58
4.3.2	CAM Belleğinin Test Edilmesi	60
4.3.2.1	Yapılan Testin Amacı	60
4.3.2.2	Testin Yapılışı	60
4.3.3	Reklam Anonsu Verisinin Test Edilmesi	62
4.3.3.1	Yapılan Testin Amacı	62
4.3.3.2	Testin Yapılışı	64
4.4	Yazılım Modülleri	70
4.4.1	Veri Bakım Modülleri	73
4.4.2	Diagnostik Test Modülleri	110
5.0	KISALTMALAR	144
6.0	KAYNAKLAR	146

ÖZGEÇMİŞ

Bir bölüm de tarafımızca gerçekleştirilen Reklam Anonsu Projesi'nin yürütülmesinde yardımlarını ve desteklerini esirgemeyen İktisadi İstatistik Profesörü Doç. Dr. M. Yahya Karşıoğlu'na, proje başlatmamızda Sayın Doç. Dr. E. ve aralık grubu yöneticimiz Sayın Doç. Dr. Çiğdem Çiğdem'e bir teşekkür.

İLBİZ ÜNİVERSİTESİ

ERİN BİNGÖL

İSTANBUL, 1996

ÖNSÖZ

Günümüzde, sanayi ve ticaretin gelişimiyle, paralel işlevler üstlene kurumlar arasında doğan rekabetin sonucunda önem kazanan reklamcılık sektörü, halka ulaşabilme alanında büyük bir uğraş vermektedir. Geçen gün, teknolojinin kitle iletişim araçlarına kazandırdığı yen yapılar, reklamcılık sektörüne de derin boyutlarda yansımaktadır.

Televizyon ve yayıncılığın yanı sıra, günlük hayatımızda vazgeçilmezliğini yadsıyamayacağımız telefonlarımız da, PTT'nin talep üzerine NETAŞ Araştırma ve Geliştirme Bölümünde gerçekleştirile Reklam Anonsu Projesi ile reklamcılık sektörüne kazandırılan bir kitle iletişim aracı haline gelmiştir.

Bir bölümü de tarafımdan gerçekleştirilen Reklam Anonsu Projesini hazırlanmasında yardımlarını ve desteklerini esirgemeyen tezim yöneten hocam Sayın Prof. M. Yahya Karşlıgil'e, proje başkanımız Sayı Yaşar Güngör'e ve proje grubu menejerimiz Sayın Mesut Çığay' teşekkürü bir borç bilirim.

YILDIZ ÜNİVERSİTESİ

ESİN BİNGÖL

İSTANBUL, 1990

ÖZET

PTT tarafından yapılan başvuru ile, NETAŞ Araştırma ve Geliştirme bölümünde gerçekleştirilen Reklam Anonsu Projesinin ' Çevrese Modüllerde Reklam Anonsu Verisinin Bakımı Ve Sistemin Tümüne İlişki Diagnostik Testler' başlıklı bölümü, sunulan tezin içeriğın oluşturmaktadır.

Reklam Anonsu Projesinin uygulamaya konulması ile, Türk Telefo ağında yer alan bir abone, yine bu ağda yer alan diğer bir aboney aradığı zaman, duymakta olduğu zil sesi ve sessizlikte, sessizli yerine reklam anonsu duyacaktır.

Bu reklam anonslarının süresi, Türkiye'de mevcut olan ring back ton kadansı yapısı nedeniyle dört saniyedir. Aranılan abonenin uzun sür telefonu açmaması sonucunda duyulabilecek farklı reklam anonsu sayısı ise, geliştirilen donanım gereği beş ile sınırlandırılmıştır.

Projenin uygulamaya konulması ile sistemin bakımını üstlenecek ola kullanıcı, kendisine sağlanan komutlarla, bulunduğu telefo santralindeki bir aboneyi arayan diğer bir abonenin dinleyebilmes için reklam anonslarını ilgili bölüme yukleyebilecek, yuklu buluna reklam anonslarından istediklerini geçici bir süre ile aboneler verilmesini önleyebilecek ve istediği zaman bu reklam anonslarını tekrar dinlenmesini sağlayabilecektir.

Ayrıca, kullanıcı, yine kendisine sağlanan komutlarla, bu proje ici gerçekleştirilen kartın donanımını ve kartta bulunan verini doğrulugunu test edebilecektir.

ARSIBACI

The Commercial Announcement Project requested by the Turkish PTT has been realized in the NETAS Research and Development Department. The content of this thesis is the part of the Commercial Announcement Project called 'The Maintenance of The Commercial Announcement Data in The Peripheral Moduls and The Diagnostics on The Whole of The System'

After this project begins to be applied, if the subscriber in the Turkish Telephone Network telephones another subscriber in the same network, he will hear the various commercial announcements (CA's) instead of the silence in the ring back tone.

Because of the ring back tone cadence in Turkey, the period of the CA's is four seconds. The number of the different CA's which will be heard when the subscriber who is telephoned by another subscriber does not pick up the receiver is five. The reason of this restriction is the hardware limitations of the card which has been developed for this project.

The craftsperson will be able to load the CA's to the related units in order to give them to the subscribers who telephone the subscribers connected this telephone exchange office, prevent the giving of CA's to the subscribers for a certain time which can be determined by himself, and permit them to be given again by using the supplied commands.

In addition, the craftsperson will be able to test the card which includes the CA's and the CA data in this card.

BÖLÜM 1

REKLAM ANONSU PROJESİNİN TANITIMI

1.1 Reklam Anonsu Projesinin Amacı

1.2 Türkiye'deki Ring Back Tone Yapısı

1.3 Reklam Anonsu Projesinin Kapsamı

1.0 REKLAM ANONSU PROJESİNİN TANITIMI

Giriş niteliğinde olan bu bölümde, Reklam Anonsu Projesinin tanıtımı ve uygulanmaya başlanması ile geçilecek sistemin anlatımı yer almaktadır. Yeni sistemin kavranabilmesi için, ayrıca halen Türkiye'de mevcut olan yapıya değinilecektir.

1.1 REKLAM ANONSU PROJESİNİN AMACI

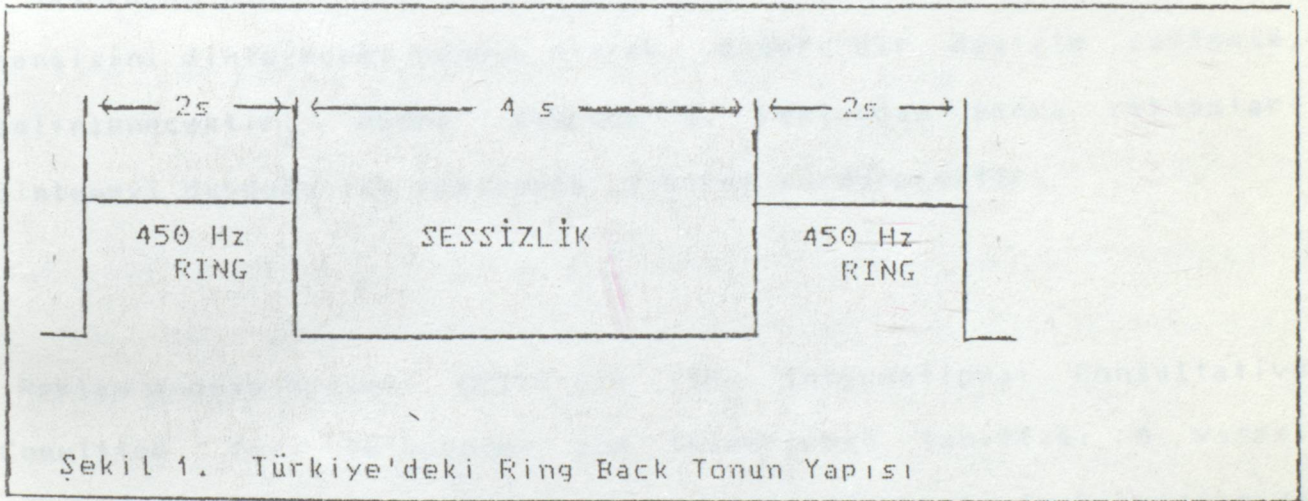
Reklam Anonsu (Commercial Announcement) projesi, PTT tarafından yapılan başvuru ile METAŞ (Northern Electric Telekomünikasyon A.Ş.) Araştırma ve Geliştirme Bölümünde gerçekleştirilmiştir.

Bu projenin amacı, Türk telefon ağında yer alan abonelere, diğer bir aboneye çağrı yaptıkları zaman, bu abonenin telefonu çalarken aldıkları zil sesi ve sessizlikte, yani ring back tone'da, sessizlik yerine reklam anonsu duyurmaktır.

1.2 TÜRKİYE'DEKİ RİNG BACK TONE YAPISI

Türkiye'de mevcut olan telefon sisteminde, diğer ülkelerde de olduğu gibi, ring back tone belirli bir frekans ve kadansa sahiptir. Kadans, ring back tone'da ring ve sessizlik, diğer bir deyişle ON ve OFF durumlarının süresel yapısıdır. Ring back tone, Türkiye'de iki saniye süreli 450 Hz.'lik bir ring ve dört saniye süreli sessizlikten oluşmaktadır.

Bu ring back tone yapısı Şekil 1'de gösterilmiştir.



Şekil 1. Türkiye'deki Ring Back Tonun Yapısı

Reklam anonsu projesinin uygulamaya konulması ile ring back tone, iki saniye süreli 450 Hz.'lik bir ring ve dört saniye süreli anonslar içerecektir.

Bir abone diğ er bir aboneye çağ ırı yaptı ğ ı takdirde, bu abonenin telefonu ç alarken, ilk olarak iki saniye sũ reli bir ring daha sonra bir reklam anonsu, daha sonra tekrar iki saniye sũ reli bir ring ve ardından ilkinden farklı bir reklam duyacak, bu anonslu ring back tone diğ er abonenin telefonu aç ıř ına kadar sıralı olarak tekrarlanacaktır. Bilindiğ i gibi, ring back tone arayan aboneye, aranılan abonenin bağı lı bulunduğ u yerel santralden verilmektedir. Duyulabilecek olan farklı reklam anonsu sayısı ise tasarlanan donanım gereğ i beř ile sınırlandırılmıř tır.

Abonenin sıralı olarak verilen beř reklam anonsundan ilk olarak hangisini dinleyeceğ i random olarak, diğ er bir deyiř le rastgele, belirlenecektir. Abone duyduğ u 5. reklamdan sonra reklamları dinlemeyi duyduğ u ilk reklamdan itibaren sũ rdũ recektir.

Reklam anonsu projesi, CCITT'nin (The International Consultative Committee for Telephones and Telegraphs) tanıttı ğ ı A yasası dönüşũ mũ ndũ kullanan ve Tũ rkiye ile aynı ring back tone kadansına sahip olan diğ er yabancı ũ lkelerde de uygulamaya konulabilecektir. Ancak farklı dönüşũ m yasası yada farklı ring back tone kadansına sahip ũ lkelerde bu projenin uygulanabilmesi iç in bazı donanım ve yazılım değ iř ikliklerinin yapılması gerekmektedir.

Bilindiğ i gibi, darbe kod modũ lasyon sistemleri PCM (Pulse Code Modulation) tũ mũ yle sayısal sistemlerdir. Bu sistemlerde bir iř aret, iletilmeden ō nce ō rneklenir, kuantalanır ve bu kuantalama sonucunda elde edilen değ erler ikili olarak kodlanır. Iř aret genliğ inin ç ok

küçük olduğu, sıfır civarında bulunduğu bölgede işaret/gürültü oranı azalır. Ancak bu bölgede gürültünün işareten ayırd edilmesi zorlaştığından gürültü, işaret gibi, kodlanıp karşı tarafa gönderilir. Bunu önlemek amacıyla, A ve μ olmak üzere, düzgün olmayan kuantalama yapan iki ayrı dönüşüm yasası bulunmaktadır. Her ikisinde de işaretin küçük genlikli bölgeleri doğrusal, genliğin arttığı bölgeleri ise sıkıştırıp kuantalanarak gürültü bastırılır. Türkiye'de A yasası, Kuzey Amerika ve Japonya'da μ yasası dönüşümü kullanılmaktadır.

1.3 REKLAM ANONSU PROJESİNİN KAPSAMI

Türkiye'de NETAŞ'ın sunduğu sayısal bir santral olan DMS-100/200 (Digital Multiplex System) ailesinin günümüzde sahip olduğu yapı, reklam anonsu projesi için gerekli olan donanımı içermemektedir. Bu nedenle, Reklam Anonsu Belleği (CAM : Commercial Announcement Memory) adı verilen bir donanımı ve bu donanımı işler hale getirecek yazılımı geliştirmek, Reklam Anonsu projesinin kapsamını oluşturmaktadır. Her DMS'in en azından dört CAM kartına sahip olacağı ve başlangıçta ülke genelinde yaklaşık olarak 500 kart bulunacağı tahmin edilmektedir.

Reklam anonsu projesinin yazılımı, birbirine bağımlı beş ayrı bölüm halinde, beş ayrı tasarımcı tarafından gerçekleştirilmiştir. Bu projenin içerdiği bölümler şunlardır :

- * Çevresel modüllerde, reklam verisinin bakımı ve projenin yazılım ve donanımına ilişkin diagnostik testleri sağlayan yazılım.
- * Merkezi kontrol biriminde, reklam verisinin bakımını sağlayan yazılım.
- * Reklam verisinin ring back tone'a konulmasını sağlayan yazılım.
- * Aboneye, aranıldığında arayan aboneye reklam anonslu ring back tone verilmemesi seçeneğini sağlayan yazılım.
- * Kullanıcıya CAM kartındaki verileri ve yapılan bazı işlemlerin sonuçlarını gösterimini gerçekleyen yazılım.

Sunulan tezin kapsadığı yazılım, çevresel modüllerde bulunan reklam anonsu verisinin bakımını sağlayan, diğer bir deyişle bu verinin kullanıcı tarafından işlenmesine olanak veren, ve projenin tümüne ilişkin diagnostik testleri içeren bölümdür. Ancak tüm bölümlerin birbirine bağımlı olması nedeni ile diğer üç bölüme de, gerek duyulduğunda, kısaca değinilecektir.

BÖLÜM 2

DMS (DIGITAL MULTIPLEX SYSTEM) AİLESİ

2.1 Merkezi Kontrol Birimi (CCC)

2.1.1 Merkezi İşlem Birimi (CPU)

2.1.2 Merkezi Mesaj Denetleyicisi (CMC)

2.1.3 Program Belleği (PS)

2.1.4 Veri Belleği (DS)

2.2 Ağ Modülleri (NM's)

2.3 Giriş/Çıkış Denetleyicisi (IOC)

2.4 Çevresel Modüller (PM's)

2.4.1 Çevresel Modüllerin Tanıtımı

2.4.2 XPM'de Yazılım

2.4.2.1 XPM Yazılımının Genel Yapısı

2.4.2.2 XPM Yazılım Modülleri

2.4.2.3 XPM PASCAL Modülünün Formatı

2.4.2.4 USES Listesi

2.4.2.5 XPM Yazılımında Task Yapısı

2.4.2.6 XPM Yükünün Derlenmesi, Bağlanması ve Yüklmesi

2.4.3 XPM'de Güvenilirlik

2.0 DMS (DIGITAL MULTIPLEX SYSTEM) AİLESİ

DMS-100/200 ailesi, yerel, şehirler arası yada her iki türde uygulamalara hizmet veren bir iletişim sistemidir. DMS-100, aboneler arasındaki iletişimi, kurulan haberleşme ağı ve ve sayısal anahtarlama ile gerçekleştirmektedir. Sayısal anahtarlama, lojik devreler ve bilgisayar teknolojilerinden yararlanılmaktadır.

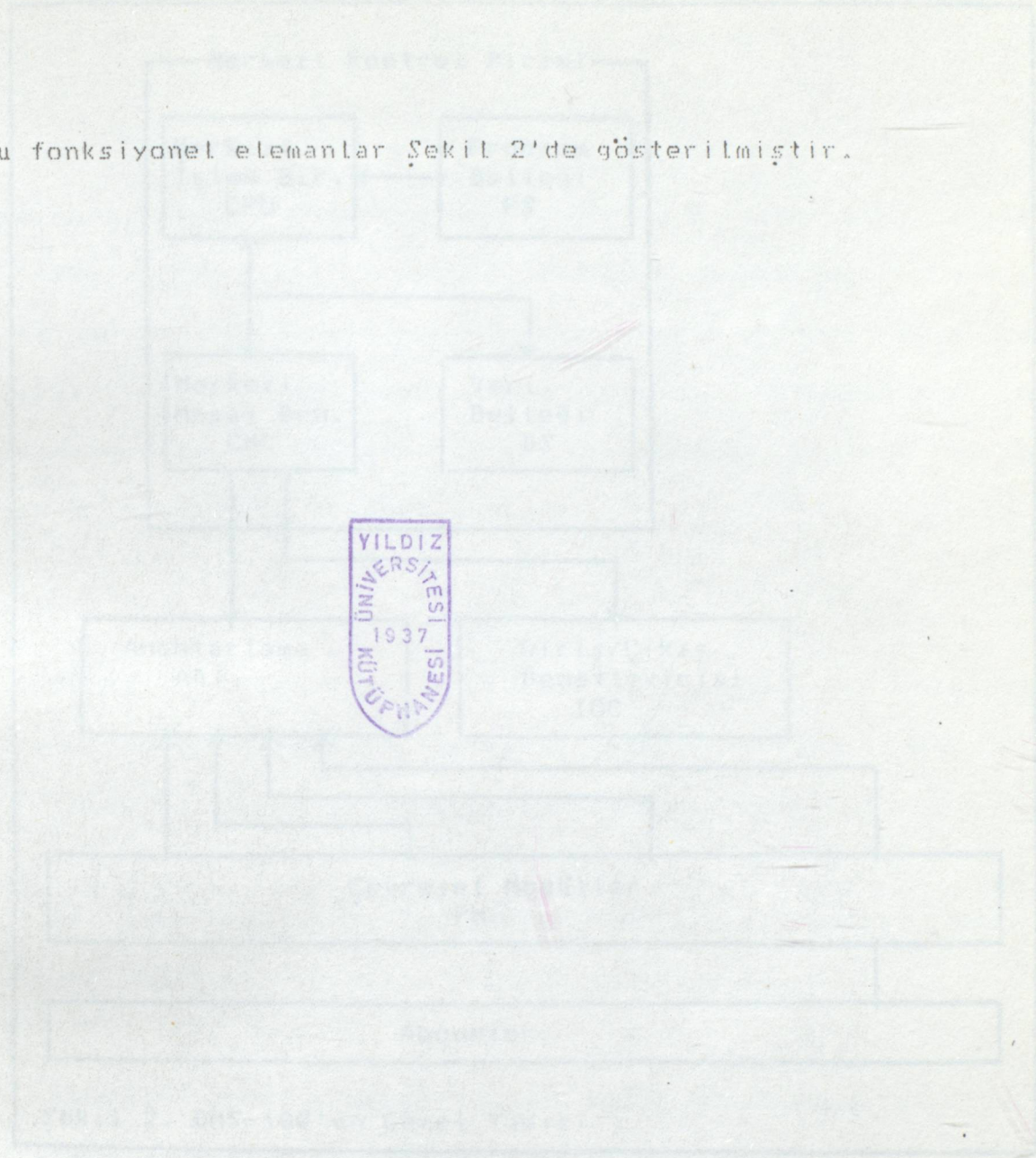
DMS-100/200 ailesinin yazılım ve donanımında önemli ölçüde modüller tasarım tekniklerinden yararlanılmıştır. Modüler yapısı nedeni ile, bu karmaşık sistem, birbirine arabağlaşımlarla bağlı basit fonksiyonel birimler halinde meydana getirilebilmiştir. Böylece sistemdeki çeşitli birimler, diğer birimlerin işleyişini etkilemeden, uygun modül ve arabağlaşımlar kullanılarak sisteme bağlanabilir, sistemden çıkarılabilir, değiştirilebilir yada geliştirilebilir.

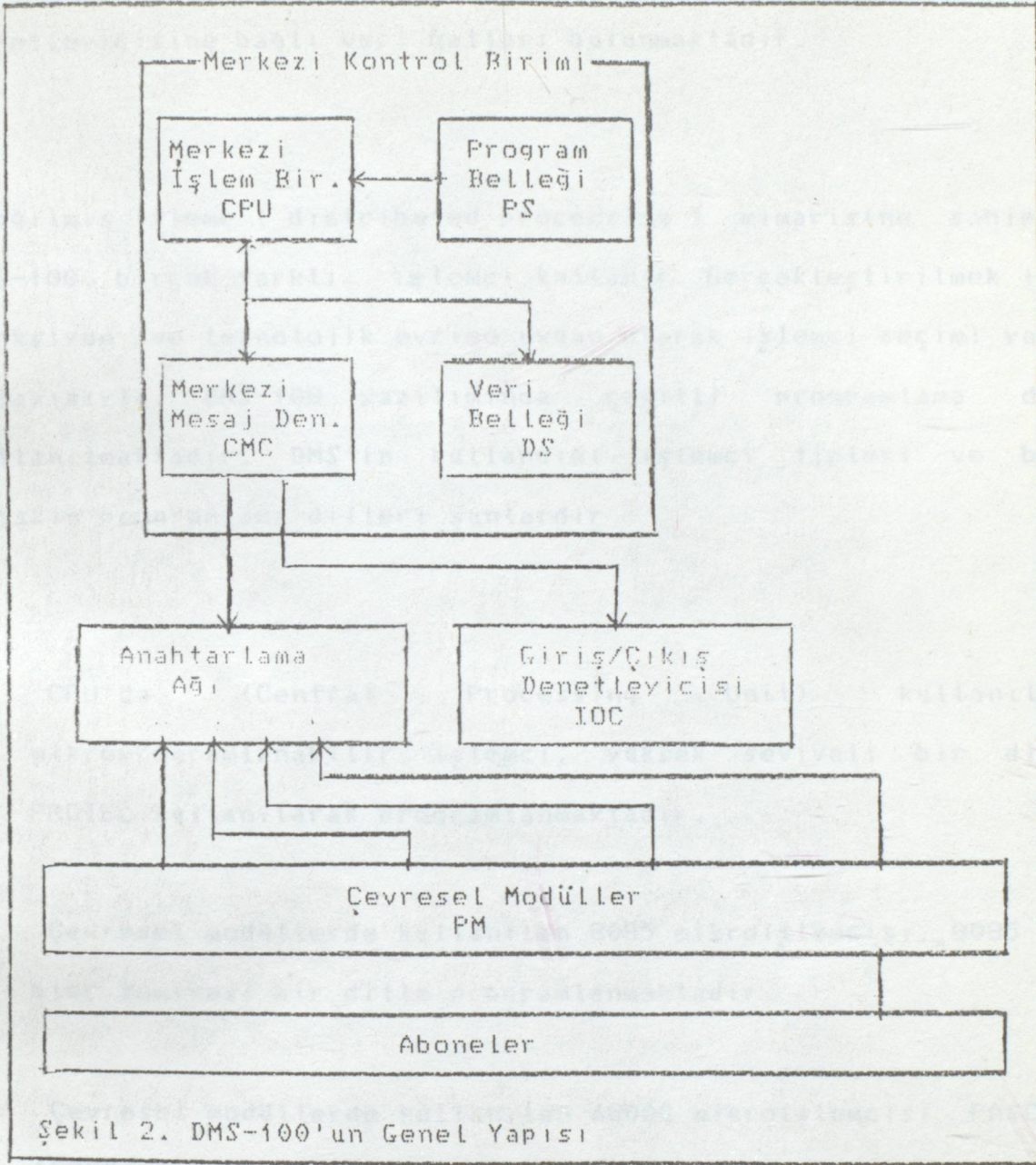
DMS-100/200 sistemi dört ana fonksiyonel eleman içerir :

1. Merkezi Kontrol Birimi
2. Ağ (Network) Modülleri
3. Giriş/Çıkış Denetleyicisi

4. Çevresel Modüller

Bu fonksiyonel elemanlar Şekil 2'de gösterilmiştir.





Merkezi kontrol birimi, çevresel modüller ve ağ modülleri arasındaki bağlaşım, sayısal, seri veri hatları ile gerçekleştirilir. Buna ek olarak, manyetik teyp sürücüler (Magnetic Tape Drivers), disk sürücüler (Disk Drivers), veri hatları, video gösterim birimleri (VDU: Video Display Unit) ve yazıcılar (printer) gibi ana bilgisayarlar (mainframe) ile ilgili cihazlara ve bu

cihazlardan veri akışını gerçekleştirmek üzere giriş/çıkış denetleyicisine bağlı veri hatları bulunmaktadır.

Dağılmış işleme (distributed processing) mimarisine sahip olan DMS-100, birçok farklı işlemci kullanır. Gerçekleştirilmek istenen fonksiyon ve teknolojik evrime uygun olarak işlemci seçimi yapılır. Dolayısıyla, DMS-100 yazılımında çeşitli programlama dilleri kullanılmaktadır. DMS'in kullandığı işlemci tipleri ve bunlara ilişkin programlama dilleri şunlardır :

- * CPU'da (Central Processing Unit) kullanılan mikroprogramlanabilir işlemci, yüksek seviyeli bir dil olan PROTEL kullanılarak programlanmaktadır.
- * Çevresel modüllerde kullanılan 8085 mikroişlemcisi, 8085 assembler seviyesi bir dille programlanmaktadır.
- * Çevresel modüllerde kullanılan 68000 mikroişlemcisi PASCAL ve 68000 assembler kullanılarak programlanmaktadır.

Birimlerin herbiri farklı bir programlama çevresine sahip olmakla beraber, basit bir arabaylaşım olan ve seri veri hatları üzerinden gönderilebilen mesajlar ile haberleşmektedir.

2.1 MERKEZİ KONTROL BİRİMİ (CCC)

Merkezi kontrol birimi bileşenleri, beraber çalışarak gelen kontrol işaretlerini yorumlarlar, uygun tepkilerde bulunarak alt birimlere ilgili komutları gönderirler.

CCC, DMS'deki bütün operasyonları yönetmek ve kontrol etmek üzere düzenlenmiş dört ayrı bileşenden oluşmaktadır :

1. Merkezi İşlem Birimi

2. Program Belleği

3. Veri Belleği

4. Merkezi Mesaj Denetleyicisi

2.1.1 Merkezi İşlem Birimi (CPU)

Merkezi İşlem Birimi (Central Processing Unit), DMS'in tüm işleyişini kontrol etmek için gerekli lojiğe sahiptir. CPU, işlevini program ve veri belleğine erişerek yerine getirir.

Merkezi Kontrol Birimini oluşturan bileşenlerin tümü güvenliği sağlamak amacı ile duplike edilmişlerdir. Normal koşullar altında, mevcut olan iki CPU, senkron olarak çalışmaktadır. Diğer bir deyişle, her iki CPU da aynı anda, aynı veri ile aynı komutu yürütmektedir. Böylece bir CPU'da meydana gelen bir hata kolayca sezilebilmektedir. Duplike edilmiş bir CCC yapısı, Şekil 3'de görülmektedir.

Merkezi İşlem Birimi, birbirinden bağımsız iki paralel yönlü bellek portuna sahiptir. Bu portlardan biri, çeşitli uzunlukta program komutları içeren dış belleğe yani program belleğine, diğeri ise veri içeren dış belleğe yani veri belleğine ve sistemin diğeri elemanları ile haberleşmeyi sağlayan merkezi mesaj denetleyicisine bağlıdır. Komut alma ve veri alma işlemleri paralel olarak yürütülebildiğinden program yürütme hızı arttırılabilmektedir. CPU, program belleğinden aldığı komutları, veri belleğinden aldığı veriler üzerinde yürüterek verilen emirleri yerine getirir ve daha sonraki adımlar için gerekli olan kararları verir.

2.1.2 Merkezi Mesaj Denetleyicisi (CMC)

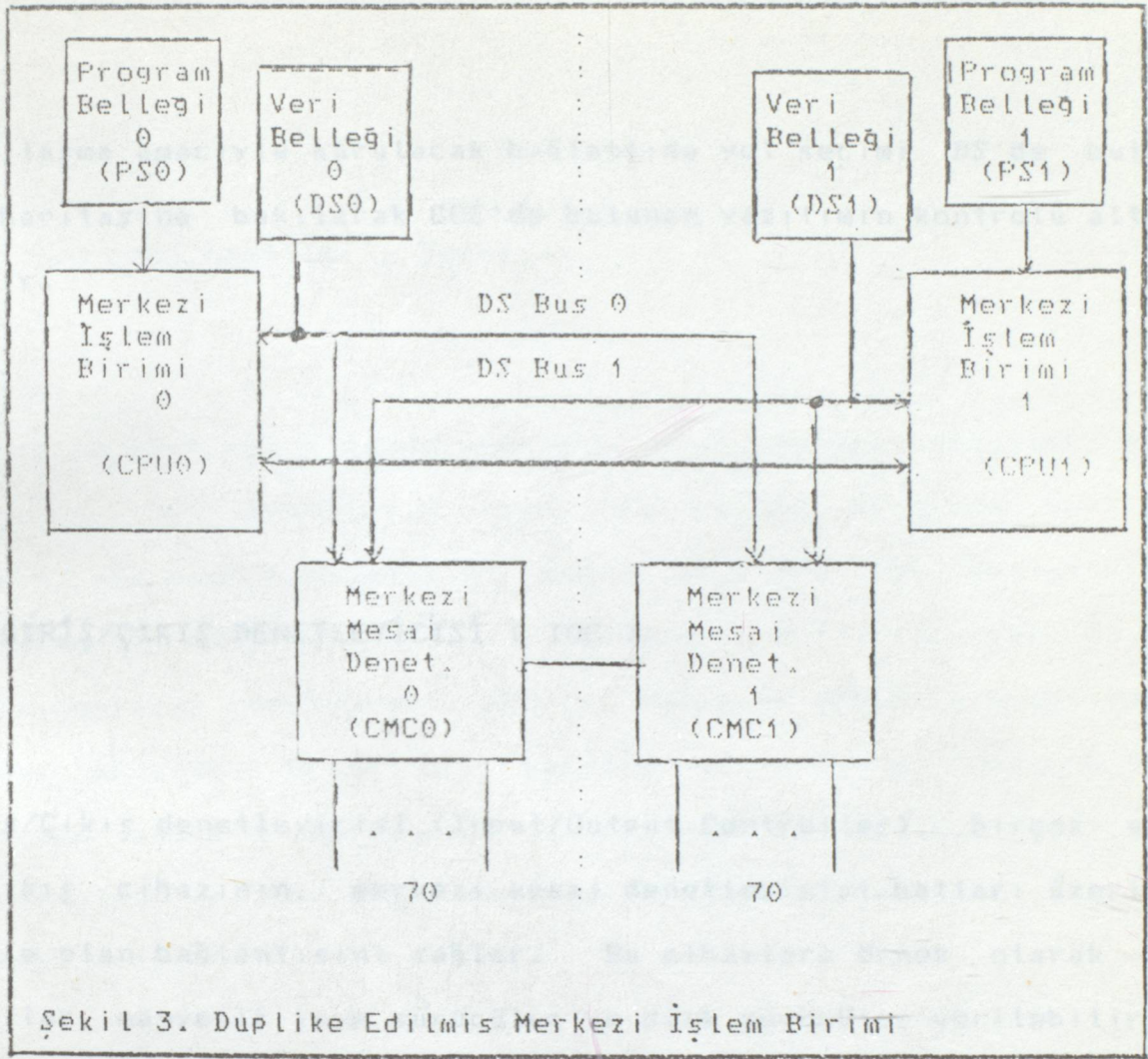
Merkezi mesaj denetleyicisi (Central Message Controller), ağıın bakım ve yönetim alanları ile CPU'nun dışarıyla olan bütün bağlantılarını gerçekleştirmekle yükümlü bir CCC birimidir.

2.1.3 Program Belleği (PS)

Program Belleği (Program Store), çağrı işlemi, bakım ve yönetim işleri için CPU tarafından gereksinim duyulan programları içerir. PS'in kelime uzunluğu 17 bittir. Bu 17 bitin 16 biti program komutu, 1 biti de eşlik içindir.

2.1.4 Veri Belleği (DS)

Veri Belleği (Data Store), her konuşmaya ait geçici bilgileri, müşteri verilerini ve santral parametrelerini içerir. DS'in kelime uzunluğu da 17 bittir. Bu 17 bitin 16 biti veri, 1 biti de eşlik içindir.



2.2 AĞ MODÜLLERİ (NM'S)

Bir girişin, bir çıkışa bağlantısı ağ modüllerinde gerçekleşir. Her ağ modülü, bu modülün merkezi kontrol birimi ve çevresel modüllerle ile mesajlaşmasını sağlayan bir ağ denetleyicisi içerir. Bu mesajlaşmada, ağ mesaj denetleyicisinin (NMC : Network Message Controller) PM'ler ve CCC ile bağlantısı merkezi mesaj denetleyicisi üzerinden gerçekleşir.

Mesajlaşma amacıyla kurulacak bağlantıda yol seçimi DS'de bulunan yol haritasına bakılarak CCC'de bulunan yazılımın kontrolü altında yapılır.

2.3 GİRİŞ/ÇIKIŞ DENETLEYİCİSİ (IOC)

Giriş/Çıkış denetleyicisi (Input/Output Controller), birçok giriş ve çıkış cihazının, merkezi mesaj denetleyicisi hatları üzerinden CPU ile olan bağlantısını sağlar. Bu cihazlara örnek olarak VDU, yazıcılar, manyetik teyp sürücüler ve disk sürücüler verilebilir.

2.4 ÇEVRESEL MODÜLLER (PM'S)

2.4.1 Çevresel Modüllerin Tanıtımı

DMS-100'un dört ana bileşeninden biri olan çevresel modüller, abone hatlarının ve trunkların, yani santraller arasında yer alan hatların, ağa olan bağlantısını sağlarlar. XPM ise Geliştirilmiş Çokluişlemcili Sistem (XPM : Extended Multiprocessor System) temelli çevresel modüllerdir. XPM'ler Motorola 68000 mikroişlemcisi kullanmaktadırlar. Ortak bir yazılım ve donanım temeline sahip XPM'lerden gelecek bölümlerde değinilecek olan sayısal trunk denetleyicisi ve hat grubu denetleyicisinin görevlerini şu şekilde özetleyebiliriz :

Hat Grubu Denetleyicisi (ILGC : International Line Group Controller): Çağrı koordinasyonu ve abonelere gereken farklı tonların verilmesi ile görevlendirilmişlerdir.

2.4.2 XPM'de Yazılım

2.4.2.1 XPM Yazılımının Genel Yapısı

XPM'in yazılım yapısı üç bölüm halinde organize edilmiştir :

1. RTSS

2. Sistem PASCAL'ı

3. Uygulama Yazılımı

RTSS (Run Time Support System): 68000 assembler kullanılarak gerçekleştirilmiştir. RTSS, işletim sisteminin yarısını ve p-code yorumlayıcıyı içerir. XPM PASCAL derleyicisi p-code şeklinde bir amaç (object) kodu üretir ve yorumlayıcı bu kodu XPM'de yürütebilir.

Sistem PASCALı: Sistem PASCAL'ında yer alan yazılım XPM PASCAL kullanılarak gerçekleştirilmiştir. Sistem PASCAL'ı da işletim sisteminin bir bölümünü içerir. XPM PASCAL, modüleritesi ve birkaç farklı komutu haricinde TURBO PASCAL yapısında bir bilgisayar dilidir.

Uygulama Yazılımı: Uygulama yazılımı, XPM PASCAL ve 68000 assembler kullanılarak gerçekleştirilir. Ayrıca bu yazılım XPM'ler için uygulama yazılımı kadar işletim sisteminin bazı yüksek seviyeli bölümlerini de içerir. Tezi oluşturan yazılım, XPM açısından bir uygulama yazılımıdır.

2.4.2.2 XPM Yazılım Modülleri

XPM yazılımının en basit yapısal bloğu modüldür. Modül, derlenebilen yada yüklenebilen en küçük yazılım birimidir. Bir modül, bir yada daha fazla fonksiyonu gerçekleştirmek üzere kullanılan prosedür ve veriler grubu olarak da tanımlanabilir.

Bir modül için kullanılan diğer iki terim ise birim (unit) ve segmenttir. Birim ve segment şu durumlarda kullanılır :

- * "Modül" kaynak koduna verilen isimdir. Bilindiği gibi, kaynak kodu bir programın uygulanmadan önce, diğer bir deyişle programcı tarafından yazıldığı zaman sahip olduğu yapıdır.
- * PASCAL derleyicisi, bir kod birimini derlediği için "unit" terimi, modül derlenirken kullanılır.

- * XPM yükleyicisi, bir birimin derlenmesi ve bağlanması (linking) elde edilen yük file'ni (load file) yükler. Segment ise bu yük file'ine denilmektedir.

2.4.2.3 XPM Pascal Modülünün Formatı

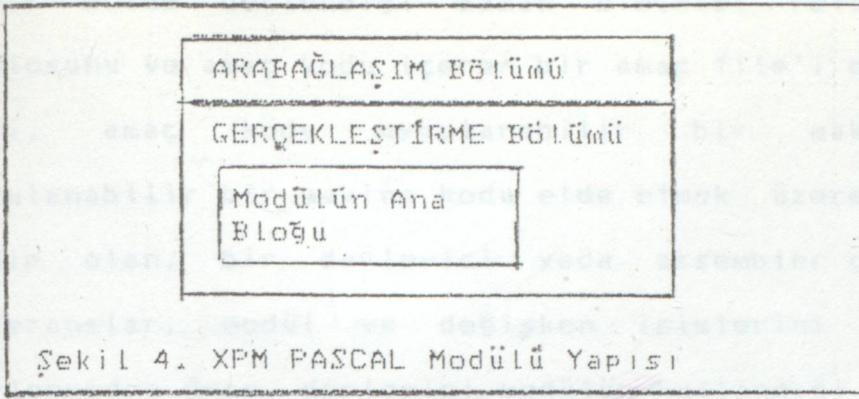
Bir XPM PASCAL modülü, Şekil 4'de görüldüğü gibi, iki bölüm içerir:

1. Arabağlaşım Bölümü (Interface Part)

2. Gerçekleştirme Bölümü (Implementation Part)

Bir birimin arabağlaşım bölümü, diğer birimler tarafından kullanılabilecek sabitleri, değişkenleri, tipleri, fonksiyonları, prosedürleri ve daha sonra açıklanacak olan taskları tanımlar.

Bir birimin gerçekleştirme bölümünde ise bu birime ait, özel sabitler, değişkenler, tipler, fonksiyonlar, prosedürler ve tasklar tanımlanır.



2.4.2.4 USES listesi

XPM PASCAL yapısında, diğer bir modül tarafından kullanılan bir modül, bu modülün alt seviyesindedir. Aynı modülleri kullanan fakat birbirini kullanmayan modüller ise aynı seviyededirler. Dolayısıyla bu şekilde oluşturulan yazılım hiyerarşik bir yapıya sahiptir.

Bir modülün derlenebilmesi için, bu modülün kullandığı diğer tüm modüllerin arabağlaşım bölümlerinin derlenmesi gerekir. USES listesi, bir modülün kullandığı modüllerin tümünü belirten bir listedir. USES listesinde, sadece arabağlaşım bölümleri bulunabilir. Diğer bir deyişle, sadece arabağlaşım bölümlerinde tanımlanan prosedürlere, değişkenlere, tiplere, fonksiyonlara, sabitlere ve task tiplerine bu modül dışından erişilebilir.

Bir bölüm derlendiği zaman simgesel referansları, bir simge tablosunu ve amaç kodu içeren bir amaç file'ı oluşturulur. Bilindiği gibi, amaç kodu uygulanabilir bir makina kodu olan yada uygulanabilir bir makina kodu elde etmek üzere işlenebilir yapıya sahip olan, bir derleyici yada assembler çıktısıdır. Simgesel referanslar, modül ve değişken isimlerini içerir. Bir bölüm derlenmeden önce, derleyici modülün kullandığı diğer tüm arabağlaşım bölümlerinin amaç file'larını ve simgesel bilgiyi bu bölümün isim ve tip tablosuna yükler. Bu işleme "imbedding" denir. Imbedding işlemi, tüm modüller arası referansların derleme sırasında tip kontrolünün yapılmasına olanak tanır. Örneğin, B modülünü kullanan bir A modülü olsun. A modülü derlendiği zaman, B modülünün arabağlaşım bölümü, B modülünün amaç file'ından alınabilir ve A modülü için olan kaynağa dahil edilir.

Ancak XPM PASCAL'da, geçişli imbedding işlemi yapılamamaktadır. Örneğin A, B ve C adında üç ayrı modül bulunsun. A modülü, USES listesinde B modülünü tanımlayarak B modülünü kullansın. B modülü de aynı şekilde C modülünü kullanıyorsa, A modülünün C modülünü dolaylı olarak kullanabileceğini söyleyemeyiz. Bu durumda A modülünün, C modülünü kullanabilmesi için kendi USES listesinde C modülünü tanımlaması gerekmektedir.

2.4.2.5 XPM Yazılımında Task Yapısı

XPM yazılımında, işlemlerin belirli bir sıraya dizilmesi, bu işlemlerin belirli zaman aralıkları ile tekrarlanması amacıyla yaratılan tasklara önem derecelerine göre öncelik verilmiş ve işlevlerine uygun yapı kazandırılmıştır.

XPM yazılımında kullanılan iki ana task yapısı bulunmaktadır. Bunlar 'wake-up' ve 'raise-event'dir. 'Wake-up' tasklar, XPM'in bir ana modülünde tanımlanan 'task handler' prosedürü ile sürekli olarak belirli aralıklarla uyandırılırlar. 'Raise-event' taskların uyandırılması içinse belirli bir olayın gerçekleşmesi gerekmektedir. Tasarımcı tarafından belirlenen bu olay, bir donanım kesmesi yada bir mesajın gelmesi olabilir.

2.4.2.6 XPM Yükünün Derlenmesi, Bağlanması ve Yüklmesi

Bir XPM yükünün derlenmesi ve üretilmesi şu adımlarda gerçekleştirilir:

1. İlk olarak, XPM PASCAL dili kullanılarak hazırlanan bir XPM modülünün, bir ana bilgisayardan bir XMS workstation'a yüklenmesi gerekir. XMS workstation'a yüklenen XPM modülü, burada XPM PASCAL derleyici tarafından derlenir. Bir birim derlendiği zaman, bu birim

inçin aşığıda belirtilen elemanları içeren bir amaç file'i oluşturulur :

II. Arabağların büküm için kaynak kodu

III. Arabağlar (Linker) bilgisi

2. Bir yekşerî tüm birimler derlendikten sonra, arabaglar her bir işlemci için (SP ve NP) bir yekşerî amaç file'leri oluşturulur. NP (Master Processor) ve SP (Signalling Processor) XPM'in kontrol bölgesinde yer alan iki ayrı işlemcidir. SP, XPM'in gerçek zamanlı fonksiyonları bir bilgisayar yardımıyla gerçekleştirir. NP'lere ve NP'lardan geçen verileri, arabaglar kontrolü bu fonksiyonlara örnek olarak verilebilir. NP ise, XPM'in arabaglarına ilişkin gerçek işlemleri fonksiyonlarını gerçekleştirir. Örnek olarak CC ve XPM arabaglarının yorumlanması, kanalları tahsis verilebilir.

NP ve SP, DMA (Direct Memory Access) yoluyla veri iletilmektedir. DMA arabagları ile NP'lar birleştirilerek, arabaglar ve veriler

yapabilmektedir. Buna karşın, MP'nin SP'ye erişimine izin verilmemektedir.

Derleyici gibi, bağlayıcı da XMS workstation'da yer almıştır. Yük için seçilecek olan modüller, bağlayıcı kontrol file'ı tarafından belirtilir. Bu yük file'ı, her bir modül için bir yük segmenti içerir. Bağlayıcı kontrol file'ında, isim verilen ilk PASCAL modül yükün ana modülü olur. Bağlayıcı tarafından üretilen işlemci yükü, bir XMS workstation'da yürütülerek test edilir.

3. Bu durumda, bağlayıcı tarafından üretilen yük file'ı, XMS Sistem Kodlayıcısı (XMS System Coder / XSYCO) Programına bir giriş olarak kullanılır. XSYCO programı, p-code'u doğal 68000 makina komutlarına dönüştürür. Bu işlem, sadece seçilen modüller için yapılır. Bağlayıcı kontrol file'ı, hangi modüllerin dönüştürülmesi gerektiğini belirtir. Dönüştürülecek olan modüllerin seçimi, bellek ve hız arasındaki ilişkiye bağlıdır. Çağrı işleme (call processing) için kullanılan modüller, doğal koda dönüştürülürken, hata bulan ve bu hataları gideren programlar (debug tools) p-code olarak kalırlar. Bu aşamada, yük, halen bir XMS workstation'da yürütülebilir bir yapıdadır.

4. XSYCO programından elde edilen çıkış, SLOG programına geçirilir. SLOG programı, (S-Record Load Generator Program) ayrı işlemci yük file'larını, tek bir yük file'ına yükler. SLOG kontrol file'ı şunları belirtmek için kullanılır :

* Belleğin düzenlenişi ;

Örneğin, MMU (Memory Management Unit) alanının büyüklüğü, XPM'deki belleğin büyüklüğü, vb.

* Hangi modüllerin yük dışında bırakılması gerektiği ;

Bu ana kadar, yükte yer alan bazı modüller, yükün bir XMS work-station üzerinde yürütülmesini sağlamak amacı ile kullanılmışlardır. Belirtilen modüller, bu aşamada yük dışına alınır.

* Hangi modüllerin, yükten kaldırılan bu modüllerin yerini alacağı ;

Modül büyüklüğü ve SP belleğinin sınırlı büyüklüğü nedeni ile, SP yüküne ait bazı modüller, yükten kaldırılan bu modüllerin yerini alırlar. (Bunun yerine MP belleğine yüklenirler.) Bu şekilde yükten kaldırılan bir modülün yerini alan bir SP yüküne, SP diagnostik örnek olarak verilebilir. Doğrudan bellek erişimi, SP diagnostik SP bellek yerine MP bellekten alınarak yürütülmesine olanak tanır.

* Hangi modüllerin paylaşılması gerektiğine ;

SP ve MP yüklerinde yer alan bazı modüller birbirine özdeş ve hatta bazen aynı modüllerdir. Bu modüllere örnek olarak hata giderici programlar içeren modüller verilebilir. Bu durumda, bellekten tasarruf etmek amacıyla, SP ve MP'ye yüklenecek olan modül, sadece MP'ye yüklenir. SP ve MP, MP'de yüklü olan aynı kodu yürütür.

MP'de ve SP'de özdeş olan birçok modül bulunabilir. Ancak, verimlilik nedeni ile bu modüllerin her iki bellekte de yer alması gerekebilir. Bununla beraber, böyle modüller için, yükte, yük segmentinin sadece bir kopyası bulunur. Örneğin, SP ve MP için RTSS kod özdeştir. Bu nedenle, CC'den XPM'e yükün sadece bir kopyası gönderilir. Fakat, yükleyici, yük segmentini her iki işlemciye de (SP ve MP) yükler. Çünkü SP'nin, tüm RTSS kodu MP bellekten alarak yüklenmesi, SP'nin MP'yi yavaşlatmasına neden olmaktadır.

5. Bu aşamadan itibaren, yük file'ına, her iki işlemcinin kullanımı için uygun bir yapı kazandırılmıştır ve artık XMS workstation'da koşturulamaz.

6. Yük file'i ana bilgisayara gönderilir, bir yük teybi yaratılır. Daha sonra bu yük teybi kullanılarak yük file'ı CC'ye yüklenir ve manyetik teyp yada diske kopyalanır.

XPM'in Yüklenmesi:

XPM'in yüklenmesi işlemi su aşamalarda yapılır :

1. Yeni bir yükün yerleştirilebilmesi için, XPM biriminin mutlaka reset edilmesi gerekir. XPM biriminin reset edilmesi aşağıda belirtilen şekilde gerçekleşir :

- * Yeni bir işlemci kartı (MP yada SP) yerleştirildiğinde donanım bir reset üretir.
- * CC, SP'ye XPM biriminin kendisini reset etmesi için gereken bir mesaj gönderebilir.
- * Aktivite zamanlayıcılar bir reset üretebilir.

2. Bir resetin sonucu olarak, XPM WAM (Who Am I) durumuna geçer.

3. CC, XPM'e düğüm numarası ve düğüm tipini gönderir.

4. Basit bir ilk yükleyici (bootstrap loader) ve CC ile mesaj değişim yapabilecek basit bir mesaj sistemi, hem SP ROM'a hemde MP ROM'a yerleştirilir.

5. CC, geri kalan yükleyici mesajlarını XPM'e gönderir.

6. RAM dinamik yükleyicileri (hem SP hemde MP'de bulunan) modüllerin (segmentlerin) bellekte nereye yüklenmesi gerektiğine karar verirler. RAM yükleyici, yükleme işlemini belleğin yüksek adresli bölümünden düşük adresli bölümüne doğru gerçekleştirilir.

7. Tüm modüller (segmentler) yüklendikten sonra, RAM mesaj sistemi gerekli mesajları göndermek üzere harekete geçer. SP ve MP başlangıç durumuna getirme işlemi için artık hazırdırlar.

Başlangıç Durumuna Getirme (Initialization):

XPM yüklenir yüklenmez, CC, XPM'e bir koş komutu gönderir ve her iki işlemci de, aynı anda, başlangıç durumuna getirme işlemine başlar. RTSS link edilen ilk modüldeki ilk prosedürü çağırır. Bu prosedür, sıra ile, yüklenen her bir modülün başlangıç durumuna getirme prosedürünü çağırır. Başlangıç durumuna getirme prosedürü, bir modülün ilk prosedürüdür. RTSS, her modüle ilişkin bilgileri (modüllerin, ayrı ayrı, prosedür directory'lerini) içeren bir tabloda gerekli değişikliklerin yapılması görevini de üstlenmiştir.

Bir modülün başlangıç durumuna getirme prosedürü, tasklar yaratabilir ve veriyi başlangıç durumuna getirebilir. İki istemci için, başlangıç durumuna getirme işlemi, ayrı olarak gerçekleştirilir. Buna karşın, işlemciler arasında eş zamanlılık sağlanması gereken iki nokta vardır :

1. MP'de hem MP hemde SP tarafından kullanılan İşlemler Arası Haberleşme (IPC : Inter-Process Communication) kuyrukları yer almaktadır. Dolayısıyla, SP, MP ile haberleşebilmek için, bu kuyrukların bulunduğu yeri bilmek zorundadır.

2. Ayrıca, MP'de modül haberleşme bölgesi vardır. Bu bölge, iki işlemci için de gerekli olan bilgiyi içerir. Bu bilgiye örnek olarak, IPC işaretçileri (pointer), bakım bilgisi verilebilir. SP bu bölgeye DMA ile erişebilmektedir.

Bir işlemci için başlangıç durumuna getirme işlemi, bu noktalardan birine eriştiği zaman, diğer işlemci de bu noktaya erişinceye kadar bekler. MP ve SP, başlangıç durumuna getirme işlemi sırasında kullanılan belirli bir bellek gözüne erişerek veri değişimi yapabilir.

2.4.3 XPM'de Güvenilirlik

XPM, güvenilirlik amacıyla 0 ve 1 olmak üzere iki ayrı birim halinde gerçekleştirilmiştir. Bu birimlerden biri aktif birim, diğeri ise yedek (standby) birimdir. Aktif birim, çağrı işleme ile yükümlüdür. Aktif birimde bir hata meydana geldiği takdirde, yedek

birim aktif birimin görevini üstlenir. Bu olaya 'swact' (switch activity) denilmektedir.

Bir swact meydana geldiği zaman, sırayla şu işlemler yapılır :

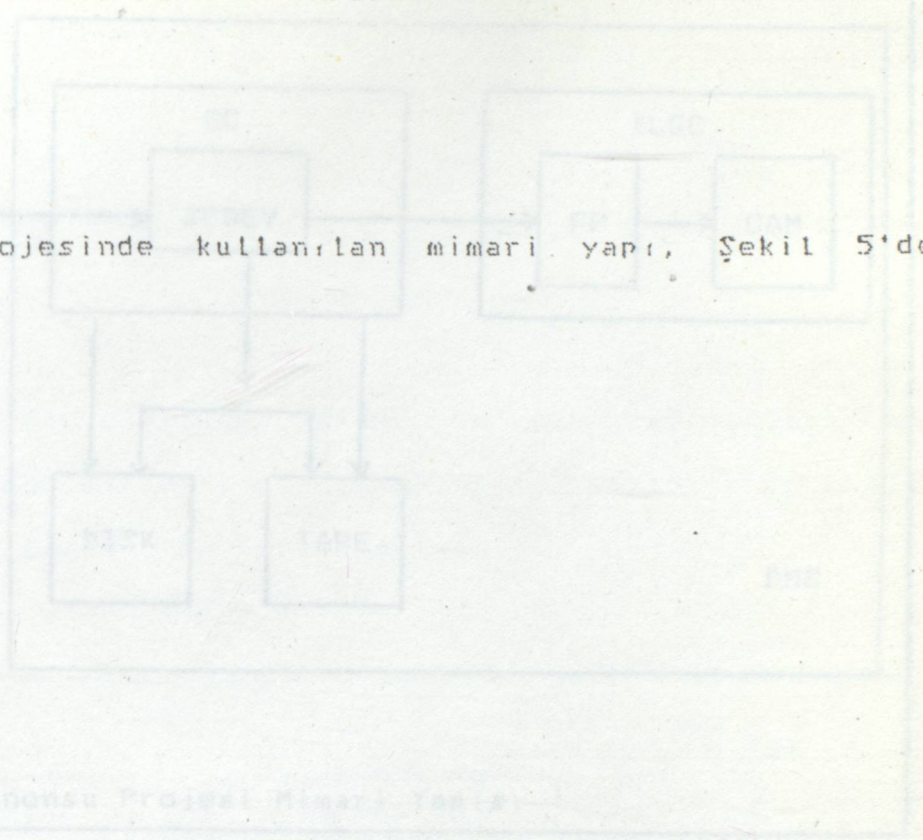
1. Aktif birim, yedek birime swact yapılacağını bildiren bir kesme gönderir.
2. Aktif birim, yaptığı işlemi durdurur.
3. Yedek birim kontrolü alarak aktif birim olur.
4. Her iki birim de swact'i CCC'ye bildirir. Başlangıçtaki aktif birim, aktiviteyi devrettiğini, yedek birim de aktiviteyi üstlendiğini bildirmek zorundadır.
5. Yeni aktif birim, eski aktif birimden hata bilgisini toplayarak, bunları CCC'ye bildirir.
6. Sadece aktivitenin devredilmesi sırasında ulaşan çağrılar kaybedilmez. Aktif birimde hata sezilmesi ve aktivitenin devri arasında gelen çağrılar ise kaybedilir.

BÖLÜM 3

REKLAM ANONSU PROJESİ DONANIM YAPISI

3.0 REKLAM ANONSU PROJESİ DONANIM YAPISI

Reklam anonsu projesinde kullanılan mimari yapı, Şekil 5'de görülmektedir.

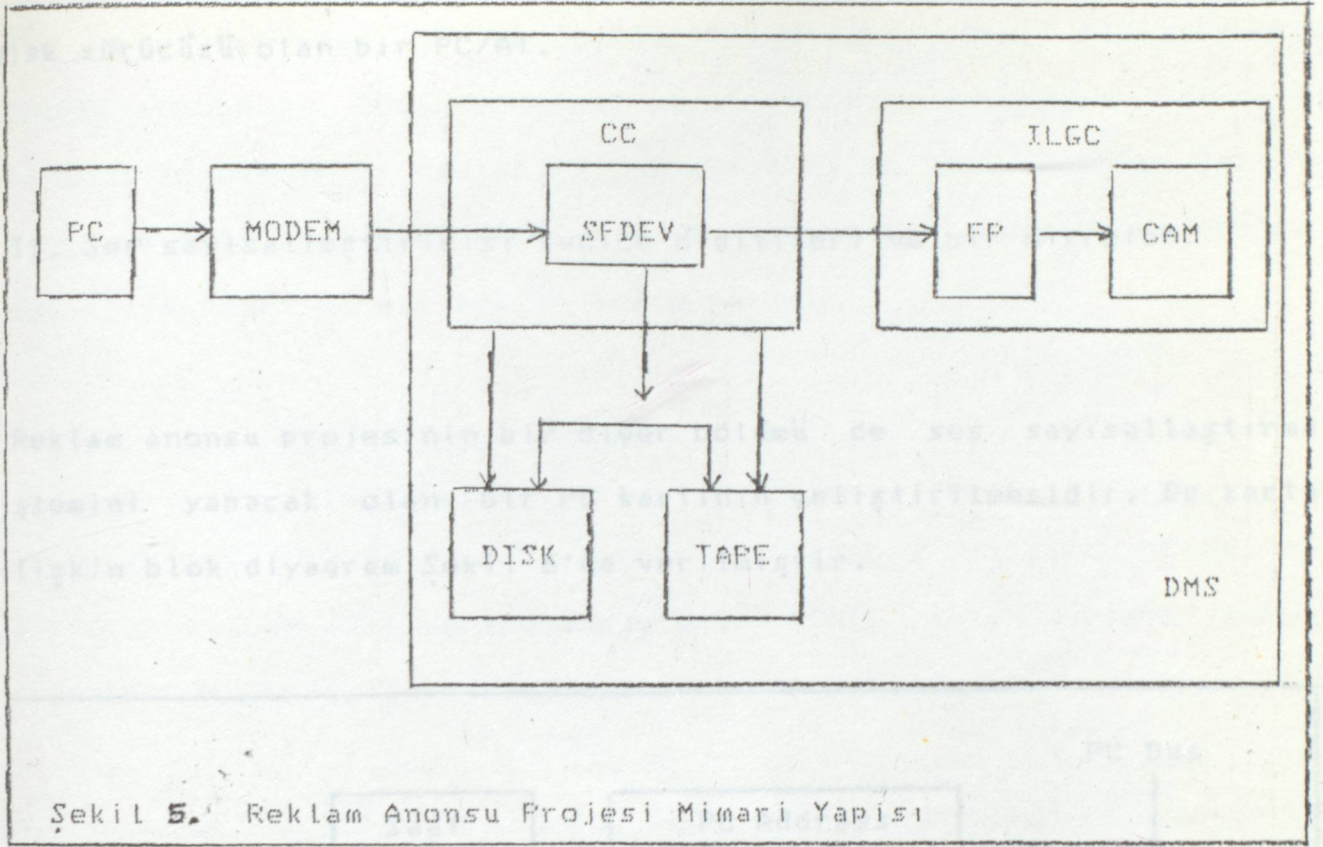


Şekil 5: Reklam Anonsu Projesi Mimari Yapısı

Bu yapı, aşağıdaki işlevleri gerçekleştirir:

1. Reklam anonsunun ses verisi dijital bir bilgisayarda yani bir PC'de, sayısal (dijital) veriye dönüştürülmektedir. Sürekli (analog) verinin sayısalıya dönüştürülmesi sırasında, yaklaşık 8000 örnek alınmakta ve her bir örnek, 16 bitli sistemde, 16 bit başlangıç ile kodlanmaktadır.

Ses verisi geliştirme sistemi, ses verilen ve ses, sayısal veriye dönüştürülen bu sistem, iki ana bölüme ayrılır:



Bu yapı, sırasıyla şu fonksiyonları gerçekleştirir :

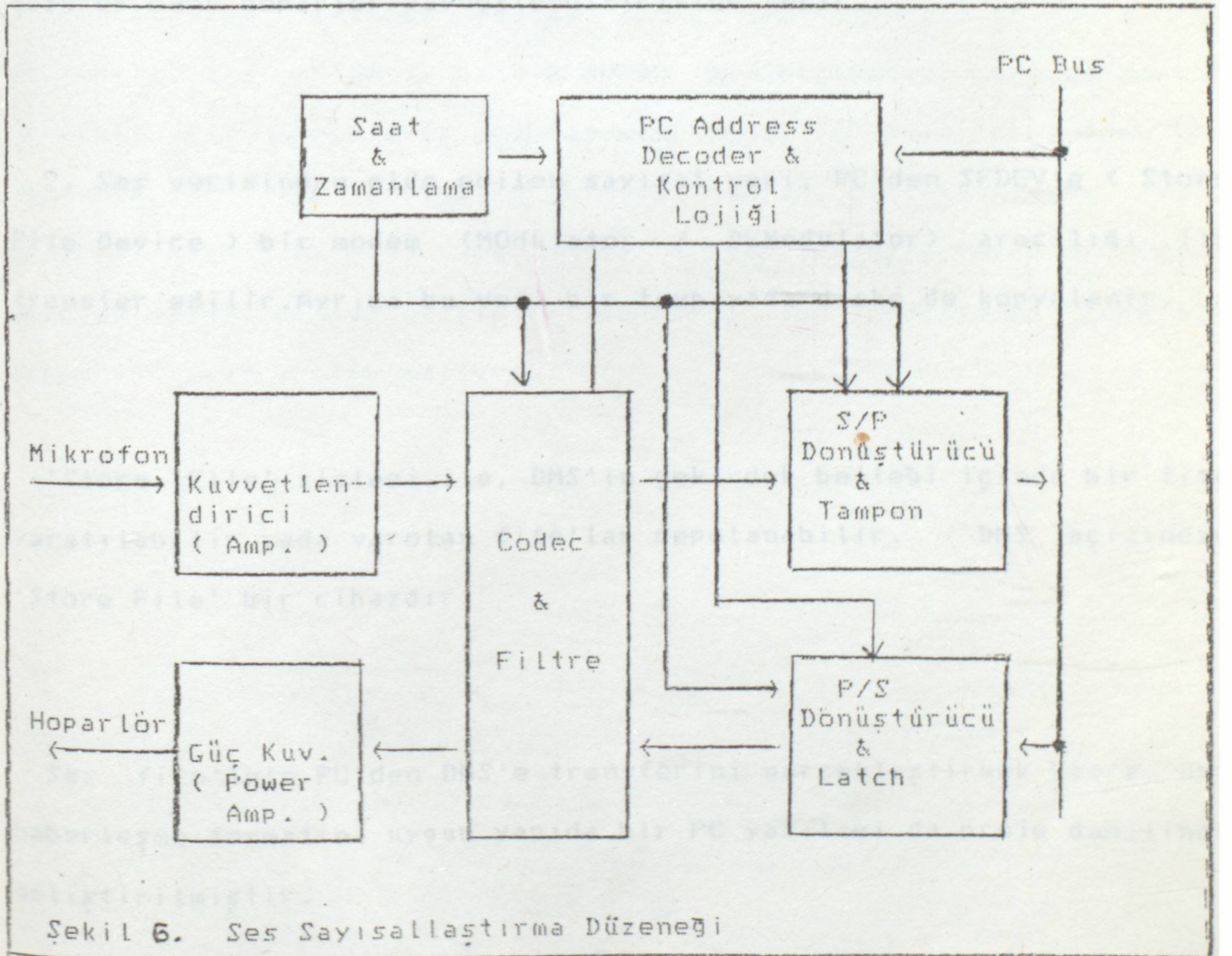
1. Reklam anonsunun ses verisi kişisel bir bilgisayarda yani bir PC'de, sayısal (digital) veriye dönüştürülmektedir. Sürekli (analog) verinin sayısala dönüştürülmesi sırasında, saniyede 8000 örnek alınmakta ve her bir örnek, ikili sistemde, sekiz basamak ile kodlanmaktadır.

Ses verisi geliştirme sistemi adı verilen ve sesi sayısal veriye dönüştüren bu sistem, iki ana bileşenden oluşur :

I. IBM uyumlu, 10 MByte bellek kapasiteli hard disk ve iki floppy disk sürücüsü olan bir PC/AT.

II. Ses sayısallaştırıcısı (voice digitizer) ve bir mikrofon.

Reklam anonsu projesinin bir diğer bölümü de ses sayısallaştırma işlemini yapacak olan bir PC kartının geliştirilmesidir. Bu karta ilişkin blok diyagram Şekil 6'da verilmiştir.



Şekil 6. Ses Sayısallaştırma Düzenegi

Ses sayısallaştırıcı kartı, IBM uyumlu kişisel bir bilgisayarın PC bus slot'larından birine yerleştirilmiştir. Ses, Kodlayıcı/Kod Çözücü (CODEC : COder/DECoder) ve filtrede, örtüşmeye (aliasing) karşı filtrelenir ve kodlanır. Seri/Paralel dönüştürücüsü ve tampon, seri veriyi paralele dönüştürerek, adres kod çözücüsü ve kontrol lojisinin denetimi altında PC bus'a verir.

Sayısallaştırılmış olan reklam anonsunun, kontrol amacı ile dinlenebilmesi için, paralel/seri dönüştürücü ve latch, PC'den ses verisini alarak seri veriye dönüştürür. CODEC, gelen verinin kodunu çözerek bunu hoparlör kuvvetlendiricisine verir.

2. Ses verisinden elde edilen sayısal veri, PC'den SFDEV'e (Store File Device) bir modem (MODulator / DEModulator) aracılığı ile transfer edilir. Ayrıca bu veri bir teyp yada diske de kopyalanır.

'Store File' sistemi ile, DMS'in çekirdek belleği içinde bir file yaratılabilir yada varolan file'lar depolanabilir. DMS açısından 'Store File' bir cihazdır.

Ses file'inin PC'den DMS'e transferini gerçekleştirmek üzere, DMS haberleşme formatına uygun yapıda bir PC yazılımı da proje dahilinde geliştirilmiştir.

3. Reklam anonsu verisi, bir SFDEV, bir teyp yada diskten okunarak, ILGC'de yer alan FP'ye (Facilities Processor) yüklenir.

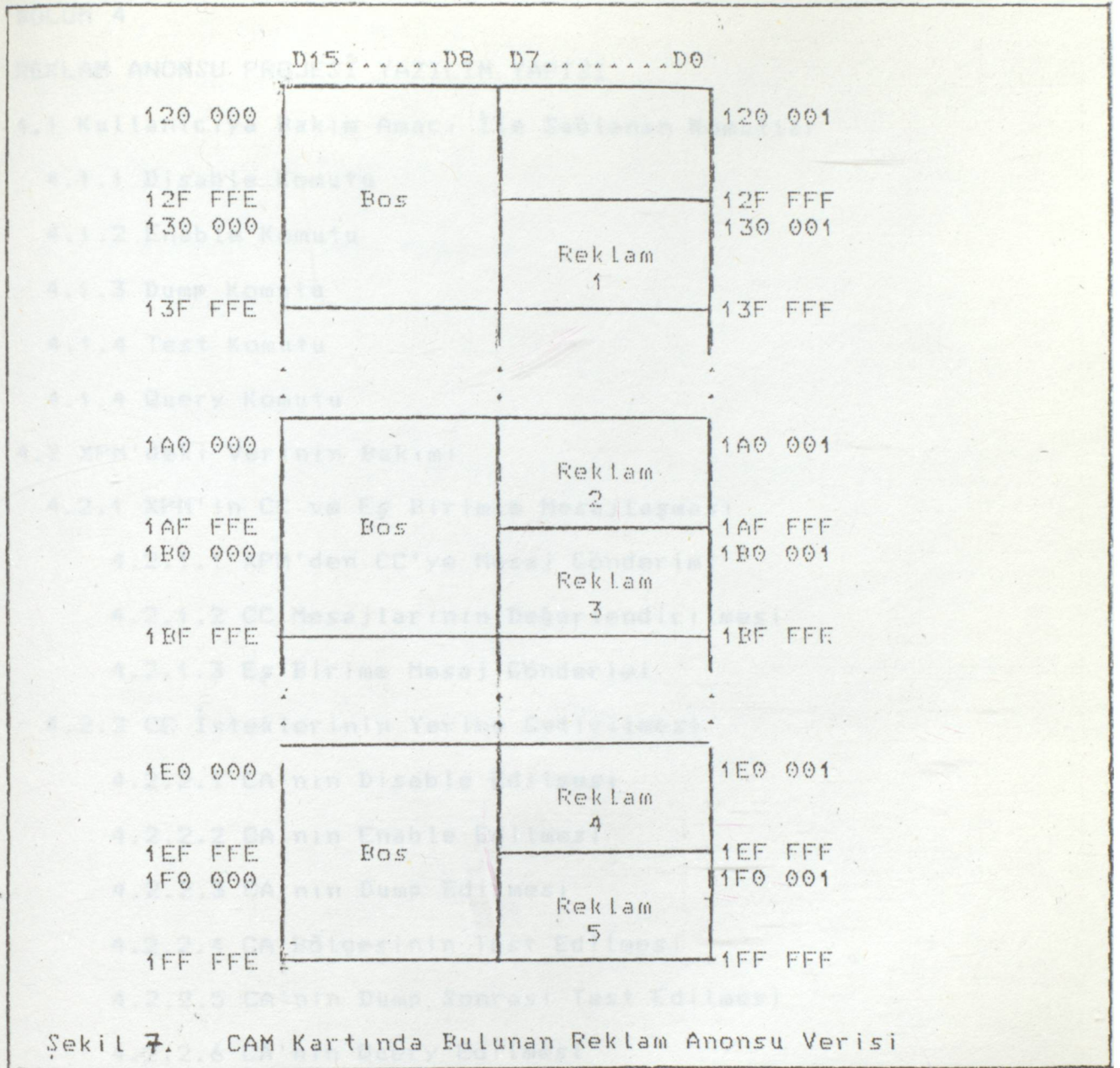
4. Reklam anonsu verisi, CAM kartına yüklenir.

5. CAM kartındaki reklam anonslu ring back tone'a yerleştirilir ve aboneye verilir.

CAM kartına yüklenen reklam anonsu verisinin yapısı Şekil 7'da gösterilmiştir. FP'de 16 bitlik 68000 mikroişlemcisi kullanılmakla birlikte dış birimlerle bağlaşımın kolay olması açısından CAM kartında yer alan bellek 8 bitlik olarak düzenlenmiştir. Veri yolunun diğer 8 biti ise lojik 1 yapılmıştır. Böylece meydana gelecek hatalı bellek erişimlerinde NOP'a (No Operation) karşılık düşen #FF değeri elde edilecektir.

Şekil 7. CAM kartında bulunan Reklam Anonsu Verisi

1. reklam için veri #130000 ve #13FFFF, 2. reklam için veri #140000 ve #14FFFF, 3. reklam için veri #150000 ve #15FFFF, 4. reklam için veri #160000 ve #16FFFF ve 5. reklam için veri #170000 ve #17FFFF adreslerine yüklenmiştir.



İlk reklama ilişkin veri #130000 ve #13FFFF, 2. reklama ilişkin veri #1A0000 ve #1AFFFF, 3. reklama ilişkin veri #1B0000 ve #1BFFFF, 4. reklama ilişkin veri #1E0000 ve #1EFFFF ve 5. reklama ilişkin veri ise #1F0000 ve #1FFFFF adreslerine yüklenmiştir.

BÖLÜM 4

REKLAM ANONSU PROJESİ YAZILIM YAPISI

4.1 Kullanıcıya Bakım Amacı İle Sağlanan Komutlar

4.1.1 Disable Komutu

4.1.2 Enable Komutu

4.1.3 Dump Komutu

4.1.4 Test Komutu

4.1.4 Query Komutu

4.2 XPM'deki Verinin Bakımı

4.2.1 XPM'in CC ve Eş Birimle Mesajlaşması

4.2.1.1 XPM'den CC'ye Mesaj Gönderimi

4.2.1.2 CC Mesajlarının Değerlendirilmesi

4.2.1.3 Eş Birime Mesaj Gönderimi

4.2.2 CC İsteklerinin Yerine Getirilmesi

4.2.2.1 CA'nın Disable Edilmesi

4.2.2.2 CA'nın Enable Edilmesi

4.2.2.3 CA'nın Dump Edilmesi

4.2.2.4 CA Bölgesinin Test Edilmesi

4.2.2.5 CA'nın Dump Sonrası Test Edilmesi

4.2.2.6 CA'nın Query Edilmesi

4.3 Diagnostik Testler

4.3.1 CAM Kartının Test Edilmesi

4.3.1.1 Yapılan Testin Amacı

4.3.1.2 Testin Yapılışı

4.3.2 CAM Belleğinin Test Edilmesi

4.3.2.1 Yapılan Testin Amacı

4.3.2.2 Testin Yapılışı

4.3.3 CA'nın Test Edilmesi

4.3.3.1 Yapılan Testin Amacı

4.3.3.2 Testin Yapılışı

4.4 Yazılım Modülleri

4.4.1 Veri Bakım Modülleri

4.4.2 Diagnostik Test Modülleri

4.0 REKLAM ANONSU PROJESİ YAZILIM YAPISI

Bir reklam, ayrıca sadece belirli bir süre için 'disable' duruma getirilebilir. Diğer bir deyişle, 'enable' durumunda olan bir reklam, kullanıcı tarafından verilen 'disable' komutu ile 'disable' durumuna getirildikten sonra, tekrar 'enable' durumuna getirilebilir. Bunun için de kullanıcıya 'enable' komutu verilmelidir. 'Enable' komutu ile istenilen birimdeki istenilen

4.1 KULLANICIYA BAKIM AMACI İLE SAĞLANAN KOMUTLAR

4.1.1 Disable Komutu

Arzat reklam anonsu verisinden, 'disable' durumunda olduğu görülebilir. CAM kartında bulunan reklam anonsları, iki durumda olabilirler. Bu durumlardan biri 'enable', diğeri ise 'disable'dir. Bir reklamın abonelere verilebilmesi için, reklam anonsu durumunun 'enable' olması gerekmektedir. Verisinde bir bozulma olduğu sezilen yada PTT ile olan sözleşme süresi biten bir reklamın abonelere verilmesi, kullanıcının gireceği ve parametreleriyle hangi birimdeki hangi reklam anonsunu 'disable' durumuna getirmek istediğini belirteceği bir 'disable' komutu ile 'disable' durumuna geçirilerek önlenebilir.

4.1.2 Enable Komutu

Bir reklam, ayrıca sadece belirli bir süre için 'disable' duruma getirilebilir. Diğer bir deyişle, 'enable' durumunda olan bir reklam, kullanıcı tarafından girilen 'disable' komutu ile 'disable' durumuna getirildikten sonra, tekrar 'enable' durumuna getirilebilir. Bunun için de kullanıcıya 'enable' komutu sağlanmıştır. 'Enable' komutu ile, istenilen birimdeki istenilen reklam anonsu 'disable' durumdan 'enable' duruma getirilebilir.

Ancak reklam anonsu verisinin, 'disable' durumunda olduğu süre boyunca bir bozulmaya uğrayıp uğramadığının, bu reklam anonsu verisi 'enable' duruma getirilmeden önce test edilmesi gerekmektedir. İleri bir bölümde açıklanacak olan bu testin sonucu olumlu ise, yani bu reklam anonsu 'disable' durumda iken, bu reklam anonsunun verisinde herhangi bir bozulma meydana gelmemişse reklam anonsu yeniden 'enable' durumuna getirilir. Aksi takdirde, kullanıcının 'enable' durumuna getirme isteği geri çevrilerek, kullanıcıya bunun nedeni bildirilir.

4.1.3 Dump Komutu

Kullanıcıya sağlanan bir diğer komut da 'dump'dir. Kullanıcı 'dump' komutunu girerek, istediği reklam anonsuna ilişkin veriyi, CC üzerinden XPM'de bulunan CAM kartına yükleyebilir. CAM kartında sadece beş anonsun bulunmasına rağmen, disklerde birçok reklam anonsu saklanabilir ve bu reklam anonsları abonelere verileceği zaman hangi birime hangi reklam anonsunu yüklemek istediğini belirten parametrelere sahip bir 'dump' komutu girilerek CCC üzerinden CAM kartına alınabilir.

Dump işlemi tamamlanan bir reklama, ileri bölümlerde açıklanacak olan test uygulanarak 'dump' işleminin başarıyla sonuçlanıp sonuçlanmadığı, verinin doğru olarak CAM kartına yazılıp yazılmadığı kontrol edilir. Hatasız olarak CAM kartına yazılan bir reklam anonsu, belirtilen testten geçtikten sonra, otomatik olarak 'enable' durumuna getirilerek, sırası geldiğinde abonelere verilmeye başlanır. Ancak bir hata sezilirse, diğer bir deyişle test sonucu olumsuz ise, durum kullanıcıya bildirilir ve reklam anonsu 'disable' durumunda bırakılarak abonelere verilmez.

4.1.4 Test Komutu

Kullanıcı, bir reklam anonsu ses verisinde hata sezdiği, aynı reklam anonsu verisini birkaç defa CC üzerinden CAM kartına yüklenmesine rağmen, yükleme işlemi sonunda yapılan testten olumsuz sonuç alınırsa, CAM kartında o reklam anonsunun bulunduğu bellek bölgesinin donanımını test etmek zorundadır. Bunun için de kullanıcıya 'test' komutu sağlanmıştır. 'Test' komutu ile kullanıcı, bu bellek bölgesinin doğru olarak çalışıp çalışmadığını kontrol edebilmektedir. Ancak ileri bir bölümde ayrıntılı olarak açıklanacak olan bu bellek testinin sonucunda, bulunduğu bellek bölgesi üzerinde test yürütülen reklam anonsu verisi tamamen bozulacağı için bu reklam 'disable' durumuna getirilerek, kullanıcıya durumu açıklayan bir uyarı mesajı verilecektir.

4.1.5 Query Komutu

Kullanıcıya sağlanan son komut 'query'dir. 'Query' komutunun girilmesi ile, kullanıcıya, CAM kartındaki beş reklam anonsuna ilişkin durum bilgilerini saklayan ve CC'de saklanarak güncelleştirilen bir tablo verilir.

4.2 XPM'DEKİ VERİNİN BAKIMI

4.2.1 XPM'in CC Ve Eş Birimle Mesajlaşması

XPM, DMS-100/200'de yürütülen işlemler arasında haberleşmeyi sağlayabilmek amacıyla, çeşitli mesaj sistemlerinden yararlanır. XPM'in kullandığı bu mesaj sistemlerinden biri İşlemler Arası Haberleşme'dir (IPC:Inter-Process Communication). Bir taskın mesaj gönderebilme ve alabilmesi için, bir mesaj kuyruğu ve bir haberleşme tanıtıcısı (CID:Communication Identifier) ayrılması gerekir. Bir task, kendisine ilişkin bir yada daha fazla CID'ye sahip olabilir. Ancak bir CID, tek bir taskı tanıtabilir.

4.2.1.1 XPM'den CC'ye Mesaj Gönderimi

XPM'den CC'ye mesaj gönderiminde aşağıda gösterilen mesaj yapısı kullanılmıştır.

00	Standart Mesaj Başlığı	
07		
08	Mt_general_handshake	(Hex 02)
09	Düğüm tipi (node_type)	(Hex 0B)
0A	Solicitor numarası	
0B	Yanıt (reply)	REQ_DONE, REQ_FAILED
0C	İstek (request)	CAM_NOT_PRESENT
0D	Reklam Anonsu Numarası (0, (1 - 5))	DISABLE, ENABLE, DUMP_REQ
	Units_comm_bool	TEST, QUERY, ENABLE_INI
0E	Birim numarası (unit_num)	
0F	Mesaj Checksumı (message checksum)	

Şekil 8. CC'ye gönderilen onay mesajı

Kullanıcının, kendisine sağlanan komutları girerek XPM'de verinin bakımını yürütmesi, CC'nin arabağlasımı ile gerçekleşmektedir. Bu komutlar sonucu yapılması gereken işlemler CC'den XPM'e gönderilen mesajlarla bildirilmekte ve bu işlemlerin sonuçları XPM'den CC'ye onay mesajında gönderilmektedir.

CC'ye gönderilen onay mesajında, Şekil 8'de de görüldüğü gibi, 10 ayrı bölüm bulunmaktadır. Mesajın ilk bölümü XPM mesajlarının tümünde bulunan standart mesaj başlığıdır. Standart mesaj başlığı, düğüm numarası, mesaj uzunluğu gibi bilgileri içermektedir. Bir CC'ye bağlı olarak birden fazla ILGC bulunabilir. Bu nedenle,

XPM'den CC'ye gönderilen bir mesajın, hangi ILGC'den gönderildiğinin CC tarafından anlaşılabilmesi için, mesajın ILGC'nin düğüm numarasını içermesi gerekir. Bunun için mesajın 3. bölümü kullanılmaktadır.

XPM'den CC'ye gönderilen mesaj, CC'nin isteği üzerine gönderilmiſse, diğeri bir deyiſle, CC XPM'den bu mesajın gönderilmesini bekliyorsa bu mesaja 'solicitor' mesaj denir. Ancak XPM, CAM kartına erişim yapılamaması gibi bir durumla karşı karşıya kalmışsa, bunun CC'ye bildirilmesi için, CC'nin beklemediği bir mesaj gönderebilir. Bu tür mesajlara 'unsolicitor' mesaj denilmektedir. CC'ye gönderilen mesajın 4. bölümünde, bu mesajın 'solicitor' yada 'unsolicitor' olduğu bildirilmektedir. CC'den gelen mesajdan, CC'nin bu mesaja verdiği 'solicitor' numarası alınarak, bir değışkene atanır ve yanıt mesajında bu değışkenden alınan sayı, mesajın 4. bölümüne atanır.

CC isteğine verilen yanıt mesajın, 5. bölümünde yer almaktadır. Bu yanıt istek yerine getirildiğinde req_done, istek yapılamadığında req_failed, CAM kartı takılı olmadığı yada erişilemediğinde ise cam_not_present'dir. 6. bölüm, bu mesajın CC'nin hangi isteğine yanıt olduğunu belirtmek üzere gönderilmektedir. CC'nin bir reklam anonsunu enable etme isteği, enable; disable etme isteği, disable; dump etme isteği, dump_request; test etme isteği, test; query etme isteği, query ve dump edildikten sonra enable etme isteği de enable_info ile XPM'e bildirilir.

Mesajın 7. bölümü, CC'nin bu isteği hangi reklam için gönderdiğini belirtir. Bu bölümdeki değer 1 ile 5 arasında ise, istek bu numaraya sahip reklam anonsu için yerine getirilir. 0 ise tüm reklam anonsları için bu istek geçerlidir.

Mesajın ILGC'nin hangi biriminden gönderildiğini belirtmek üzere, 9. bölüm kullanılmaktadır. Bilindiği gibi, bir ILGC'de unit0 ve unit1 olmak üzere iki ayrı birim bulunmaktadır.

Son mesaj bölgesi ise mesaj checksumını içerir. Mesaj checksumının kontrolünü sistem otomatik olarak yapmaktadır. Checksum'ı uyuşmayan mesajların gönderimi yinelenmektedir.

CC'ye mesaj gönderiminde kullanılan bu mesajın yanı sıra daha sonra açıklanacak olan checksum testinin sonucunun CC'ye bildirilmesi durumunda, farklı yapıya sahip bir mesaj gönderilecektir. Bu mesaj, checksum testinin sonucunun yanı sıra, anons adını ve durumunu da içerecektir. Gönderilen anons adı ve durumu, CAM kartından okunacak ve ilk mesaj formatına eklenen bölümlerde gönderilecektir. Gönderilen bu değerler CC tarafından, CC'deki değerlerle karşılaştırılarak ilgili testler yapılacaktır. Bu mesajın yapısı Şekil 9'da gösterilmiştir.

Checksum testi sonucunun bildirildiği bu mesaj, query_msg_to_cc prosedürü ile gönderilecektir.

00	Standart Mesaj Başlığı	
07		
08	Mt_general_handshake	(Hex 02)
09	Dugum tipi (node_type)	(Hex 0B)
0A	Solicitor numarası	
0B	Yanıt (reply)	QUERY_REPLY
0C	Istek (request)	QUERY
0D	Reklam Anonsu Numarası	(0, (1 - 5))
0E		
	Reklam Anonsunun Adı	
15		
16	Reklam Anonsunun Durumu	
17	Units_comm_bool	
18	Birim numarası (unit_num)	
19	Mesaj Checksumı (message checksum)	

Şekil 9. -CC'ye gönderilen checksum testi sonucu

4.2.1.2 CC Mesajlarının Değerlendirilmesi

XPM'deki verinin bakımı amacıyla yaratılan `ann_task_type` adındaki task, bir 'raise_event' tasktır. Bu tür tasklar, XPM'e bir mesaj geldiği zaman uyanır, ilgili işlemleri yaptıktan sonra, bir sonraki mesaj gelene kadar tekrar uyur. O halde, CC'den mesaj gelmesi ile task uyanacak ve gelen mesajı değerlendirecektir.

Standard Mesaj Başlığı

NXCOMANN modülünde bulunan `comm_hdlr` prosedürü, CC'den gelen mesajda belirtilen isteğe göre işlemler yapar. Bu prosedürde, ilk olarak tamponda bir mesajın olup olmadığı kontrol edilir. Tamponda CC'ye gelen bir mesajın olması durumunda, işlemler ve CC'ye işlem sonucunun gönderilmesi sırasında gereken parametreler mesajdan alınarak ilgili değişkenlere atanır. Daha sonra bu işlemleri yapar yada bunların yapılması için bazı isteklerde bulunur. İstekte bulunduğu işlemler, `ca_task` tarafından çağrılarak 2 sn.lik ring süresinde tamamlanması gereken işlemlerdir.

Solicitor numarası

CC'den XPM'e gönderilen bakım mesajının formatı Şekil 10'da gösterilmiştir. CC'den gönderilen bakım mesajında, XPM'den CC'ye gönderilen onay mesajından farklı olmak üzere bazı bölümler bulunmaktadır. Farklı bölümlerden ilki, 'send'dir. Bu bölüm gönderilen mesajın XPM'e yönlendirildiğini göstermektedir.

Bir diğer farklı bölüm de 'fiat'tır. Bu bölüm, CC'den XPM'e gönderilen mesajın, bakım mesajı mı yoksa veri mesajı mı olduğunu XPM'in anlaması için kullanılmaktadır. Fiat bölümünün NETAS_MTC_FIAT içermesi, bu mesajın bir bakım mesajı olduğunu ve

CC'nin XPM'de yapılmasını istediği işlemi içerdiğini gösterir. Fiat bölümünde NETAS_TBL_FIAT olması durumunda ise gönderilen mesaj, reklam anonsu verisi içeriyor demektir.

00	Standart Mesaj Başlığı		
07			
08	Send	(Hex DE)	
09	Varis (Destination)	(Hex 24)	CID_COM_ANN
0A	Kaynak (Source_CC)	(Hex CC)	CID_CC
0B	Uzunluk (Length)	(Hex 06)	
0C	Fiat	(Hex 51)	NETAS_MTC_FIAT
0D	İstek (Request)		DISABLE, ENABLE, DUMP TEST, QUERY, ENABLE_INF
0E	Mt_general_handshake	(Hex 02)	
0F	Düğüm tipi (Node_type)	(Hex 0B)	
10	Solicitor numarası		
11	Reklam Anonsu Numarası	(0 , (1 - 5))	
12	Birim numarası (Unit number)		
13	Mesaj checksumı		

Şekil 10. CC'den XPM'e gönderilen bakım mesajı

CC'den XPM'e bakım mesajının yanı sıra veri mesajı da gönderilmektedir. Bu veri mesajı ile reklam anonsu verisi, CC

üzerinden XPM'e dump edilmektedir. Veri mesajının yapısı Şekil 11'de gösterilmiştir.

00	Standart Mesaj Başlığı		
0F			
11	Varış	(Hex 24)	CID_COM_ANN
12	Kaynak	(Hex CC)	CID_CC
13	Uzunluk		
14	Fiat	(Hex 50)	NETAS_TBL_FIAT
15	Solicitor Numarası		
16	Reklam Anonsu Numarası	(1 - 5)	
17	Sayfa	(0 -125)	
0			
.			
.	Reklam Anonsu Verisi		
.			
255			
	Mesaj Checksumı		

Şekil 11. XPM'e Gönderilen Reklam Anonsu Verisi

Veri mesajında, bakım mesajından farklı olmak üzere, sayfa ve reklam anonsu verisi alanları bulunmaktadır. 32K'lık reklam anonsu verisinin her 256 byte'ı tek bir mesaj içinde gönderilmektedir. Mesajın sayfa bölümü, verinin kaçınıcı 256 byte'lık bölümünün gönderildiğini belirtmektedir.

4.2.1.3 Eş Birime Mesaj Gönderimi

Aktif birimde, CC'den gelen isteklerle yapılan işlemlerin eş birimde de yapılabilmesi için aynı isteklerin eş birime de gönderilmesi gerekmektedir. Bu mesajın gönderimi, `send_req_to_mate` prosedürü ile gerçekleştirilir. CC'ye gönderilen onay mesajı yapısı kullanılmakta, sadece varış yeri olarak eş birim verilmektedir.

4.2.2 CC'den Gelen İsteklerin Yerine Getirilmesi

CC'nin isteklerini XPM'e bildirmesi, XPM'e gönderdiği mesajlarla gerçekleşmektedir. XPM tarafından alınan bu istek mesajları, NXCOMANN modülünde tanımlanan `comm_hdlr` prosedüründe incelenerek ilgili işlemler yapılır.

4.2.2.1 CA'nın Disable Edilmesi

CC'den gelen mesaj, bir reklam anonsunun disable duruma getirilmesi isteğini içeriyorsa, ilgili bayrağa `set_flags` prosedürü ile false

atanır ve CC'ye, gelen isteğin yapıldığını belirten bir mesaj gönderilir.

4.2.2.2 CA'nın Enable Edilmesi

Gelen CC mesajı, bir reklam anonsunun enable duruma getirilmesi için bu reklama ilişkin, ileri bölümlerde açıklanacak olan checksum testinin yapılması isteğini belirten bayrağa true atanır. Böylece 2 sn.lik ring süresinde checksum testinin yapılması sağlanır. Checksum testinin sonucu olumlu ise reklam enable duruma getirilir. Test sonucunun olumsuz olması durumunda reklam anonsu disable durumda bırakılarak, durum kullanıcıya bildirilir.

4.2.2.3 CA'nın Dump Sonrası Test Edilmesi

4.2.2.3 CA'nın Dump Edilmesi

CC'den bir reklam anonsunun dump edilmesi isteği gelmesi durumunda, bu dump isteğini belirten bayrağa, 2 sn.lik ring süresi içinde dump işlemini gerçekleyen prosedürün çağırılması için true atanır ve bu reklam anonsu set_flags prosedürü ile disable duruma getirilir. Bir reklam anonsu, bu reklam anonsu verisini oluşturan byte'ların CC'den

XPM'e gönderilen veri mesajından alınması ve CAM kartına yazılması ile dump edilmektedir.

4.2.2.4 CA Bölgesinin Test Edilmesi

XPM'e gelen CC mesajında bellek testi isteğinde bulunuluyorsa test isteğini belirten bayrağa true atanır ve set_flags prosedürü ile ilgili reklam anonsu disable duruma getirilir.

4.2.2.5 CA'nın Dump Sonrası Test Edilmesi

CC'den gelebilecek bir diğer mesaj da enable_info'dur. Bir reklam anonsu CAM kartına dump edildikten sonra bu reklam anonsuna ilişkin checksum testi yapılır. Eğer test sonucu olumlu ise reklam anonsu enable edilir. Aksi takdirde disable durumda kalır. Bu işlemler yapıldıktan sonra dump dump isteğinin kalktığını belirtmek amacı ile dump isteğini gösteren bayrağa false atanır.

4.2.2.6 CA'nın Query Edilmesi

CC'den gelen istek, bir reklam anonsunun query edilmesi ise, query isteğini belirten bayrağa true atanarak bu işlemin 2 sn.lik ring süresinde yapılması sağlanır.

4.3.1 CAM Kartının Test Edilmesi

4.3 DIAGNOSTIK TESTLER

Diagnostic programlar, bir sistemde meydana gelen yazılım ve donanım hatalarını sezmek üzere hazırlanırlar.

Reklam anonsu projesi için hazırlanan diagnostic programlar, CAM kartı ve CAM kartında bulunan CA verisi üzerinde yürütülmek üzere üç ana test içerir. Bu testler :

* Sistemdeki ILGC'ye bir kartın takılıp takılmadığını ve eğer kart takılı ise bunun CAM kartı olup olmadığını

* CAM kartı belleğine yazma ve bu bellekten okuma işleminin sağlıklı bir şekilde yapıp yapılmadığını

- * CA verisinin zamanla bozulup bozulmadığını ve CA verisinin CAM kartına hatasız olarak yazılıp yazılmadığını, diğer bir deyişle dump işleminin hatasız tamamlanıp tamamlanmadığını kontrol eder.

4.3.1 CAM Kartının Test Edilmesi

4.3.1.1 Yapılan Testin Amacı

Abonelere diğer bir aboneye çağrı yaptıkları zaman reklam anonsu verebilmek için aranılan abonenin bağlı olduğu sanralde bulunan ILGC'de CAM kartının takılı ve işler durumda olması gerekmektedir. Bu nedenle kartın takılı olup olmadığı ve karta erişimin yapılıp yapılamadığının sürekli olarak kontrolü gerekmektedir.

4.3.1.2 Testin Yapılışı

Kartın varlığının ve bu karta erişimin testi iki aşamada yapılmaktadır. İlk olarak XPM için daha önceden yaratılan bir modülde bulunan bir fonksiyonla, ILGC'de takılı bulunan karta erişim yapılıp yapılamadığı kontrol edilmektedir. Bu amaçla hazırlanmış

olan ACCESS_CHECK fonksiyonunun iki parametresi bulunmaktadır. Bunlardan biri erişilmek istenen bellek adresi, diğeri de erişilmek istenen byte sayısını saptayan bir parametredir. Reklam anonsu projesi kart testi için tek bir byte'lik okuma yapılması nedeniyle 2. parametre sürekli olarak sıfırdır. Access_check fonksiyonu NXCOMANN modulunun USES listesinde adı verilen UTILMOD modülünde tanımlanmış olup mevcut bir fonksiyondur. Erişim gerçekleşmediği takdirde, bazı hata yok edici programlara dallanmaktadır.

ACCESS_CHECK fonksiyonu ile erişilmek istenilen adres, CAM kartı tanıtıcısının bulunduğu adres olarak seçilmiştir. Bu adrese erişim gerçekleştikten yani ACCESS_CHECK fonksiyonu olumlu sonuç verdikten sonra, ILGC'de takılı olan kartın gerçekten CAM kartı olup olmadığının kontrol edilmesi gerekmektedir. Bunun içinse kart tanıtıcısı #5A'nın #12FFFE adresinde bulunup bulunmadığı kontrol edilmelidir. Testin ikinci aşaması da olumlu sonuç verirse, ILGC'de bir kartın takılı bulunduğu ve bu kartın da CAM kartı olduğu sonucu elde edilir.

4.3.2 CAM Belleğinin Test Edilmesi

4.3.2.1 Yapılan Testin Amacı

Reklam anonsu için hazırlanan bir diğer test de CAM bellek elemanlarının uygun bir şekilde çalışıp çalışmadığını, bellekte zamanla bir bozulma meydana gelip gelmediğini kontrol eden bellek testidir.

4.3.2.2 Testin Yapılışı

Sistem donanımının işleyişinin, yazılım aracılığıyla kontrol edilebilmesi için geliştirilen testte, test edilecek olan tüm bellek bölgesine #AA gibi bir test örneği atanır. Bu bellek testinin ilk yazma işlemini gerçekleştirir. Belirli bir süre sonra, bu bellek bölgesi sırayla okunarak daha önce yazılan değerlerin geri alınıp alınamadığı kontrol edilir. Eğer testin bu şekilde tamamlanan ilk bölümünün sonucu olumlu ise, aynı işlemler #AA'nın ikili olarak tersi olan #55 test örneği kullanılarak yinelenir. Yazılan değer geri okunduğu takdirde, bellek elemanları uygun olarak çalışıyor demektir. Ancak yazılandan farklı bir değer okunduğunda bu bellek bölgesinin hatalı olduğu anlaşılır ve durumun kullanıcıya

bildirilmesi gerekir. Bellek testinin başarıtlı sayılabilmesi için, her iki aşama da olumlu sonuç vermelidir.

Hic bir hatalı değere rastlanmadığı zaman ilgili reklam alanı bölgesine ilişkin bellek testi sonucu olumlu olur.

Bellek testini gerçekleştirmek üzere dört ayrı fonksiyon tanımlanmıştır. Bu fonksiyonlar XPM PASCAL dili kullanılarak yazılmıştır. Bellek testi fonksiyonları READ sırası ile, FIRST_WRITE_TEST, FIRST_READ_TEST, SECOND_WRITE_TEST ve SECOND_READ_TEST' dir.

FIRST_WRITE_TEST fonksiyonu, reklam verisinin başlangıç adresi belirlendikten sonra, bu adresten itibaren 32K'lık bellek bölgesine #AA örneği yazımı ile yükümlüdür.

FIRST_READ_TEST fonksiyonunda, aynı adreslere erişilerek bu adreslerin içeriğinin #AA olup olmadığı kontrol edilir. Eğer tüm bellek bölgesi bu değeri içeriyorsa, testin ilk aşaması olumlu sonuçlanmış demektir. Ancak hatalı bir değer bulunduğu takdirde, bu reklama ilişkin testin sonucu olumsuz olarak belirlenir ve durum kullanıcıya bildirilir. Ayrıca testin ikinci bölümünü yapmaya gerek kalmaz.

'Checksum Testi' ile kontrol edilecektir. Checksum testi yapılması gereken durumlar aşağıda belirtilmiştir.

FIRST_READ_TEST fonksiyonunun sonucu olumlu olduğu durumda, SECOND_WRITE_TEST çağrılarak, FIRST_WRITE_TEST #55 örnek değeri ile tekrarlanır.

Resitli nedenlerle borçlanmaya başlayabilir. Dışa bir reklamın duplâ işlemi bittikten hemen sonra enable edilecek abonelere duyurulması gerekmektedir.

SECOND_WRITE_TEST fonksiyonundan sonra, SECOND_READ_TEST fonksiyonu çağrılarak yazılan değerlerin geri okunup okunmadığı kontrol edilir. Hiç bir hatalı değere rastlanmadığı zaman ilgili reklam anonsu bölgesine ilişkin bellek testi sonucu olumludur. 32K'lik reklam bölgesine yazma ve okuyarak karşılaştırma yapma işlemleri uzun zaman alacağı için bu işlemleri gerçeklemek üzere 68000 assembler ile adları WRITETEST olan bir prosedür ve READTEST olan bir fonksiyon tanımlanmıştır.

4.3.3 Reklam Anonsu Verisinin Test Edilmesi

4.3.3.1 Yapılan Testin Amacı

CAM kartında bulunan reklam anonsu (CA) verisinde zamanla bir bozulma meydana gelip gelmediği ve bu verinin hatasız olarak dump edilip edilmediği 'Checksum Testi' ile kontrol edilecektir. Checksum testi yapılması gereken durumlar aşağıda belirtilmiştir :

1. Bir CA verisi, CC'den bir ILGC'de bulunan CAM kartına yüklenirken cesitli nedenlerle bozulmaya uğrayabilir. Oysa bir reklamın dump işlemi bittikten hemen sonra enable edilerek abonelere duyurulması gerekmektedir.

0 halde, bir CA verisinin dump işlemi tamamlandıktan sonra, dump işleminin hatasız olarak gerçekleşip gerçekleşmediğini kontrol etmek amacı ile checksum testinin yapılması gerekir. Eğer checksum testi sonucu olumlu ise bu reklam otomatik olarak enable edilir ve abonelere verilmeye başlanır. Test sonucu olumsuz olduğu takdirde, dump işleminin yinelenmesi gerektiği kullanıcıya bildirilir ve reklam enable edilmez.

2. Kullanıcı bir reklamı enable etmek istediği zaman CA verisinin, bu CA disable olduğu sürece bir bozulmaya uğrayıp uğramadığını kontrol etmek amacı ile test edilmesi gerekir. Eğer checksum testi sonucu olumlu ise bu CA enable edilir, değilse disable olarak kalır ve durum kullanıcıya bildirilir.

3. Kullanıcı, abonelere verilen reklam anonsunda bozulma duyulması gibi bir nedenle, bir CA verisine ilişkin checksum testi yapmak isteyebilir.

4. Başlangıç durumuna getirme prosedürü sonunda, enable olan CA verilerine checksum testi yapılarak, sistemin bir aradan sonra yeniden hizmete girmesinden önce bozulma meydana gelip gelmediği kontrol edilmelidir. Test sonucu olumsuz ise bu reklam anonsu disable duruma getirilmeli ve sonuç CC'ye bildirilmelidir.

Belirli zaman aralıkları ile, CA verileri otomatik olarak yapılan checksum testi ile kontrol edilmelidir. Bu işlemi gerçekleştirmek üzere belirli aralıklarla uyanarak ilgili fonksiyon ve prosedürleri çağırmakla yükümlü audit prosesden yararlanılır. Bir CA'ya ilişkin checksum testi sonucu olumsuz olduğu zaman, durum CC'ye bildirilmelidir.

4.3.3.2 Testin Yapılışı

Bir CA verisi üzerinde checksum testinin yapılabilmesi için, CA verisine ilişkin checksum değerinin, bu CA file'i PC'den CC'ye transfer edilmeden önce hesaplanması ve CA file'ı içinde CAM kartına yüklenmesi gerekir.

PC'de oluşturulacak olan CA file'ının yapısı Şekil 12'de görülmektedir.

D7.....D0	
Reklam	1X0000
Anonsunun Adı (ASCII)	
Checksum 1 (MSB)	1X0008
Checksum 1 (LSB)	1X0010
Checksum 2 (MSB)	1X0012
Checksum 2 (LSB)	1X0014
CA'nin durumu	1X0016
	1X0018

Şekil 12. Anons File'ının Yapısı. (X:3,A,B,...,E)

Ses verisinin sayısal veriye dönüştürülmesi, transfer öncesi bu veriye ilişkin checksumin hesaplanmasının diğer bir araştırma grubunun çalışması olması nedeni ile, bu çalışmalar tez dahilinde sunulmayacaktır. Bu checksum, CAM kartında bulunan ve 32000 byte'dan oluşan CA verisine ilişkin checksum, bu 32000 byte'ın word olarak toplanmasından elde edilecek ve meydana gelecek olan taşma ihmal edilecektir. Bu checksum, CA file'ı CAM kartına transfer edilmeden önce hesaplanan checksum ile karşılaştırılmak üzere bir değişkene atılacaktır. Karşılaştırmanın yapılabilmesi için transfer öncesi belirlenen checksumın da aynı yöntemle hesaplanması gerekmektedir. Eğer bu iki checksum birbirine

CAM kartında bulunan ve 32000 byte'dan oluşan CA verisine ilişkin checksum, bu 32000 byte'ın word olarak toplanmasından elde edilecek ve meydana gelecek olan taşma ihmal edilecektir. Bu checksum, CA file'ı CAM kartına transfer edilmeden önce hesaplanan checksum ile karşılaştırılmak üzere bir değişkene atılacaktır. Karşılaştırmanın yapılabilmesi için transfer öncesi belirlenen checksumın da aynı yöntemle hesaplanması gerekmektedir. Eğer bu iki checksum birbirine

66
eşitse checksum testi olumlu, değilse olumsuz olarak sonuçlanacaktır.

Reklam anonsunun abonelere verilmesi sırasında CAM kartı belleğine erişim yapılması gerektiği için, 4 sn.'lik anons verilmesi süresince checksum testi yapılamamaktadır. Geriye kalan 2 sn.'lik ring süresinde de yapılması gereken erişimler nedeniyle bu testin mümkün olduğu kadar kısa sürede tamamlanması gerekmektedir. Bunu sağlamak üzere, testi gerçekleştiren fonksiyon, iki esit parçaya bölünmüştür. Bu bölünme işlemi zamandan kazandırdığı kadar, testin güvenliğini de arttırmıştır. Oluşan iki yeni fonksiyon CSUM_FIRST_TEST ve CSUM_SECOND_TEST olarak adlandırılmıştır.

CSUM_FIRST_TEST bir XPM PASCAL fonksiyondur ve 32000 byte'dan oluşan reklam verisinin ilk 16000 byte'lık bölümünün toplamını elde etmekte ve reklam anonsu file'ında yer alan checksumla bu toplamı karşılaştırıp sonucu boolean olarak, diğer bir deyişle 'true' yada 'false' olarak vermektedir.

CSUM_SECOND_TEST de bir XPM PASCAL fonksiyonudur ve reklam verisinin geri kalan 16000 byte'lık bölümünün toplamını elde ederek anons file'ında yer alan checksumla karşılaştırır ve sonucu boolean olarak verir. Ancak bu fonksiyonun çağrılabilmesi için CSUM_FIRST_TEST fonksiyonunun sonucunun olumlu olması gerekmektedir. Eğer her iki fonksiyonun sonucu da olumlu ise ilişkin reklamın

testden geçtiği söylenebilir. Görüldüğü gibi, checksum testi mümkün olan en kısa sürede gerçekleştirilmektedir.

checksum testinde, test fonksiyonunun her çağrısında 16000 byte'lık verinin toplamı hesaplanmalıdır.

Daha önce de belirtildiği gibi checksum testinin yapılabilmesi için, anons file'ında bulunan checksumın diagnostik test fonksiyonunda insaptanan checksumla aynı yöntemler kullanılarak hesaplanması gerekmektedir. Dolayısıyla, anons file'ındaki checksumın da iki ayrı 16000 byte'lık verinin toplamı şeklinde hesaplanması zorunludur.

Önceden hesaplanmış olan checksum değeri, test fonksiyonuna geçirilerek, testin ilk 16000 byte'lık bölümünde hesaplanmalıdır.

Checksum test fonksiyonlarının hızını arttırmak amacı ile 16000 byte'ın toplamını elde etmek üzere 68000 assembler ile bir alt program hazırlanmıştır. Bu fonksiyon, kendisine parametre olarak geçirilen adresden itibaren, yine parametre olarak geçirilen sayıda byte'ı toplamakta ve sonucu long integer olarak test fonksiyonuna vermektedir. Toplama işlemini gerçekleyen bu fonksiyon CHKSUM olarak adlandırılmıştır.

CHKSUM fonksiyonunun çalışması için, anons file'ında bir parametre olarak geçirilmesi gerekmektedir.

CAM kartında birbirini izleyen bellek gözlerinde bes ayrı anons bulunduğu için, herhangi bir reklama ilişkin checksum testi yapılması gerektiği zaman bu reklama ilişkin başlangıç adresi saptanır ve 68000 assembler kullanılarak hazırlanmış olan CHKSUM fonksiyonuna geçirilir.

CHKSUM fonksiyonunun çalışması için, anons file'ında bir parametre olarak geçirilmesi gerekmektedir. Bu parametre, testin yapılacağı anonsun başlangıç adresi ve toplam uzunluğunu belirtir.

CHKSUM fonksiyonuna geçirilen bir diğ er parametre de kaç byte'in toplanması gerektiğini belirtir. Daha sonra değ inilecek olan audit checksum testinde, test fonksiyonunun her çağ ırılış ında 200 byte'lik toplama yapılması gerekmektedir.

Özetle, üzerinde testin yürütüleceğ i reklamın başlangıç adresi XPM PASCAL fonksiyonda saptanmakta, bu değ er 68000 assembler fonksiyona aktarılmakta ve bu fonksiyonun sonucu olarak alınan toplam, XPM PASCAL fonksiyonda daha önceden hesaplanmış olan checksum değ eri ile karşılaştırılmakta ve reklamın ilk 16000 byte'lik bölümünde bir hata sezildiğ i takdirde teste devam edilmemektedir. İlk bölümden olumlu sonuç alındığ ı durumda ise, ikinci bölüm için de aynı işlemler tekrarlanmakta ve ikinci bölümün sonucu testin sonucunu belirlemektedir.

Daha önce açıklandığı gibi, checksum testi, bir reklam anonsu verisi CCC'den XPM'e dump edildikten sonra dump işleminin hatasız olarak yapılıp yapılmadığını saptamak ve bir reklam anonsu 'enable' duruma geçirilmeden önce 'disable' durumda kaldığı sürece bozulmaya uğrayıp uğramadığını kontrol etmek amacıyla yapılmaktaydı.

Ancak, yukarıda belirtilen iki durumdan biri meydana gelmediğ i sürece reklam anonsu verisinin doğruluğ u kontrol edilmemesi sorunlara neden olacağı için belirli zaman aralıkları ile bu testin tekrarlanması uygun görülmüştür. Bununla beraber, 16000 byte'in toplanması ve bilinen checksum değ eri ile karşılaştırılmasından

oluşan iki ayrı checksum testinin yapılması zaman alacağı için, bu testi gerçekleştirmek amacı ile yeni bir XPM PASCAL prosedür hazırlanması uygun görülmüştür.

AUDIT_CHECKSUM_TEST adı verilen bu prosedür, belirli zaman aralıkları ile audit proses tarafından her çağrılışında 32000 byte'lık verinin 200 byte'lık bölümünü toplamakta ve sonucu global olarak tanımlanmış olan bir değişkene atamaktadır. AUDIT_CHECKSUM_TEST prosedürü, 80 defa çağrılışının sonucunda 16000 byte'lık verinin toplama işlemini tamamlamış olur. Dolayısıyla, daha önceden hesaplanmış olan checksum değeri ile o an ulaşılan toplam değer karşılaştırılır. Eşit değillerse, bu fonksiyonların gelecek çağrılışlarında, bir sonraki reklam anonsu üzerinde aynı testi uygular. Eğer bu iki değer eşitse, işlemler geri kalan 16000 byte'lık veri üzerinde tekrarlanır. AUDIT_CHECKSUM_TEST prosedüründe gereken toplama işlemi de CSUM_FIRST_TEST ve CSUM_SECOND_TEST fonksiyonlarında olduğu gibi CHKSUM adında 68000 assembler ile hazırlanmış olan fonksiyon tarafından gerçekleştirir.

Bir reklam anonsu 'disable' durumdan 'enable' duruma geçirilmek istendiğinde, zorunlu olarak checksum testi yapılacağı için, 'disable' durumda olan reklam anonslarına AUDIT_CHECKSUM_TEST uygulanmaz.

4.4 YAZILIM MODÜLLERİ

XPM yazılımında modüllerin içereceği fonksiyonların aynı programlama dili kullanılarak hazırlanması gerekmekte ve ayrıca bunların ortak bir amaç için kullanılan fonksiyon ve prosedürler olmasına önem verilmektedir.

Bu nedenle, Reklam Anonsu Projesinin tez dahilinde sunulan 'verinin bakımı ve diagnostik testler' bölümü için iki ayrı XPM yazılım modülünün oluşturulması uygun görülmüştür. Ancak 'verinin bakımı' bölümü için gerekli olan prosedürlerin hazırlanmasında XPM Pascal ile ulaşılabilecek hızın yeterli olmasına rağmen, 'diagnostik testler' bölümü için XPM Pascal fonksiyon ve prosedürlerin yanı sıra 68000 assembler kullanarak bazı fonksiyon ve prosedürlere hız kazandırılması gerekmiştir. Dolayısıyla 'diagnostik testler' bölümü için biri XPM Pascal ve diğeri 68000 assembler kullanılarak iki ayrı yazılım modülü hazırlanmıştır. Reklam anonsu verisinin bakımı amacıyla yaratılan ve XPM Pascal kullanılarak hazırlanan prosedürleri içeren XPM yazılım modülüne NXCOMANN, diagnostik testleri içeren XPM Pascal kullanılarak hazırlanan fonksiyon ve prosedürleri içeren XPM yazılım modülüne NXCADIAG, diagnostik testlerde kullanılan ve 68000 assembler ile hazırlanan alt programları içeren XPM yazılım modülüne de ASMNXCAD adı verilmiştir.

Yazılım tasarımı, çeşitli aşamalara gelen tasarımın sunulması, daha sonra yapılacak olan tasarımda bu tasarımda kullanılan tekniklerin gözden geçirilebilmesi amacıyla saklanması dikkate alınması gereken iki önemli noktadır.

Yazılımının sahip olduğu yapı, tasarımın sunulacağı kişilerin programlama konusundaki bilgi düzeyleri gibi konulara bağlı olarak tasarım biçimi belirlenmelidir. Mevcut tasarım sunuşlarına örnek olarak kod listeleme, tablolar oluşturma, sözde kod (pseudo code), akış diyagramı (flowchart), yapısal diyagramlar, HIPO diyagramları, kontrol grafiği, vb verilebilir.

Bu tasarım sunuş yapılarından en eski ve en bilineni akış diyagramıdır. Evrensel bir sunuş şekli olan akış diyagramı, görsel ve programlama konusunda eğitim görmemiş bir kişinin anlayabileceği yapısı nedeniyle birçok yazılım tasarımında tercih edilmektedir. Bununla beraber çok yer kaplaması, yapısal değişiminin emek gerektirmesi nedeni ile güncelleştirilmesinin zorluğu ve alt programların, kesmelerin resimlenmesinin yarattığı sorunlar nedeni ile her yazılım tasarımının sunulmasında kullanılmaktadır.

Diğer bir sunuş şekli olan sözde kod ise nasıl hazırlanacağını kolay öğrenilmesi, yapısının İngilizce cümle yapısına uygunluğu, çok az yada hiç bir değişim yapılmaksızın bölümlerinin kod üzerinde taşınabilirliği, akış diyagramından daha yoğun olması, bilgisayarda saklama ve güncelleştirmedeki kolaylığı nedeni ile birçok tasarım sunuşunda kullanılmaktadır. Akış diyagramında olduğu gibi, sözde kodun da şekilsel olmama ve standart yapısının bulunmaması gibi bazı dezavantajları mevcuttur.

HIPO (Hierarchy plus Input-Process-Output) diyagramları da diğer bir sunuş şeklini oluşturmaktadır. Tasarımın başlangıcı, ayrıntılı bir yapı kazanması, tasarım eleştirisi ve değerlendirmesi, tasarımın

test edilmesi ve bakımı gibi her tasarım aşamasında bu diyagramlardan yararlanılabilir.

Kullanıldığı aşamaya bağlı olarak, HIPO diyagramları program modülleri arasındaki ilişkileri; giriş değişkenleri, işlem fonksiyonları ve çıkış değişkenleri arasındaki bağlantıyı; yada yapılan tasarıma özgü noktaları verecek yapıya sahip olabilir.

1975 yılında yayınlanan IBM memo'da, HIPO diyagramlarında kullanılabilecek yüklemeler (verbs) belirtilmiştir. Bunlara ekle, başla, sakla, test et, seç, tahsis et, yaz, vb. yüklemeler örnek verilebilir. Tasarımın her seviyesi, özellikle son aşamalarda uygun sunuş yapısına sahip olan HIPO diyagramlarının diğer bir üstünlüğünde giriş ve çıkışların vurgulanabilmesidir. Ayrıca programlama konusunda eğitim görmemiş kişilerin kolayca anlayabileceği HIPO diyagramlarının diğer sunuş biçimlerinde olduğu gibi, tasarımın tümüne uygulandığında çok yer gerektirmesi, hazırlanabilmeleri için başlangıçta birkaç pratik yöntemin öğrenilmesi ve yapısal değişimlerinin zorluğu nedeniyle kolay güncelleştirilmesi gibi bazı dezavantajları bulunmaktadır.

Sunulan tezin kodunun görsel yapıda açıklanmasında, tasarım aşamalarında sözde kod kullanılmasına rağmen tasarımın son aşamasında oluşturulan HIPO diyagramlarından yararlanılmıştır. Bu diyagramların proses bölümünde verilen tümceler tasarım aşamasında kullanılan sözde kodlardan alınmıştır. HIPO diyagramlarında kullanılan koyu renkteki oklar akış kontrolünü, bir blokta başlayıp bir sonraki blokta biten oklarsa veri akışını temsil etmektedir.

4.4.1 Veri Bakım Modülleri

XPM Pascal Kullanılarak Hazırlanan NXCOMANN Modülü :

Modul Adı : NXCOMANN	
Tanım : Bu modul CA verisinin çevresel birimlerde	
birimlerde bakımını gerçekler.	

parallel unit NXCOMANN;

interface

(Bu modülün arabağlaşım bölümünün başlangıcını belirtir.)

USES UTILCONS, OSXPM, UTILTYPE, UTILMOD, UTIL, UTILPCODE,

MONIF, BASEMON, MSGUTIL, SPCHNLS, MTSDEFS, LNGIPC,

TONEDIAG;

(USES listesinde bu modül tarafından kullanılan modüllerin isimleri yer almıştır.)

FUNCTION chksum (add_data : longint; num : integer) : longint;

(chksum fonksiyonunun kaynak kodu, ASMNXCAD modülünde

bulunmaktadır. Bu fonksiyon, kendisine 'num'

parametresi ile geçirilen sayıda byte'ı, 'add_data'

parametresi ile geçirilen adresten itibaren

toplayarak sonucu çağrıldığı yere aktarır.

chksum fonksiyonu, csum_first_test yada csum_second_test

prosedürü tarafından çağrılıyorsa 'num' 16000'e ,

audit_checksum_test tarafından çağrılıyorsa 200'e

eşittir.)

PROCEDURE write_test (start : longint; test_pattern : integer);

(write_test prosedürünün kaynak kodu, ASMNXCAD

prosedüründe bulunmaktadır. Bu fonksiyon, daha sonra

okunarak test edilmek üzere, bir reklam anonsuna ilişkin tüm verilerin bulunduğu bellek bölgesine, bir test örneği yazar. write_test prosedürü, first_write_test fonksiyonu tarafından çağırılıyorsa, bu test örneği #AA, second_write_test fonksiyonu tarafından çağırılıyorsa #AA'nın ikili olarak tersi olan #55'dir. Test örneği değeri, bu prosedüre 'test_pattern' parametresi ile geçirilir. 'start' parametresi ise, yazma işleminin yapılacağı bellek bölgesinin başlangıç adresini belirtir. Bu adresten itibaren 32K byte'lık bölgeye test örneği yazılır. }

FUNCTION read_test (start : longint; test_pattern : integer) : integer

(read_test fonksiyonunun kaynak kodu, ASMNXCAD modülünde bulunmaktadır. Bu fonksiyon write_test prosedürünün çağırılarak test örneğinin ilgili bellek bölgesine yazılmasından belirli bir süre sonra, bu değer doğru olarak yazılıp okunabildiğini kontrol etmek amacıyla hazırlanmıştır. Daha önce write_test prosedürü ile yazma işleminin gerçekleştirildiği tüm bellek bölgesinin içeriği, bu test örneği ile karşılaştırılır. Bir uyumsuzluk sezildiği takdirde, fonksiyon 0, tüm değerlerin doğru olarak okunması halinde ise 1 sonucunu vermektedir. Karşılaştırma yapılacak olan değer 'test pattern', karşılaştırmaların başlatıldığı adres ise 'start' parametreleri ile bu fonksiyona geçirilir. Her iki parametre de write_test prosedüründe kullanılan bu değerlerin kaybolmaması

için global olarak tanımlanmıştır. }

PROCEDURE init_nxcomann;

IMPLEMENTATION

(Bu modülün gerçekleştirme bölümünün başlangıcını belirtir.)

CONST

(Global olan sabitler tanımlanacaktır.)

ca_hndlr_high = #0012;

ca_hndlr_low = #FFFE;

(CAM kartının tanıtıcısı olan #5A, #12FFFE

adresindedir. Ayrıca, CAM kartına birşey yazılacağı

zaman CAM kartının 'disable' edilmesi gerekmektedir.

'Disable' etme işlemi bu adrese #FFFF yazılarak

yapılmaktadır. }

cam1_high = #0013;

cam2_high = #001a;

cam3_high = #001b;

cam4_high = #001e;

cam5_high = #001f;

(cam1_high, ... cam5_high adresleri, beş reklam

anonsu bölgesinin başlangıç adreslerinin

en anlamlı byte'larıdır. }

cam_low_shared = #0600;

(Bu sabit, reklam anonsu ses verisinin

sayısallaştırılması ile elde edilen verinin

başladığı bellek bölgesi adresinin en az

anlamlı byte'ıdır. }

netas_tbl = #50;

netas_mtc = #51;

(DMS'de CC'den XPM'e gelen mesajların, veri mesajı mı yoksa bakım mesajı mı olduğunu XPM'in ayırd edebilmesi için CC'den XPM'e gelen bu mesajın bir bölgesi bu ayırımın yapılabilmesi amacıyla ayrılmıştır.

Bu bölümde netas_tbt varsa mesaj veri, netas_mtc varsa bakım mesajıdır.)

cid_com_ann = #24;

(CC ve XPM arasında mesaj gönderimi, betirli bir haberleşme protokolüne bağlı olarak yapılmaktadır. Mesajın ilgili bölümünde bu haberleşme tanıtıcısı gönderildiği takdirde gönderilen mesajın, reklam anonsuna ilişkin olduğu mesajı alan bölüm (CC yada XPM) tarafından anlaşılmaktadır.)

mt_gen_hand = #02;

node_cons = #0b;

(XPM mesaj yapısında bulunması gereken sabitlerdir.)

unsol_id = #44 ;

(XPM'den CC'ye, CC'nin herhangi bir isteği olmadan gönderilen mesajlar yani 'unsolicitor' mesajlar için mesajın ilgili bölümünde bu sabitin yer alması gerekmektedir.)

disable_ann = 0;

(ilgili reklam anonsunun 'disable' edilme isteğini belirtir.)

enable_ann = 1;

(ilgili reklam anonsunun 'enable' edilme isteğini belirtir.)

dump_request = 2;

{ ilgili reklam anonsunun 'dump' edilme
isteğini belirtir. }

test_req = 3;

{ ilgili reklam anonsunun bulunduğu bellek
bölgesinin 'test' edilme isteğini belirtir. }

query_req = 4;

{ ilgili reklam anonsunun 'query' edilme isteğini
belirtir. }

enable_info = 5;

{ ilgili reklam anonsunun dump işlemi bittikten sonra,
CC otomatik olarak XPM'e bu mesajı gönderir. }

{ NOT : Kullanıcının isteği üzerine yapılması
gereken işlemler, CC'nin XPM'e gönderdiği
mesajlar ile XPM'e bildirilir. Yukarıda
tanımlanan sabitler, CC'den XPM'e
gönderilebilecek istek mesajlarına
parametre olarak geçirilmektedir. }

req_done = 0;

{ CC'den gönderilen isteğin yapıldığını belirtir. }

req_failed = 1;

{ CC'den gönderilen isteğin yapılmadığını belirtir. }

test_reply = 2;

{ CC'nin test isteği üzerine yapılan test sonucunu bildirir. }

query_reply = 3;

{ CC'nin query isteği üzerine yapılan test sonucunu bildirir. }

cam_not_present = 4;

{ CAM kartının takılı olmadığını yada bu karta erişim

yapılamadığını belirtir.)

(NOT : XPM, CC'nin isteği üzerine ve belirli

aralıklarla yaptığı işlemlerin sonucunu CC'ye
bildirmek üzere CC'ye mesaj gönderir..

Yukarıda tanımlanan sabitler, bu mesajlara
parametre olarak geçirilmektedir.)

flags = PACKED RECORD

CASE int OF

0: (first_ann : BOOLEAN;

second_ann : BOOLEAN;

third_ann : BOOLEAN;

fourth_ann : BOOLEAN;

fifth_ann : BOOLEAN;

(first_ann, ... fifth_ann bayrakları, ilgili reklam
anonsunun 'disable' yada 'enable' durumda olduğunu
göstermek için kullanılmışlardır. Bu bayrak true
ise 'enable', false ise 'disable' durumdadır.)

dump_ann : BOOLEAN;

(Bir dump işleminin sürmekte olduğunu gösteren
bayraktır. Bu bayrak true olduğu sürece CAM kartı
üzerinde başka hiç bir işlem yapılamaz.)

res_dump : BOOLEAN;

(Bu bayrağın true olması, dump isteği olduğunu
gösterir.)

chnq_ann : BOOLEAN;

(Herhangi bir reklam anonsunun durumunda bir
değişiklik yapıp yapılmadığını bu bayrak gösterir.
chnq_ann true ise reklam anonslarından biri
'enable' iken 'disable' yada 'disable' iken 'enable'

yapılmış demektir. Bu bayrak kontrol edilerek, aboneye verilen reklam anonslarının sırasında herhangi bir değişiklik yapıp yapılmayacağına karar verilir.)

req_chkck : BOOLEAN;

(Bu bayrağın true olması, CC'nin checksum testi isteği olduğunu gösterir.)

req_query : BOOLEAN;

(Bu bayrağın true olması, CC'nin query testi isteği olduğunu gösterir.)

req_test : BOOLEAN;

(Bu bayrağın true olması, CC'nin bellek testi isteği olduğunu gösterir.)

ca_id : BYTE;

(Reklam anonsu numarasının atanması içindir.)

solic_no : BYTE;

(XPM'in CC'ye gönderdiği mesajın, CC'nin hangi isteği üzerine gönderdiğini belirtmek üzere kullanılmaktadır.)

card_stat : BOOLEAN;

(Bu bayrağın true olması, CAM kartının takılı ve erişilebilir olduğunu göstermektedir.)

filler : 0..2);

(Daha sonra kullanılmak üzere boş alanlar ayrılmıştır.)

1: (all : longint);

(Tüm bayraklara aynı anda atama yapabilmek için bu yapı tanımlanmıştır.)

END;

cam_ptr_type = PACKED RECORD

CASE int OF

0 : (msw : integer;


```
    lsw : integer );
```

```
1 : ( all : integer );
```

```
END; ( record )
```

(CAM kartına bir verinin yazılabilmesi ve okunabilmesi için tanımlanmıştır. cam_ptr_type.msw'ye erişilmek istenen adresin en anlamlı byte'ı ve cam_ptr_type.lsw'ye bu adresin en az anlamlı byte'ı yazıldıktan sonra, cam_ptr_type.all' bu adresi point eder.)

```
check_ptr = PACKED RECORD
```

```
    CASE int OF
```

```
        0 : (high : byte ;
```

```
            low : byte) ;
```

```
        1 : (word : integer);
```

```
    END ; ( record )
```

(Kartın çift adreslerindeki değer daha öncede değinilen nedenle #FF olarak okunmaktadır. CA file'ı PC'den gönderilmeden önce hesaplanan ve bu file içinde gönderilen checksum değerinin word yapısında olması ve çift adreslere herhangi bir değer yazılamaması nedeniyle bir word'luk checksum değeri, ard arda gelen iki word'un en az anlamlı byte'larında yer almaktadır. Checksum değerinin en anlamlı byte'ı düşük adresli, en yüksek anlamlı byte'ı da yüksek adresli bellek bölgesinde yer almaktadır. Bu yapı kullanılarak bu iki ardışıl adresden, checksum değeri tek bir word halinde elde edilebilmektedir.)

VAR

(Global olan değişkenler tanımlanacaktır.)

```
com_flags      : flags;
```

```
ipc_m_idx      : ipc_idx;
```


idx : ipc_idx;

{ IPC için indeks olarak kullanmak üzere tanımlanmışlardır.}

ptr_cam : cam_ptr_type;

cam_low : integer;

ann_msg : mts_blk_ptr;

chsum_stat : byte;

{ Checksum testinin fazlarını belirtir.}

bool_mate : boolean;

{ CC'den gelen istekler aktif birimde olumlu olarak

yanıtlanabilirse, aynı işlem eş birim için de yapılır.

Ancak CC'nin isteği yerine getirilemezse bu değişkene

false atanarak, işlemin es birim için yapılması önlenir.}

ptr_start : cam_ptr_type;

ptr_last : cam_ptr_type;

{ Bu bayraklar, bellek testinde bir CA verisinin başlangıç ve

bitis adreslerini gösterir. Bellek testi, birbirinden

bağımsız iki fonksiyon ve prosedürden oluştuğu için, adres

değerlerinin bu fonksiyon ve prosedürlerin çağrılışı

sırasında kaybolmaması gerekmektedir. Bu nedenle adresler,

global değişkenler olarak tanımlanmışlardır.}

fst_write : boolean ;

fst_read : boolean ;

snd_write : boolean ;

snd_read : boolean ;

{ Bu bayraklar hangi bellek testinin tamamlandığını, çağrılma

sırasının hangi fonksiyon yada prosedürde olduğunu göstermek

amacıyla tanımlanmıştır.}

check : check_ptr ;

{ CA file'ı PC'den gönderilmeden önce hesaplanan checksum değeridir.}

au_data : longint ;

au_chk : cam_ptr_type;

{ Bu iki değer, audit checksum testi sırasında, reklam anonsu verisinin ve bu reklam anonsuna ilişkin checksum'ın bulunduğu bellek bölgelerinin başlangıç adresleridir.}

count : integer ;

{ Audit checksum testinin kaçınıcı adımda olduğunu gösterir.}

sum : longint ;

{ Çağırın fonksiyona bağılı olarak 16000 yada 200 byte'lık verinin toplamını verir.}

ca_iden : integer ;

{ audit_checksum_test için reklam anonsu numarası}

add_data : longint ;

add_chk : cam_ptr_type;

{ csum_first_test ve csum_second_test için kullanılan adresler. }

start : longint ;

{ Bellek testi için başlangıç adresi }


```

| Adı : PROCEDURE send_msg_to_cc |
| Tanım : XPM'den CC'ye mesaj gönderir. |
| Parametreler : |
| comm_no (byte)-Reklam anonsu numarası. |
| request (byte)-CC'nin hangi isteği üzerine |
| bu mesajın gönderildiğini belirtir. |
| reply (byte)-Yapılan işlemin sonucunu belirtir. |
|

```

```

PROCEDURE send_msg_to_cc (comm_no      : byte;
                           request      : byte;
                           reply        : byte;
                           solicitor_no : byte);

VAR
    idx      : ipc_idx;
    ( IPC indeksini içerir.)

BEGIN ( send_msg_to_cc )

    IF (mod_com_area?.module_activity = active)
    ( Mesaj gönderilen birimin aktif birim olması gerekmektedir.
      USES listesindeki bir modülde tanımlanan bu değişkenle
      birimin aktif olup olmadığı saptanır.)

    THEN

        BEGIN ( if )

            IF ipc_get(idx)
            ( Bir IPC tamponu elde edilir.)

            THEN

                BIND ipcann TO ipc_bfs?[idx] IN
                ( Bu prosedürün içinde geçerli olmak üzere
                  isim değişikliği yapılmıştır.)

```


BEGIN { if }

ipcann.dest := cid_cc;

{ Mesajın varış yeri olarak CC verildi.}

ipcann.src := cid_com_ann;

{ Mesaj kaynağının XPM olduğunu ve mesajın
reklam anonsu ile ilgili olduğunu belirtir.}

IF solicitor_no = #ff then

ipcann.intmsg.msg[1] := unsol_id

{ Eğer fonksiyona solicitor_no olarak #FF
geçirilirse, bu mesaj CC tarafından
unsolicitor olarak değerlendirilir.}

ELSE

ipcann.intmsg.msg[1] := mt_gen_hand;

{ Mesaj solicitor ise mesajın ilk bölgesine
bu değer atanır. CC mesajın ilk bölgesinde
bu değere rastladığında mesajın 3. bölgesindeki
solicitor numarasını göz önüne alır.}

ipcann.intmsg.msg[2] := node_cons;

{ ILGC'nin düğüm numarası CC'ye mesajın
2. bölgesi ile bildirilir.}

ipcann.intmsg.msg[3] := solicitor_no;

{ Gelen mesajın CC'nin hangi mesajına yanıt
olduğunu belirtir.}

ipcann.intmsg.msg[4] := reply;

{ Reply, bu mesajda CC'nin isteği üzerine
yapılan işlemin sonucudur ve req_done,
req_failed, test_reply, query_reply,
cam_not_present olabilir.}


```

ipcann.intmsg.msg[5] := request;

( Request, bu mesajda CC'nin hangi isteğine
  yanıt verildiğine bağlı olarak disable_ann,
  enable_ann, query_req, dump_request, test_req,
  enable info olabilir.)

ipcann.intmsg.msg[6] := comm no;

( Yanıtın hangi reklam anonsuna ilişkin olduğu
  comm no ile belirtilir.)

IF (mod_com_area.module_k stats.imc ok)
( Bu değişken USES listesinde adı verilen MTSDEF
  modülünde tanımlanmıştır. Aktif birimle yedek
  birimin haberleşip haberleşemediğini gösterir.
  True ise iki birim haberleşiyor, false ise
  haberleşemiyor demektir. Durum mesajın 7. bölgesi
  kullanılarak CC'ye bildirilir.)
THEN
ipcann.intmsg.msg[7] := 1
( Aktif ve yedek birim haberleşmektedir.)
ELSE
ipcann.intmsg.msg[7] := 0;
( Aktif ve yedek birim haberleşememektedir.)
ipcann.intmsg.int_node := my_intnode;
( CC'ye mesajın ILGC'nin hangi biriminden
  gönderildiğini bildirir. 0 yada 1 olabilir.)
ipcann.intmsg.length := 16;
( Mesaj uzunluğu 16'dır.)
ipc_send(idx);
( Doldurulan IPC tamponu gönderilir.)
END ( if )

```


ELSE

swerr2('NO Buffer Available', NIL);

(Tampon elde edilemezse swerr verir. Swerr (software error), programın geçtiği noktaların izlenmesinde kullanılır.)

END; (if)

END;


```

| Adı : PROCEDURE send_req_to_mate
| Tanım : Eş birime istekleri gönderir.
| Parametre :
| idx (mts_idx)- IPC indeksini içerir,
|

```

```

PROCEDURE send_req_to_mate (idx : mts_idx);
BEGIN ( send_req_to_mate )

    BIND ipm TO ann_msg7.ipc IN

    ( ann_msg7.ipc'ye adının uzun olması nedeniyle, sadece
      bu prosedür de geçerli olmak üzere ipm adı verilmiştir.)

    BEGIN

        IF (mod_com_area7.module_activity = active)

            ( Mesaj gönderilen birimin aktif birim olup olmadığı
              kontrol edilir. Sadece aktif birimden mesaj gönderilir.)

            THEN

                BEGIN ( if )

                    ipm.dest          := cid_imc;

                    ( Mesajın varış yerinin eş birim olduğu belirtilir.)

                    ipm.intmsg.msrc    := cid_com_ann;

                    ipm.intmsg.mdest   := cid_com_ann;

                    ( Mesajın gönderildiği yerin XPM olduğu ve bu mesajın
                      reklam anonsu ile ilgili olduğu belirtilir.)

                    ipc_send(idx);

                    ( Doludurulan IPC tamponu gönderilir.)

                END ( if )

            ELSE

                ipc_release(idx);

                (IPC tamponu serbest bırakılır.)

```


END;

END;


```

| Adı : PROCEDURE set_flags |
| Tanım : Reklam anonsları için tanımlanan |
| bayraklara değer atar. |
| Parametreler : |
| status_parm (boolean)-Bayraklara atanacak olan |
| değerdir. |

```

```
PROCEDURE set_flags (status_parm : boolean);
```

```
BEGIN ( set_flags )
```

```
CASE com_flags.ca_id OF
```

```
0 : BEGIN
```

```
com_flags.first_ann := status_parm;
```

```
com_flags.second_ann := status_parm;
```

```
com_flags.third_ann := status_parm;
```

```
com_flags.fourth_ann := status_parm;
```

```
com_flags.fifth_ann := status_parm;
```

```
END;
```

```
{ Tüm reklam anonslarına değer atanacağını belirtir.}
```

```
1 : com_flags.first_ann := status_parm;
```

```
{1. reklam anonsuna status_parm atanır.}
```

```
2 : com_flags.second_ann := status_parm;
```

```
{2. reklam anonsuna status_parm atanır.}
```

```
3 : com_flags.third_ann := status_parm;
```

```
{3. reklam anonsuna status_parm atanır.}
```

```
4 : com_flags.fourth_ann := status_parm;
```

```
{4. reklam anonsuna status_parm atanır.}
```

```
5 : com_flags.fifth_ann := status_parm;
```

```
{5. reklam anonsuna status_parm atanır.}
```


OTHERWISE

swerr2('Invalid Ann Number', NIL);

{ Hatalı anons numarası girilmişse swerr verir. }

END; (case)

END;


```

| Adı : PROCEDURE an_nam_chsum
| Tanım : Reklam anonsu adını ve checksum
| değerini CAM kartına koyar ve enable_hdler
| prosedürünün çağrılabilmesi için req_chck
| bayrağına 'true' atar.
| Parametre :
| idx (mts_idx)-IPC indeksini içerir.

```

```
PROCEDURE an_nam_chsum (idx : mts_idx);
```

```
VAR
```

```
    index_num : integer;
```

```
BEGIN ( an_nam_chsum )
```

```
    IF buff_get_ptr(ann_msg, idx)
```

(Bu prosedür, USES listesinde adı verilen MTSDEF modülünde tanımlanmıştır ve tampona bir mesaj gelmişse true verir. Gelen mesaj aşağıda görüldüğü şekilde değerlendirilir.)

```
THEN
```

```
    BIND ipm TO ann_msg7.ipc IN
```

(ann_msg7.ipc'ye adının uzun olması nedeniyle, sadece bu prosedür de geçerli olmak üzere ipm adı verilmiştir.)

```
    BEGIN ( if )
```

```
        cam_low := 0;
```

```
        CASE ipm.intmsg.msg[7] OF
```

{ Gelen mesajdan reklam anonsu numarası elde edilir.}

```
            1 : ptr_cam.msw := cam1_high;
```

```
            2 : ptr_cam.msw := cam2_high;
```

```
            3 : ptr_cam.msw := cam3_high;
```



```

4 : ptr_cam.msw := cam4_high;
5 : ptr_cam.msw := cam5_high
END;   ( case )

(Reklam anonsunun adının yazılacağı adres belirlendi.)
FOR index_num := 0 TO 7 DO
  BEGIN ( for )
    ptr_cam.lsw := cam_low;
    ptr_cam.all := ipm.intmsg.msg[8 + index_num];
    cam_low := cam_low + 2;
  END;   ( for )
  (Reklam anonsunun adı karta yazıldı.)
  FOR index_num := 0 TO 3 DO
    BEGIN ( for )
      ptr_cam.lsw := cam_low;
      ptr_cam.all := ipm.intmsg.msg[9 + index_num];
      cam_low := cam_low + 2;
    END;   ( for )
    (Checksum değeri de karta yazıldı.)
    com_flags.dump_ann := false;
    (Dump işleminin bittiğini belirtmek üzere bu bayrağa
     false atandı.)
    chsum_stat := 0;
    ( Daha sonra checksum testinin sonucunun atanacağı
      değışkene ilk değeri atandı.)
    com_flags.req_chk := true;
    ( Task tarafından checksum testinin başlatılması için
      bu bayrağa true atandı.)
  END;

```


END;

PROCEDURE start_date

DATE: 1999-11-11 10:10:10

DATE: 1999-11-11 10:10:10

PROCEDURE start_date

DATE: 1999-11-11 10:10:10

DATE: 1999-11-11 10:10:10

DATE: 1999-11-11 10:10:10

DATE: 1999-11-11 10:10:10

DATE: 1999-11-11 10:10:10

DATE: 1999-11-11 10:10:10

DATE: 1999-11-11 10:10:10

DATE: 1999-11-11 10:10:10

DATE: 1999-11-11 10:10:10

DATE:


```

-----
| Adı : PROCEDURE start_data |
| Tanım : Dump işlemini başlatmak için |
| send_msg_to_cc prosedürünü çağırır. |
|-----

```

```
PROCEDURE start_data ;
```

```
BEGIN ( start_data )
```

```
com_flags.dump_ann := true;
```

```
( Task tarafından çağrılabilmek için dump isteği  
olduğunu belirtir.)
```

```
com_flags.req_dump := false;
```

```
( Dump isteğinin kalktığını gösterir.)
```

```
send_msg_to_cc(com_flags.ca_id, dump_request, req_done,  
com_flags.solic_no);
```

```
( Dump isteğinin yerine getirildiğini CC'ye bildirir.)
```

```
END;
```



```

|
| Adı : PROCEDURE enable_hdler
|
| Tanım : Checksum testi fonksiyonlarını
|
| çağırır ve ilgili reklam anonsunu test
|
| sonucu olumlu ise 'enable' durumuna getirir.
|
|

```

```
PROCEDURE enable_hdler;
```

```
BEGIN ( enable_hdler )
```

```
IF buff_get_ptr(ann_msg, idx)
```

```
( Bu prosedür, tampona bir mesaj gelmişse true verir.
```

```
Gelen mesaj aşağıda görüldüğü şekilde değerlendirilir.)
```

```
THEN
```

```
    BIND ipm TO ann_msg".ipc IN
```

```
( ann_msg".ipc'ye adının uzun olması nedeniyle, sadece
```

```
bu prosedür de geçerli olmak üzere ipm adi verilmistir.)
```

```
    BEGIN
```

```
com_flags.req_chck := false;
```

```
( Checksum testi isteğinin kalktığı bu flaga false
```

```
atanarak belirtilir.)
```

```
CASE chsum_stat OF
```

```
( Checksum testinin sonucuna göre aşağıdaki
```

```
işlemler yapılır.)
```

```
0 :
```

```
    BEGIN
```

```
        IF csum_first_test(com_flags.ca_id)
```

```
( Reklam anonsunun ilk yarısı için yapılan
```

```
checksum testi olumlu sonuç verirse aşağıdaki
```

```
işlemler yapılır.)
```

```
    THEN
```


BEGIN

chsum_stat := 1;

{ Checksum testinin reklam anonsu verisinin ikinci yarısı için de bu testin yapılabilmesi amacıyla bu değişkene 1 atandı.}

com_flags.req_chck := true;

{ Checksum testi isteği olduğu bildirilir.}

END

ELSE

BEGIN

IF (ipm.src = cid_cc)

{ Mesajın CC'den gelip gelmediği kontrol edilir.}

THEN

send_msg_to_cc(com_flags.ca_id, enable_ann, req_failed, com_flags.solic_no);

{ Reklam anonsu verisinin ilk yarısı için yapılan checksum testi olumlu sonuç vermezse durum CC'ye bildirilir.}

END;

END;

1 :

BEGIN

IF csum_second_test(com_flags.ca_id)

{ Reklam anonsu verisinin ikinci yarısına ilişkin checksum testi yapılır.}

THEN

BEGIN

set_flags(true);

{ Checksum testinden geçen reklam anonslarına


```

        ilişkin bayraklara true atanır.)
com_flags.chng_ann := true;
( Bir reklam anonsunun durumunda değişiklik
  yapıldığı bildirilir.)
IF (ipm.src = cid_cc)
( Mesajın CC'den gelip gelmediği kontrol edilir.)
THEN
    send_msg_to_cc(com_flags.ca_id, enable_ann, req_done,
com_flags.solic_no);
    ( CC'ye, gelen enable isteğinin yerine getirildiğine
      dair onay mesaj gönderilir.)
END
ELSE
IF (ipm.src = cid_cc)
( Mesajın CC'den gelip gelmediği kontrol edilir.)
THEN
    send_msg_to_cc(com_flags.ca_id, enable_ann, req_failed,
com_flags.solic_no);
    ( CC'ye, gelen enable isteğinin yerine getirilemediğini
      bildiren bir mesaj gönderilir.)
END;
OTHERWISE
    swerr2 ('Invalid Parameter ', NIL);
    ( Parametrenin hatalı olduğuna dair swerr verir.)
END;
END;
END;
END;

```



```

| Adı : PROCEDURE comm_hdlr |
| Tanım : CC'den gelen dump, enable yada |
| disable isteklerini değerlendirir. |
| Parametre : |
| midx (mts_idx) - IPC indeksi |

```

```

PROCEDURE comm_hdlr (midx : mts_idx);
BEGIN ( comm_hdlr )

    IF buff_get_ptr(ann_msg, midx)
    ( Bu prosedür, tampona bir mesaj gelmişse true verir.
      Gelen mesaj aşağıda görüldüğü şekilde değerlendirilir.)
    THEN

        BIND ipm TO ann_msg%.ipc IN
        ( ann_msg%.ipc'ye adının uzun olması nedeniyle, sadece
          bu prosedür de geçerli olmak üzere ipm adı verilmiştir.)
        BEGIN ( if )

            com_flags.solic_no := ipm.intmsg.msg[6];
            ( Gelen mesaja ilişkin solicitor numarası alınır.)

            com_flags.ca_id := ipm.intmsg.msg[7];
            ( Mesajın ilgili olduğu anons numarası alınır.)

            CASE ipm.intmsg.msg[3] OF

                ( CC'den gelen isteğe göre yapılması gereken
                  işlemler saptanır.)

                disable_ann :
                ( Gelen istek bir reklam anonsunun disable edilmesi
                  ise aşağıdaki işlemler yapılır.)

                BEGIN

```



```
set_flags(false);
```

```
{ İlgili flağa false atanır.}
```

```
IF (ipm.src = cid_cc)
```

```
{ Mesajın CC'den gelip gelmediği kontrol edilir.}
```

```
THEN
```

```
    send_msg_to_cc(com_flags.ca_id, disable_ann, req_done,  
    com_flags.solic_no);
```

```
{ Reklamın disable edildiği CC'ye bildirilir.}
```

```
END;
```

```
enable_ann :
```

```
{ Gelen istek bir reklam anonsunun enable edilmesi  
ise aşağıdaki işlemler yapılır.}
```

```
BEGIN
```

```
    idx := midx;
```

```
    com_flags.chng_ann := false;
```

```
{ Bir reklam anonsunun durumunda değişiklik  
yapıldığını gösteren bayrağa ilk değer atandı.}
```

```
    chsum_stat := 0;
```

```
{ Reklam anonsu verisinin ilk yarısına ilişkin  
checksum testi yapan fonksiyonun çağrılabilmesi için  
bu değişkene 0 atandı.}
```

```
    com_flags.req_chck := true;
```

```
{ Checksum testi isteği olduğu bu bayrağa true  
atanarak bildirilir.}
```

```
END;
```

```
dump_request :
```

```
{ Gelen istek bir reklam anonsunun dump edilmesi  
ise aşağıdaki işlemler yapılır.}
```

```
BEGIN
```




```

com_flags.req_dump := true;
( Dump isteği olduğu bu bayrağa true
  atanarak bildirilir.)
set_flags(false);
( İlgili bayrağa false atanır.)
END;

test_req :
( Gelen istek bir reklam anonsuna ilişkin bellek
  bölgesinin test edilmesi ise aşağıdaki işlemler yapılır.)
com_flags.req_test := true;
( Bellek testi isteği olduğu bu bayrağa true
  atanarak bildirilir.)

query_req :
( Gelen istek bir reklam anonsunun query edilmesi
  ise aşağıdaki işlemler yapılır.)
com_flags.req_query := true;
( Query isteği olduğu bu bayrağa true
  atanarak bildirilir.)

enable_info :
( Gelen istek bir reklam anonsunun dump sonrası
  enable edilmesi ise aşağıdaki işlemler yapılır.)
BEGIN
  idx := midx;

  an_nam_chsum(idx);
  enable_hdlr;

END;

OTHERWISE
BEGIN

```



```

com_flags.chng_ann := false;

( Bir reklam anonsunun durumunda deęişiklik
  yapılmadığını gösterir.)

bool_mate := false;

( CC'nin isteęi yerine getirilemezse bu deęişkene
  false atanarak, işlemin eş birim için yapılması
  önlenir.)

IF (ipm.src = cid_cc)

( Mesajın CC'den gelip gelmedięi kontrol edilir.)

THEN

send_msg_to_cc(com_flags.ca_id, ipm.intmsg.msg[3], req_faile

com_flags.solic_no);

( Yukarıda belirtilenlerden farklı bir istek gelirse
  gelen isteęin yapılmadığı CC'ye bildirilir.)

END;

END; ( case )

END (if)

ELSE

swerr2('No Buffer Available', NIL);

END;

```



```

| Adı : PROCEDURE data_loader
| Tanım : Reklam anonsu verisini FP'ye yükler
| ve bu veriyi CAM kartına yazmak için
| put_data_in_cam prosedürünü çağırır.
| Parametre :
| idx (mts_idx) - IPC indeksini içerir.

```

```

PROCEDURE data_loader (idx : mts_idx);

VAR
    index_num : integer;
    block_id  : integer;

BEGIN ( data_loader )

    IF buff_get_ptr(ann_msg, idx)
    ( Bu prosedür, tampona bir mesaj gelmişse true verir.
      Gelen mesaj aşağıda görüldüğü şekilde değerlendirilir.)
    THEN
        BIND ipm TO ann_msg^.lipc IN
        ( ann_msg^.lipc'ye adının uzun olması nedeniyle, sadece
          bu prosedür de geçerli olmak üzere ipm adı verilmistir.)
        BEGIN ( if )

            com_ann_hndlr.lsw := ca_hndlr_low;
            com_ann_hndlr.msw := ca_hndlr_high;
            com_ann_hndlr.all^ := #ffff;
            ( CAM kartı donanım açısından disable duruma
              getirilir.)

            CASE ipm.intmsg.msg[4] OF
                ( Reklam anonsu numarası mesajdan alınır.)
                1 : ptr_cam.msw := cam1_high;

```



```

2 : ptr_cam.msw := cam2_high;
3 : ptr_cam.msw := cam3_high;
4 : ptr_cam.msw := cam4_high;
5 : ptr_cam.msw := cam5_high;
END;   ( case )

( Reklam anonsunun numarasına göre bu reklamın
  yazılacağı bellek bölgesinin başlangıç adresi
  belirlenir.)
block_id := ipm.intmsg.msg[5];
( Reklam anonsu verisinin blok numarası
  mesajdan alınır.)
IF block_id < 125
( Bu koşul sağlandığı sürece veri mesajının
  gönderimi tamamlanmamıştır. Bir reklam anonsuna
  ilişkin verinin tamamı 125 mesaj içinde
  gönderilmektedir.)
THEN
  BEGIN
    cam_low := cam_low_shared + (2 * block_id * 256);
    ( Her anons verisi RAM'in kendisi için ayrılmış
      olan adres bölgesine yerleştirilir.)
    FOR index_num := 0 TO 255 DO
      BEGIN ( for )
        ptr_cam.lsw := cam_low;
        ptr_cam.all := ipm.intmsg.msg[6 + index_num];
        cam_low := cam_low + 2;
      END;   ( for )
      ( CC'den gönderilen ve reklam anonsu verisini

```


içeren her bir mesajın içinde 256 byte'lık veri bulunmaktadır. Her bir byte yukarıda gösterildiği şekilde CAM kartına yazılır.)

```
send_msg_to_cc(com_flags.ca_id, dump_request, req_done,
```

```
{ CC'ye dump isteğinin yerine getirildiği bu mesajla bildirilir.}
```

```
com_flags.solic_no);
```

```
{ Yazım işlemi tamamlandıktan sonra CC'ye dump isteğinin yerine geldiğini belirten bir mesaj gönderilir.}
```

```
END      { if }
```

```
ELSE
```

```
BEGIN   { else }
```

```
bool_mate := false;
```

```
{ CC'nin isteği yerine getirilemezse bu değişkene false atanarak, işlemin eş birim için yapılması onlenir.}
```

```
swerr2('inv data',PTR(ipm.intmsg));
```

```
send_msg_to_cc(com_flags.ca_id, dump_request, req_failed, com_flags.solic_no);
```

```
END;     { else }
```

```
END      { if }
```

```
ELSE
```

```
swerr2('No Buffer Available', NIL);
```

```
END;
```

```
TASK    TYPE ann_tsk_type;
```

```
VAR
```

```
myself      : taskid;
```

```
ann_queue   : qwid;
```



```

ann_event      : event;
q_index        : mts_idx;

```

```

-----
| Adi : HANDLER ann_handler |
| Tanim : CC'den bir mesaj gelip gelmediğini |
| kontrol etmek üzere kuyruğu kontrol eder. |
| Mesaj gelmişse bu mesajın türünü saptar. |
|-----

```

```

HANDLER ann_handler ;

```

```

VAR

```

```

    count,

```

```

    events : int;

```

```

BEGIN (ann_handler)

```

```

    events := event_count(ann_handler);

```

```

    FOR count := 0 TO events DO

```

```

        BEGIN ( for )

```

```

            deqwi(ann_queue, q_index);

```

```

            IF q_index <> 0

```

```

            THEN

```

```

                BEGIN ( if )

```

```

                    IF buff_get_ptr(ann_msg, q_index)

```

```

( Bu prosedür, tampona bir mesaj gelmişse true verir.

```

```

Gelen mesaj aşağıda görüldüğü şekilde değerlendirilir. )

```

```

    THEN

```

```

        BEGIN ( if )

```

```

            IF com_flags.card_stat

```

```

            ( Aşağıdaki işlemler kartın takılı olması

```

```

            durumunda yapılacaktır. Bu bayrağın true

```


olması kartın var olduğunu gösterir.)

THEN

BEGIN

bool_mate := true;

(Bu değişkene true atanarak, aynı CC isteğinin eş birim için de yapılması sağlanır.)

CASE ann_msg7.ipc.intmsg.sendfiat OF

netas_tbl : data_loader(q_index);

netas_mtc : comm_hdlr(q_index);

OTHERWISE

swerr2('inv fiat', PTR (ann_msg7.ipc.aint_data[2]));

END; (case)

IF bool_mate

(Aktif birime gönderilen CC isteğinin yerine

getirilip getirilmediği, dolayısıyla eş

birim için de aynı işlemin yapılıp yapılmayacağı

saptanır.)

THEN

send_req_to_mate(q_index);

(Eş birime de aynı istek gönderilir .)

END (if)

ELSE

send_msg_to_cc(ann_msg7.ipc.aint_data[7], ann_msg7.ipc.aint_data [3]

cam_not_present, ann_msg7.ipc.aint_data[6]);

(CC'ye CAM kartına erişilemediği bildirilir.)

ipc_release(idx);

(Tampon bırakılır.)

END (if)

ELSE

swerr2(' No Buffer Available' NIL);

ADJ END; (if } _accept

END; (for } _accept

END; _accept

PROCEDURE ann_accept

BEGIN

WHILE true DO

ACCEPT (ann_handler)

END;

Adi : PROCEDURE ann_accept Tanım : Taskin yürütülmesinde kullanılan sabit bir yapıdır.
--

```
PROCEDURE ann_accept ;
```

```
BEGIN
```

```
  WHILE true DO
```

```
    ACCEPT (ann_handler);
```

```
END;
```



```

| Adı      : PROCEDURE init_nxcomann |
| Tanım    : NXCOMANN modülünü başlangıç durumuna getirir. |

```

```

PROCEDURE init_nxcomann ;
BEGIN ( init_nxcomann )

    initiate(ann_tvar, 750, 3);
    ( Taskın önceliği 3, yığın büyüklüğü de 750'dir.)
    com_flags.card_stat := card_presence_test;
    com_flags.card_stat := false;
    IF com_flags.card_stat
    THEN
        BEGIN ( if )
            init_cadiag;
            com_flags.req_dump := false;
            com_flags.dump_ann := false;
            com_flags.req_test := false;
            com_flags.req_query := false;
            com_flags.req_chck := false;
        END ( if )
    ELSE
        com_flags.all := 0;
    END;
    (init_nxcomann)
END. ( unit )

```


4.4.2 Diagnostik Test Modülleri

XPM Pascal Kullanılarak Hazırlanan NXCADIAG Modülü :

```

| Modul Adı : NXCADIAG kontrol eder. |
| Tanım : Bu modul CA diagnostik testlerini gerçekleştirir. |

```

```

FUNCTION chksum_CAM ; EXTERNAL ;
PROCEDURE write_test ; EXTERNAL ;
FUNCTION read_test ; EXTERNAL ;

( EXTERNAL olarak tanımlanmış olması, bu fonksiyon ve
prosedürlerin 68000 çevirici dili ile yazıldığını
ve çevirici dili ile yazılmış olan fonksiyon ve
prosedürleri içeren modül yada modüllerde olduğunu
bildirir.)

```

```

IS integer ;
1- (add : byte )
END ; ( record )

( Bir bellek gözünün içeriğini okumak
için ptr_card tanımlanmıştır.)

VAR
ptr_card_id = ptr_card ;

BEGIN ( card_exchange_test )
( CAM kartının tanıtıcı 612FFFE adresindedir.)
ptr_card_id := 612FFFE;
ptr_card_id := 612FFFE;
( ptr_card_id'nin gösterdiği bellek gözünde
kart tanıtıcı 612FFFE bulunmaktadır.)
IF (access_check(ptr_card_id, add) = act_pass)

```



```

Adi      : FUNCTION card_presence_test
Tanım    : CAM kartının takılı ve erişilebilir
           olup olmadığını kontrol eder.
Sonuç    : cam_presence_test (boolean)
           true-CAM kartı takılı ve erişilebilir
           false-CAM kart takılı değil yada
           erişim yapılamıyor.

```

```

FUNCTION card_presence_test : boolean;

```

```

TYPE

```

```

    ptr_card = PACKED RECORD

```

```

        CASE int OF

```

```

            0:(ms : integer ;

```

```

               ls : integer );

```

```

            1:(add : ^byte );

```

```

        END ; ( record )

```

```

    ( Bir bellek gözünün içeriğini okumak
      için ptr_card tanımlanmıştır.)

```

```

VAR

```

```

    ptr_card_id : ptr_card ;

```

```

BEGIN ( card_presence_test )

```

```

    ( CAM kartının tanıtıcısı $12FFFE adresindedir.)

```

```

    ptr_card_id.ms := ca_hndlr_low;

```

```

    ptr_card_id.ls := ca_hndlr_high;

```

```

    ( ptr_card_id'nin gösterdiği bellek gözünde
      kart tanıtıcısı bulunmaktadır.)

```

```

    IF (access_check(0,ptr_card_id.add) = act_pass)

```



```
( access_check fonksiyonu kart tanıtıcısının
bulundugu bellek gözüne erişim yapılip
yapılamadığını kontrol eder.)
```

```
THEN
```

```
IF (ptr_card_id.addr = #5a)
```

```
( CAM karti için kart tanıtıcısı #5A'dir. Kart
tanıtıcısının bulunduğu bellek gözüne erişim
yapılabiliyorsa bu adresin içeriği kontrol edilir.)
```

```
THEN
```

```
BEGIN ( if )
```

```
card_presence_test := true;
```

```
( Test sonucu 'true' olarak çağrılan yere
aktarılır.)
```

```
com_flags.card_state := true;
```

```
( CAM kartinin takılı ve erişilebilir olduğunu
gösteren bayraktır.)
```

```
END ( if )
```

```
ELSE
```

```
BEGIN c else g
```

```
card_presence_test := false;
```

```
( Test sonucu 'true' olarak çağrılan yere
aktarılır.)
```

```
IF com_flags.card_state
```

```
THEN
```

```
BEGIN
```

```
send_msg_to_cc(0,#ff,cam_not_present,com_flags.solic_no );
```

```
( CAM karti takılı değilse yada erişim yapılamıyorsa
durum CC'ye bildirilir.)
```

```
com_flags.card_state :=false;
```



```

( CAM kartının takılı ve erişilebilir olduğunu
  gösteren bayraktır.)
swerr2('No CAM Card',nil)
( CAM kartına erişilemediğini belirten
  swerr verir.)
      END; ( then )
END; ( else )
END; (card_presence_test)

```



```

| Adı      : PROCEDURE query_msg_to_cc |
| Tanım    : CC'ye query isteğinin yanıtını |
|           gönderir. |
| Parametreler : |
| comm_no (byte)- Reklam anonsu numarası |
| reply   (byte)- Test sonucunu içerir : |
| req-done yada req_failed'dir. |
| solicitor_no (byte)-CC'den gönderilen |
| mesajın solicitor numarasını içerir. |

```

```

PROCEDURE query_msg_to_cc (comm_no : byte;
                           reply : byte;
                           solicitor_no : byte);

```

```

VAR

```

```

    idx      : ipc_idx;

```

```

    {IPC tamponu indeksini içerir.}

```

```

    i          : integer;

```

```

    c_add      : cam_ptr_type;

```

```

BEGIN      ( query_msg_to_cc )

```

```

    IF (mod_com_area.module_activity = active)

```

```

    THEN

```

```

        BEGIN ( if )

```

```

            IF ipc_get(idx)

```

```

                ( Eğer tampon elde edilebiliyorsa
                  aşağıdaki işlemler yapılır.)

```

```

            THEN

```

```

                BIND ipcann TO ipc_bfs[idx] IN

```

```

                ( Geçici isim değişikliği yapıldı.)

```

```

                BEGIN ( if )

```

```

                    ipcann.dest := cid_cc;

```



```
ipcann.src := cid_com_ann;
```

```
IF solicitor_no = #ff
```

```
THEN
```

```
    ipcann.intmsg.msg[1] := unsol_id
```

```
    { 'unsolicitor' mesaj olduğunu belirtir. }
```

```
ELSE
```

```
    ipcann.intmsg.msg[1] := mt_gen_hand;
```

```
    { 'solicitor' mesaj olduğunu belirtir. }
```

```
    ipcann.intmsg.msg[2] := node_cons;
```

```
    ipcann.intmsg.msg[3] := solicitor_no;
```

```
    ipcann.intmsg.msg[4] := reply;
```

```
    ipcann.intmsg.msg[5] := request;
```

```
    ipcann.intmsg.msg[6] := comm_no;
```

```
CASE comm_no OF
```

```
    1 : c_add.msw := cam1_high;
```

```
    2 : c_add.msw := cam2_high;
```

```
    3 : c_add.msw := cam3_high;
```

```
    4 : c_add.msw := cam4_high;
```

```
    5 : c_add.msw := cam5_high
```

```
END; { case }
```

```
c_add.lsw := 0;
```

```
{ Reklam anonsunun adı CAM belleğinin ilk 16
```

```
byte'ında bulunmaktadır. 8 byte'lık
```

```
reklam anonsu adı, çift adreslere yazma
```

```
yapılamadığı için 16 byte'lık yer almıştır. }
```

```
FOR i := 0 TO 7 DO
```

```
    BEGIN { for }
```

```
        ipcann.intmsg.msg[i + 7] := c_add.all;
```


{ Reklam anonsunun adı mesaj içinde
bu şekilde gönderilmektedir.}

c_add.lsw := c_add.lsw + 2

{ Çift adreslerde herhangi bir veri bulunmadığı
için adres 2 arttırılmaktadır.}

END; { for }

c_add.lsw := c_add.lsw + 8;

ipcann.intmsg.msg[16] := c_add.all7;

{ Reklam anonsunun durumu mesaj içinde
bu şekilde gönderilir.}

ipcann.intmsg.int_node := my_intnode;

ipcann.intmsg.length := 25;

ipc_send(idx);

END { if }

ELSE

swerr2('NO Buffer Available', NIL);

{ Tampon elde edilemezse swerr verir.}

END; { if }

END; { query_msg_to_cc }


```

Adı      : FUNCTION csum_first_test
Tanım    : CA verisinin değişip değişmediğini
yada başarılı olarak dump edilip
edilmediğini test eder.
Parametre :
ca_id (byte)- Reklam anonsu numarası
Sonuç    :
csum_first_test (boolean)
    true-CA verisi bozulmamıştır.
    false-CA verisi bozulmuştur.

```

```
FUNCTION csum_first_test (ca_id : byte) : boolean;
```

```
BEGIN ( csum_first_test )
```

```
  CASE ca_id OF
```

```
    1 :
```

```
      BEGIN
```

```
        add_data := #130601;
```

```
        add_chk.msw := cam1_high;
```

```
      END;
```

```
    2 :
```

```
      BEGIN
```

```
        add_data := #1a0601;
```

```
        add_chk.msw := cam2_high;
```

```
      END;
```

```
    3 :
```

```
      BEGIN
```

```
        add_data := #1b0601;
```



```

118 false;
add_chk.msw := cam3_high;
END;
4 :
BEGIN
add_data := #1e0601;
(Checksum uygun değil, 4. seri ile reklam anonsu 'disable'
durumuna getirilir.)
END;
5 :
BEGIN
add_data := #1f0601;
add_chk.msw := cam5_high;
END;
END; ( case )

```

(ca_id'ye göre veri ve checksum adresleri belirlenir.)

```

add_chk.lsw := #0010;
check.low := add_chk.all;
add_chk.lsw := add_chk.lsw + 2;
check.high := add_chk.all;

```

(Daha önceden hesaplanan checksum değeri bellekten
okundu.)

```
IF chksum(add_data,16000) <> check.word
```

(16000 byte'lık veriyi toplayan assembler fonksiyonun
verdiği değerle bilinen checksum değeri karşılaştırıldı.)

THEN

BEGIN

```
csum_first_test := false;
```

(Eşit değilse fonksiyon false verir.)

CASE ca_id OF


```

1 : com_flags.first_ann := false;
2 : com_flags.second_ann := false;
3 : com_flags.third_ann := false;
4 : com_flags.fourth_ann := false;
5 : com_flags.fifth_ann := false
( Checksum uyumsuzluğu sezilen reklam anonsu 'disable'
durumuna getirilir.)
END; ( case )
com_flags.chng_ann := true;
( Bir reklam anonsunun durumunda değişiklik yapıldığı
bu bayrağa 'true' atanarak belirtilir.)
query_msg_to_cc(ca_id, req_failed, com_flags.solic_no)
( CC'ye test sonucu olumsuz olarak bildirilir.)
END
BEGIN ( csum_second_test )
ELSE
add_chk.lsw := add_chk.lsw + 2;
BEGIN
check.lsw := add_chk.all;
csum_first_test := true;
add_chk.lsw := add_chk.lsw + 2;
( Esitlerse fonksiyon true verir.)
check.high := add_chk.all;
query_msg_to_cc(ca_id, req_done, com_flags.solic_no)
( CC'ye test sonucu olumlu olarak bildirilir.)
IF check.data > 15000 THEN check.word
END;
( csum_first_test'de yapılan işlem, 2, 15000 byte ile
veritabanı tekrarlanır.)
THEN
BEGIN ( csum_first_test )
csum_second_test := false;
( Esit değilse fonksiyon false verir.)
CASE IS OF
1 : com_flags.first_ann := false;

```



```

Adı      : FUNCTION csum_second_test
Tanım    : CA verisinin değişip değişmediğini
yada başarılı olarak dump edilip
edilmediğini test eder.
Parametre :
ca_id (byte)- Reklam anonsu numarası
Sonuç     :
csum_second_test (boolean)
    true-CA verisi bozulmamıştır.
    false-CA verisi bozulmuştur.

```

```

FUNCTION csum_second_test (ca_id : byte) : boolean;
BEGIN ( csum_second_test )
    add_chk.lsw := add_chk.lsw + 2;
    check.low := add_chk.all;
    add_chk.lsw := add_chk.lsw + 2;
    check.high := add_chk.all;
    add_data := add_data + 32000;

    IF chksum(add_data,16000) <> check.word
    ( csum_first_test'de yapılan işlem, 2. 16000 byte'lık
      veri için tekrarlanır.)
    THEN
        BEGIN ( if )
            csum_second_test := false;
            ( Eşit değilse fonksiyon false verir.)
            CASE ca_id OF
                1 : com_flags.first_ann := false;

```



```

121
2 : com_flags.second_ann := false;
3 : com_flags.third_ann := false;
4 : com_flags.fourth_ann := false;
5 : com_flags.fifth_ann := false;
END; ( case )

( Checksum uyusmazlığı sezilen reklam anonsu 'disable'
durumuna getirilir.)
com_flags.chng_ann := true;
( Bir reklam anonsunun durumunda değişiklik yapıldığı
bu bayrağa 'true' atanarak belirtilir.)
query_msg_to_cc(ca_id, req_failed, com_flags.solic_no)
( CC'ye test sonucu olumsuz olarak bildirilir.)
END ( if )
ELSE
BEGIN ( else )
csum_second_test := true;
( Eşitlerse fonksiyon true verir.)
query_msg_to_cc(ca_id, req_done, com_flags.solic_no) .
( CC'ye test sonucu olumlu olarak bildirilir.)
END; ( else )
END; ( csum_second_test )

```



```

|
|  Adi      : PROCEDURE audit_checksum_test
|
|  Tanım    : CA datasında bir bozulma olup olmadığını
|
|  belirli zaman aralıkları ile kontrol eder.
|
|

```

```
PROCEDURE audit_checksum_test ;
```

```
VAR
```

```
st : boolean ;
```

```
( audit_checksum_test'i sadece 'enable' olan reklam
  anonsları üzerinde yapılacağı için bu değişken
  tanımlanmıştır.)
```

```
rep : integer ;
```

```
( Durumu 'enable' olan reklam anonsu olmadığı
  takdirde audit_checksum_test fonksiyonundan
  çıkılmalıdır. Bu kontrolün yapılabilmesi için
  rep değişkeni tanımlanmıştır.)
```

```
BEGIN ( audit_checksum_test )
```

```
IF card_presence_test
```

```
( CAM kartı takılı ise ve erişilebiliyorsa
  bu test yapılır.)
```

```
THEN
```

```
BEGIN ( if )
```

```
IF count = 1
```

```
( Bu değişken, 1'e eşit olduğu zaman toplama
  işlemi yeni başlayacaktır. Bazı değişkenlere
  ilk değer atamaları yapılır.)
```

```
ca_iden := ca_iden + 1;
```

```
IF ca_iden > 5
```

```
THEN
```



```

        ca_iden := 1;
THEN
    BEGIN {count =1}
        sum := 0;
        { Toplam degeri bu degiskene atanir.}
        rep := 0;
        { Beş reklamın 'enable' olup olmadıklarını kontrol
          ettikten sonra, 'enable' durumda olan reklam
          yoksa işlemi sonlandırması için rep tanımlandı..}
    REPEAT
        rep := rep + 1;
        { Sırayla 'enable' durumda olan tüm reklam
          anonsları test edildikten sonra, test ilk
          'enable' durumdaki reklam anonsundan başlanarak
          tekrarlanır.}
    CASE ca_iden OF
        1 : st := com_flags.first_ann;
        2 : st := com_flags.second_ann;
        3 : st := com_flags.third_ann;
        4 : st := com_flags.fourth_ann;
        5 : st := com_flags.fifth_ann
    END; { case }
    ca_iden := ca_iden + 1;
    IF ca_iden > 5
    THEN
        ca_iden := 1;
        IF rep > 5
        THEN
            exit(audit_checksum_test);

```


UNTIL st;

CASE ca_iden OF

1 : BEGIN

au_chk.msw := cam1_high;

au_data := #130601

END;

2 : BEGIN

au_chk.msw := cam2_high;

au_data := #1a0601

END;

3 : BEGIN

au_chk.msw := cam3_high;

au_data := #1b0601

END;

4 : BEGIN

au_chk.msw := cam4_high;

au_data := #1e0601

END;

5 : BEGIN

au_chk.msw := cam5_high;

au_data := #1f0601

END

END; (case)

END; (count = 1)

IF (count = 81) OR (count = 161)

check.low := au_chk.lsw;

au_chk.lsw := au_chk.lsw + 2;

check.high := au_chk.msw;

{ Bilinen checksum değeri karttan okundu.}

au_chk.lsw := au_chk.lsw + 2;

{ count'in 81 olması halinde 161'de checksum'in belirlenmesine hazırlık olması açısından bu atama yapılmıştır.}

{ count'un 81 olması, ilk 16000 byte'in 161 olması ise son 16000 byte'in işleminin bittiğini gösterir.}

THEN

BEGIN { if }

IF (check.word <> sum)

{ İlk 16000 byte'in toplamı, bilinen checksum değeri ile karşılaştırılır.}

THEN

BEGIN { if }

query_msg_to_cc(ca_iden, req_failed, &ff);

{ Toplam, bilinen checksum değerine eşit değilse durum CC'ye bildirilir.}

count := 1;

{ Sayaç başlangıç durumuna getirilir.}

{ Bu fonksiyonun bir sonraki çağrılışında bir sonraki reklam anonsu için test yapılır.}

CASE ca_iden OF

1 : com_flags.first_ann := false;

2 : com_flags.second_ann := false;

3 : com_flags.third_ann := false;

4 : com_flags.fourth_ann := false;

5 : com_flags.fifth_ann := false


```

END; ( case )

( Checksum uyumsuzluğu sezilen reklam
  anonsu 'disable' durumuna getirilir.)

count := 81;
com_flags.chng_ann := true;

( Bir reklam anonsunun durumunda değişiklik
  yapıldığı bu bayrağa 'true' atanarak
  belirtilir.)

exit(audit_checksum_test);

( audit_checksum_test'inden cikilir.)

END ( if )

ELSE

BEGIN

  IF count = 81 THEN sum := 0;

  IF count = 161

  THEN

    BEGIN

      query_msg_to_cc(ca_iden, req_done, #ff) ;

      ( Toplam bilinen checksum değerine eşitse

        CC'ye test sonucu olumlu olarak bildirilir.)

      count := 1 ;

      exit(audit_checksum_test);

      ( audit_checksum_test'inden cikilir.)

    END; c if g

  END; ( count = 81 yada 161)

au_data := au_data + 400;

( Fonksiyonun her çağrılışında 200 byte'lık
  veri toplandığı ve çift adreslerde veri
  bulunmadığı için, verinin alındığı

```


adres, 400 arttırılır.)

Adt sum := sum + chksum(au_data,200);

Top (Elde edilen toplam bir önceki toplama eklenir.)

Top count := count + 1; (ok bölgesine yazar.)

Par (Sayac arttırılır.)

END (cam card check)

END; (audit_checksum_test)

PROCEDURE first_write_test (ca_id integer);

BEGIN (first_write_test)

IF card_presence_test

THEN

BEGIN (if)

(Bu test CAM kartı takılı ve erişilebilir durumda ise yapılır.)

CASE ca_id OF

0 : start := 010000;

1 : start := 010000;

2 : start := 010000;

3 : start := 010000;

4 : start := 011000;

END; (if)

(Başlangıç adresi test edilerek, bir önceki testin başlangıç adresi olarak belirlenir.)

write_checksum_test;

(start başlangıç adresinden sonraki 200 byte için okuma yapılır ve checksum test yapılır.)

END; (if)

END; (first_write_test)


```

Adı      : PROCEDURE first_write_test
Tanım    : Test örneği #AA'yi bir reklam anonsu
          için ayrılmış olan bellek bölgesine yazar.
Parametreler:
ca_id (INTEGER)- Reklam anonsu numarası

```

```
PROCEDURE first_write_test (ca_id : integer);
```

```
BEGIN ( first_write_test )
```

```
IF card_presence_test
```

```
THEN
```

```
  BEGIN ( if )
```

```
    ( Bu test CAM kartı takılı ve erişilebilir
      durumda ise yapılır.)
```

```
    CASE ca_id OF
```

```
      0 : start := #130000;
```

```
      1 : start := #1a0000;
```

```
      2 : start := #1b0000;
```

```
      3 : start := #1e0000;
```

```
      4 : start := #1f0000
```

```
    END; ( case )
```

```
    ( Başlangıç adresi test edilecek olan reklam anonsu
      bölgesine bağlı olarak belirlenir.)
```

```
    write_test(start,#aa);
```

```
    ( start başlangıç adresinden itibaren 32K byte'lık
      bölgeye #AA test örneği yazılır.)
```

```
  END; ( if )
```

```
END; (first_write_test)
```



```

Adi      : FUNCTION first_read_test
Tanım    : Bir reklam anonsu için ayrılmış olan
           bellek bölgesindeki tüm gözlerin #AA olup
           olmadığını kontrol eder.
Sonuç    :
first_read_test (boolean)
           true-Bu bellek bölgesi doğru olarak
              çalışıyor demektir.
           false-Bu bellek bölgesi doğru olarak
              çalışmıyor demektir.

```

```

FUNCTION first_read_test : boolean;
BEGIN ( first_read_test )
  IF card_presence_test
  ( Bu test CAM kartı takılı ve erişilebilir durumda ise
  yapılır.)
  THEN
    BEGIN ( if )
      IF read_test(start,#aa) = 1
      THEN
        first_read_test := true
        ( start adresinden başlanılarak 32K byte'lık
          bellek bölgesinin içeriğinin #AA'ya eşit
          olup olmadığı kontrol edilir. Eğer tüm bellek
          gözlerinin içeriği #AA ise read_test fonksiyonu
          sonucu 1'dir.)
      ELSE
        first_read_test := false;

```


END; (if)

END; (first_read_test). write_test

Tanın (test) orneği (SAA'y) bir reklam sunuğu

için ayarlanmış olan diğer bölgesine geçer.

PROCEDURE second_write_test

BEGIN (second_write_test)

IF card_presence_test

THEN

BEGIN (if)

write_test(start, 100);

END; (if)

END; (second_write_test)


```

| Adı      : PROCEDURE second_write_test.
| Tanım    : Test örneği #AA'yi bir reklam anonsu
| için ayrılmış olan bellek bölgesine yazar.
|

```

```
PROCEDURE second_write_test ;
```

```
BEGIN ( second_write_test )
```

```
  IF card_presence_test
```

```
  THEN
```

```
    BEGIN ( if )
```

```
      write_test(start,#55);
```

```
    END; ( if )
```

```
END; (second_write_test)
```

```
BEGIN ( second_read_test )
```

```
  IF card_presence_test
```

```
  THEN
```

```
    BEGIN ( if )
```

```
      Bu test CAN kartı takılı ve çalışabilir
```

```
      durumda ise yapılır.
```

```
      IF read_test(start,#55) = 1
```

```
      THEN
```

```
        second_read_test ;
```

```
        Bu test adresinden başlanılarak 32K byte'lık
```

```
        bellek bölgesinin içeriğinin CAN'ya yazıl
```

```
        tır ve kontrol edilir. Eğer 1'den büyük
```

```
        bir sonuç dönerse CAN ile read test fonksiyonu
```

```
        çalışmaz.
```

```
        ELSE
```

```
          Bu test başarısızdır.
```



```

| END
| Adı      : FUNCTION second_read_test
| Tanım    : Bir reklam anonsu için ayrılmış olan
| bellek bölgesindeki tüm gözlerin #AA olup
| olmadığı kontrol eder.
| Sonuç    :
| first_read_test (boolean)
|   true-Bu bellek bölgesi doğru olarak
|         çalışıyor demektir.
|   false-Bu bellek bölgesi doğru olarak
|         çalışmıyor demektir..

```

```

FUNCTION second_read_test : boolean;
BEGIN ( second_read_test )
  IF card_presence_test
  THEN
    BEGIN ( if )
      ( Bu test CAM kartı takılı ve erişilebilir
        durumda ise yapılır.)
        IF read_test(start,#55) = 1
        THEN
          second_read_test := true
          ( start adresinden başlanılarak 32K byte'lık
            bellek bölgesinin içeriğinin #AA'ya esit
            olup olmadığı kontrol edilir. Eğer tüm bellek
            gözlerinin içeriği #AA ise read_test fonksiyonu
            sonucu 1'dir.)
        ELSE
          second_read_test := false;

```


END; (if)

END; (second_read_test)

PROCEDURE init_kasas;

VAR

 i : integer;

 result : boolean;

 st_add : boolean;

BEGIN (init_kasas)

 fst_write := false;

 fst_read := false;

 snd_write := false;

 snd_read := false;

 -- Initialize the state of the system

 -- Initialize the state of the system

 check := 0;

 check := 0;

 -- Check the state of the system

 -- Check the state of the system

 st_add := 0;

 st_add := 0;

 -- Initialize the state of the system

 st_add := 0;

 st_add := 0;

 st_add := 0;

 -- Initialize the state of the system

END;


```

|
| Adı : PROCEDURE init_cadiag
|
| Tanım : NXCADIAG modülünü başlangıç
| durumuna getirir.
|

```

```

PROCEDURE init_cadiag ;
VAR
    i : integer ;
    result : boolean ;
    st_add : cam_ptr_type;
BEGIN ( init_nxcadiag )
    fst_write := false;
    fst_read := false;
    snd_write := false;
    snd_read := false;
    ( Bellek testinde kullanılacak olan bu
    değişkenlere ilk değerleri atanır.)
    check.low := 0;
    check.high := 0;
    ( Checksum değerinin okunacağı
    adreslere ilk değerleri atanır.)
    ptr_start.lsw := 0;
    ptr_start.msw := 0;
    ( Bellek testlerinde kullanılırlar. )
    au_data := 0;
    au_chk.lsw := 0;
    au_chk.msw := 0;
    ( Audit checksum testlerinde kullanılırlar. )
    count := 1;

```



```

ca_iden := 0;
result := false;
FOR i := 1 TO 5 DO
  ( Beş reklam anonsu için checksum testi yapılarak
    bayraklarına durumlarını belirten değerler atanır.)
  BEGIN ( for )
    CASE ca_iden OF
      1 : st_add.msw := cam1_high;
      2 : st_add.msw := cam2_high;
      3 : st_add.msw := cam3_high;
      4 : st_add.msw := cam4_high;
      5 : st_add.msw := cam5_high
    END; ( case )
    st_add.lsw := 26;
    ( Reklam anonsunun durumunun CAM kartından
      okunabilmesi için durumun yazılı olduğu bellek
      adresi belirlendi.)
    IF st_add.all? = 1
      ( Bu test enable durumda olan reklamlar için
        yapılır. )
    THEN
      BEGIN ( if st? )
        IF csum_first_test(i)
        THEN
          BEGIN ( first test )
            IF csum_second_test(i)
            THEN
              BEGIN ( second test )

```



```

    { csum_first_test ve csum_second_test
      fonksiyonları olumlu sonuç verdiği
      takdirde aşağıdaki işlemler yapılır.}
    query_msg_to_cc(i, req_done, #ff);
    result := true
  END ( second test )
ELSE
  { csum_first_test fonksiyonu olumlu,
    csum_second_test fonksiyonu olumsuz
    sonuç verirse aşağıdaki işlemler
    yapılır.}
  query_msg_to_cc(i, req_failed, #ff);
  END ( first test )
ELSE
  { csum_first_test fonksiyonu olumsuz sonuç verirse
    csum_second_test fonksiyonu çağırılmadan CC'ye
    mesaj gönderilir.}
  query_msg_to_cc(i, req_failed, #ff);
IF result
THEN
  CASE i OF
    0 : com_flags.first_ann := true;
    1 : com_flags.second_ann := true;
    2 : com_flags.third_ann := true;
    3 : com_flags.fourth_ann := true;
    4 : com_flags.fifth_ann := true
  END ( case )
  { Checksum testinden geçen reklam anonsu
    enable duruma getirilir.}

```



```
com_flags.chng_ann := true;
```

```
{ Bir reklam anonsunun durumunda deęişiklik yapıldığı  
  bu bayrağa 'true' atanarak belirtilir. }
```

```
END ( if st )
```

```
END; { for }
```

```
END; { init_cadiag }
```

68000 Assembler Kullanılarak Hazırlanan ASMNXCAD Modulu :

```
-----;  
;  
; Modül Adı : ASMNXCAD  
;  
; Tanım : Bu modül, XPM PASCAL kullanılarak hazırlanmış  
;  
; olan CA diagnostik testlere hız kazandırmak amacı ile  
;  
; 68000 çevirici dili kullanılarak hazırlanan prosedür  
;  
; ve fonksiyonları içerir.  
;  
-----;
```

```
-----;  
;  
; FUNCTION CHKSUM ( ADD_DATA : LONGINT ;  
;                   NUM       : INTEGER ) : LONGINT ;  
;  
; TANIM : 'NUM' sayıda byte'i toplayıp sonucu çağrıldığı  
;        yere iletir.  
;  
; KULLANILMAYAN SAKLAYICILAR : D2-D7, A1-A5  
;  
-----;
```

```
.FUNC CHKSUM,3 ; 2 parametresi ve dönüş adresi  
               ; nedeni ile fonksiyon isminden sonra  
               ; 3 yazılmıştır.
```

```
MOVEA.L (A7)+,A6 ; A6 <--- Dönüş Adresi  
               ; Dönüş adresini yığından alarak  
               ; A6 adres saklayıcısına atar.
```


; İşlemler tamamlandıktan sonra
 ; A6 adres saklayıcısının içeriğini
 ; oluşturan adrese dallandığında
 ; ana programda kaldığı yere dönmüş
 ; olur.

CLR.L D0 ; D0 <--- 0
 MOVE.W (A7)+,D2 ; Anlamsız byte'ları al
 MOVE.W (A7)+,D2 ; Anlamsız byte'ları al
 MOVE.W (A7)+,D0 ; D0 <--- Toplanacak byte sayısı
 ; Toplanacak olan byte sayısını
 ; yığından alarak D0 veri saklayıcısına
 ; atar. D0 veri saklayıcısı, chksum
 ; fonksiyonunda bir sayaç olarak
 ; kullanılacaktır.

MOVEA.L (A7)+,A0 ; A0 <--- add_data
 ; Toplanacak olan verilerin başlangıç
 ; adresini yığından alarak A0 adres
 ; saklayıcısına atar.

SUBI #1,D0 ; D0 <--- D0 - 1
 CLR.L D3 ; D3 <--- 0
 CLR.L D1 ; D1 <--- 0
 ; D1 veri saklayıcısına 0 atar.

REP MOVE.B (A0),D3 ; D3 <--- (A0)
 ADD.W D3,D1 ; D1 <--- D1 + (A0)
 ADDA #2 ,A0 ; A0 <--- A0 + 2
 ; A0 adres saklayıcısının içeriği
 ; 2 arttırılır.

DBF D0,REP ; D0 <--- D0 - 1

; Eğer D0 (> 0 ise REP'e dallanır.

; Sayaç olarak kullanılan D0 veri

; saklayıcısının içeriği 0'a esit

; olduğunda 'num' sayıda byte'ın

; toplama işlemi tamamlanmış demektir.

; 0'a eşit değilse toplama işlemine

; devam edilir.

MOVE.L D1,-(A7) ; Toplamı yığına atar.

JMP (A6) ; Çağrıldığı yere geri döner.

MOVE.L (A7)+,A6 ; -- Dönüş Adresi

MOVE.B (A7)+,D1 ; D1 --- Test Örneği: 1000 yada 2551

MOVE.L (A7)+,A0 ; A0 --- 'Start' Adresi

MOVE.B D0,D0 ; D0 --- 00000

AAA MOVE.B D1,(A7) ; (A7) 1--- Test Örneği

AAA ADDA A2,A0 ; A0 --- A0 + 2

AAA ADDA D0,A0 ; D0 1--- D0 + 1


```

;-----;
;
; PROCEDURE WRITE_TEST( START      : LONGINT ;
;                        TEST_PATTERN: INTEGER ) ;
;
; TANIM : Test örneğini 'start'dan başlayarak 32K'lık
;        bellek bölgesine yazar.
;
; KULLANILMAYAN SAKLAYICILAR : D2-D7, A1-A5
;-----;

```

.PROC WRITETEST,3; 2 parametresi ve dönüş adresi

; nedeni ile prosedür isminden sonra

; 3 yazılmıştır.

MOVEA.L (A7)+,A6 ; A6 <--- Dönüş Adresi

; Dönüş adresini yığından alarak

; A6 adres saklayıcısına atar.

MOVE.W (A7)+,D1 ; D1 <--- Test Örneği (#AA yada #55)

; #AA yada #55 olan test örneğini yığından

; alarak D1 yeri saklayıcısına atar.

MOVEA.L (A7)+,A0 ; A0 <--- 'Start' Adresi

; Test örneğinin yazılacağı bölgenin

; başlangıç adresini yığından alarak

; A0 adres saklayıcısına atar.

MOVE #7FFF,D0 ; D0 <--- #8000

; D0 veri saklayıcısına #8000 yani

; 32768 (32K) atar.

AAA MOVE.B D1,(A0) ; (A0) <--- Test Örneği

; Test örneğini A0 adres saklayıcısında

; adresi yazılı olan bellek gözüne atar.

ADDA #2,A0 ; A0 <--- A0 + 2

; A0 adres saklayıcısının içeriği

; 2 arttırılır.

DBF D0,AAA ; D0 <--- D0 - 1


```

; Eğer D0 (>) 0 ise AAA'ya dallanır.
FUNCTION READ
; D0 veri saklayıcısı bir sayac
; olarak kullanılır. 0'a eşit değilse
; atama işlemine devam edilir.
JMP(A6) ; Çağrıldığı yere geri döner.

```

```

FUNC READTEST,3 ; 3 parametreli bir fonksiyon
; ne kadar ile fonksiyon ile döner
; 3 veri saklayıcı
MOVE.L (A7)+,A6 ; A6'ya fonksiyon adresi
; Döner adresini 7-8 video olarak
; A6 adres saklayıcısına atar
MOVE.L (A7)+,D3 ; D3'e test byte'ları gelir
MOVE.W (A7)+,D1 ; D1'e test gruba 1600 yada 1555
; 1600 yada 1555 olan test grubu ile fonksiyon
; olarak 01 veri saklayıcısına atar
MOVE.L (A7)+,A6 ; A6'ya fonksiyon adresi
; Test grubu ile ilgili karşılaştırılacak
; olan değerin başlangıç adresini
; videodan atar A6 adres saklayıcısına
; atar
MOVE.W (A7)+,D0 ; D0'e test grubu
; D0 veri saklayıcısına 1555 atar
CLR.L D2 ; D2'e 0
; D2 veri saklayıcısına 0 atar
RMB 1 (A6),D1 ; Test grubu, adresi A6'da olan bütün
; 1600 yada 1555 ile karşılaştırılır
ADD.L #1,A6 ; A6'ya A6 + 1

```



```

;-----;
; FUNCTION READ_TEST( START      : LONGINT ;
;                      TEST_PATTERN: INTEGER ) : INTEGER ;
;
; TANIM : Test örneğini 'start' adresinden başlayarak
;         32K'lık bellek bölgesinin içeriği ile
;         karşılaştırma.
;
; KULLANILMAYAN SAKLAYICILAR : D3-D7, A1-A5
;-----;

```

.FUNC READTEST,3 ; 2 parametresi ve dönüş adresi

; nedeni ile fonksiyon isminden sonra

; 3 yazılmıştır.

MOVEA.L (A7)+,A6 ; A6 <--- Dönüş Adresi

; Dönüş adresini yığından alarak

; A6 adres saklayıcısına atar.

MOVE.L (A7)+,D3 ; Önemsiz byte'ları çeker.

MOVE.W (A7)+,D1 ; D1 <--- Test Örneği (#AA yada #55)

; #AA yada #55 olan test örneğini yığından

; alarak D1 veri saklayıcısına atar.

MOVEA.L (A7)+,A0 ; A0 <--- 'Start' Adresi

; Test örneği ile içeriği karşılaştırılacak

; olan bölgenin başlangıç adresini

; yığından alarak A0 adres saklayıcısına

; atar.

MOVE #7FFF,D0 ; D0 <--- #7FFF

; D0 veri saklayıcısına 32767 atar.

CLR.L D2 ; D2 <--- 0

; D2 veri saklayıcısına 0 atar.

BBB CMP.B (A0),D1 ; Test örneği, adresi A0'da olan bellek

; gözünün içeriği ile karşılaştırılır.

ADDA #2,A0 ; A0 <--- A0 + 2


```

; A0 adres saklayıcısının içeriği
; 2 arttırılır.

BNE CNT      ; Adresi A0' da bulunan bellek gözünün
; içeriği D1 veri saklayıcısının
; içeriğinden farklı ise, CNT'a daller.

DBF D0, BBB  ; D0 <--- D0 - 1
; Eger D0 <> 0 ise BBB'ye daller.
; D0 veri saklayıcısı bir sayaç
; olarak kullanılır. 0'a eşit değilse
; karşılaştırma işlemine devam edilir.

MOVE #1,D2   ; D2 <--- 1
; D2 veri saklayıcısına 1 atar.

CNT CMPI #1,D2 ; D2 veri saklayıcısının içeriğinin
; 1 olup olmadığını kontrol eder.

BNE CCC      ; D2 veri saklayıcısının içeriği 1'e
; eşitse CCC'ye daller, değilse
; devam eder.

BRA EEE      ; Kosulsuz olarak EEE'ye daller.
; çağrıldığı yere geri döner.

CCC MOVE #0,-(A7) ; Sonucu 0 olarak yığına atar.
BRA DDD      ; Kosulsuz olarak DDD'ye daller.
; çağrıldığı yere geri döner.

EEE MOVE #1,-(A7) ; Sonucu 1 olarak yığına atar.

DDD JMP(A6)    ; Çağrıldığı yere geri döner.

.END          ; Çevirici dilde hazırlanan modülün
; sonunu belirtir.

```


5.0 KISALTMALAR

PC	Personal Computer
NETAŞ	Northern Electric Telekomünikasyon A.Ş.
PTT	Posta Telefon Telgraf
CCITT	The International Consultative Committee for Telephones and Telegraphs
DMS	Digital Multiplex System
PCM	Pulse Code Modulation
PM	Peripheral Module
XPM	Extended Peripheral Module
CA	Commercial Announcement
CAM	Commercial Announcement Memory
CPU	Central Processing Unit
CCC	Central Control Complex
CMC	Central Message Controller
PS	Program Store
DS	Data Store
NM	Network Module
NMC	Network Message Controller
XMS	Extended Multiprocessor System
IPC	Inter-Process Register
RTSS	Run Time Support System
FP	Facilities Processor
MP	Master Processor
SP	Signalling Processor
LGC	Line Group Controller
DTC	Digital Trunk Controller
VDU	Video Display Unit
IOC	Input/Output Controller

ILGC International Line Group Controller

PC Personal Computer

1. Richard M. Jones, *Introducing to Pascal and Computer Applications*, Allen and Unwin Inc., 1983
2. Martin L. Shocart, *Software Engineering*, McGraw-Hill Book Co., 1987
3. Allen B. Tucker, *Programming Language*, McGraw-Hill Book Co., 1985
4. Arthur W. Keller, *A First Course in Computer Programming Using Pascal*, McGraw-Hill Book Co., 1985
5. Halden Cooper, *Terminology of Pascal*, Krieger Books, 1985
6. IBM, *IBM-100 System Description*, 1985
7. H. Bruce Carlson, *Communication Systems*, McGraw-Hill Book Co., 1985
8. Daniel, Berthiaume & Robert Collier, *Data Structures*, Prentice-Hall Inc., 1987
9. Walter A. Jirshel & Peter Simon, *The 68000 Microprocessor: Architecture, Software and Interfacing Techniques*, Prentice-Hall, 1986
10. Motorola Semiconductor, *Motorola 68000 16/32-Bit Microprocessor*, Motorola Reference Manual, 1984
11. IBM User Documentation
 1. IBM (Cell) Northern Research Computing Technology
 - 2.1. IBM 68000 Reference Manual, 1981
 - 2.2. IBM 68000 Assembly Reference Manual, 1982
 - 2.3. IBM User Interface, 1982
 - 2.4. IBM Software Development System, 1981
 - 2.5. IBM User Interface, 1981
 - 2.6. Getting Started with IBM, 1981
 - 2.7. IBM User Interface, 1981
 - 2.8. IBM User Interface, 1981

6.0 KAYNAKLAR

1. Richard M. Jones, Introduction to Pascal and Computer Applications, Allyn and Bacon Inc., 1983
Martin L. Shooman, Software Engineering, McGraw-Hill Book Comp., 1987
2. Allen B. Tucker, Programming Languages, McGraw-Hill Book Comp., 1985
3. Arthur M. Keller, A First Course In Computer Programming Using Pascal, McGraw-Hill Book Comp., 1985
4. Haldun Akpınar, Turbo ve UCSD Pascal, Kipas Dağıtımçılık, 1985
5. BNR, DMS-100 System Description, 1985
6. A. Bruce Carlson, Communication Systems, McGraw-Hill Book Comp., 1985
7. Dimitri Bertsekas & Robert Gallager, Data Networks, Prentice-Hall Inc., 1987
8. Walter A. Triebel & Avtar Singh, The 68000 Microprocessor : Architecture, Software And Interfacing Techniques, Prentice-Hall, 1986
8. Motorola Semiconductors, M68000 16/32-Bit Microprocessor Programmer's Reference Manual, 1984
9. XMS User Documentation :
BNR (Bell-Northern Research) Computing Technology
 - 9.1. BNR PASCAL Reference Manual, 1987
 - 9.2. 68000 Assembler Reference Manual, 1988
 - 9.3. XMS User Interface, 1988
 - 9.4. The Software Development System, 1987
 - 9.5. Design Documents, 1985
 - 9.6. Getting Started with XMS, 1987
 - 9.7. Analitical tools, 1987
 - 9.8. Programing Techniques, 1986

10. Vince Coughlin, Telecommunications Equipment
Fundamentals and Network, Van Nostrand Reinhold Comp., 1984



ÖZGEÇMİŞ

Esin Bingöl, 1967'de İstanbul'da doğdu. 1983 yılında Pendik Lisesinden, 1987 yılında İstanbul Teknik Üniversitesi Elektrik Elektronik Fakültesi, Elektronik ve Haberleşme Bölümünden mezun oldu. Aynı yıl Yıldız Üniversitesi Bilgisayar Bilimleri Mühendisliği Bölümünde yüksek lisans eğitimine başlayan Esin Bingöl, Mayıs 1989'dan bu yana NETAŞ Araştırma ve Geliştirme Bölümünde DMS Yazılım Tasarımcısı olarak çalışmaktadır.

