İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

PİKO UYDU İÇİN İŞLETİM SİSTEMİNE GÖREV ENTEGRASYONU

YÜKSEK LİSANS TEZİ

Müh. Burak SAĞLAM

Anabilim Dalı : UÇAK VE UZAY MÜHENDİSLİĞİ

Programı : DİSİPLİNLER ARASI PROGRAM

HAZİRAN 2008

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

PİKO UYDU İÇİN İŞLETİM SİSTEMİNE GÖREV ENTEGRASYONU

YÜKSEK LİSANS TEZİ
Müh. BURAK SAĞLAM
511041011

Tezin Enstitüye Verildiği Tarih : 5 Haziran 2008

Tezin Savunulduğu Tarih : 9 Haziran 2008

Tez Danışmanı : Yard. Doç. Dr. Turgut Berat KARYOT

Diğer Jüri Üyeleri Doç. Dr. Selçuk PAKER

Yard. Doç. Dr. Cuma YARIM

HAZİRAN 2008

ÖNSÖZ

Günümüzde uzay teknolojileri hızla ilerlemektedir. Bu sürecin ilk başlangıcındaki Sputnik uydusundan bu zamana uydular, gelişen teknolojiyi içlerinde barındırmış ve günlük yaşam işlerinde veya özel amaçlı kullanımlarda daha da önem kazanmışlardır. Böylesine önemli bir alanda çalışma konusu, değerli hocam Yard. Doç. Dr. Turgut Berat Karyot' un ve benim ilgimizi çekmiştir. Uyduların bilgisayarlarına görev entegrasyonu yapılırken kullanılan yazılım yapıları çok önemlidir. Bu nedenle bu tez ve benzeri konular uzay mühendisliği açısından önemli ve gereklidir.

Bu çalışmada bana her zaman destek veren değerli dostum elektrik-elektronik mühendisi Hakan Aydar' a ve bana karşı hep sabırlı olan ve engin bilgisi ile ışık tutan değerli hocam Yard. Doç. Dr. Turgut Berat Karyot' a teşekkürü bir borç bilirim.

Mayıs 2008 Burak SAĞLAM

İÇİNDEKİLER

KISATMALAR

GZİS	: Gerçek Zamanlı İşletim Sistemi.
ISR	: Interrupt Service Routine
TCB	: Task Control Block
I²C	: Inter-Integrated Circuit
SPI	: Serial Peripheral Interface
RS 232	: Recommended Standard 232
SD	: Secure Digital

TABLO LİSTESİ

Sayfa No

ŞEKİL LİSTESİ

Sayfa No

PİKO UYDU İÇİN İŞLETİM SİSTEMİNE GÖREV ENTEGRASYONU

ÖZET

Bu tez çalışmasında, son yıllarda üniversitelerin üzerinde durduğu uydu yapımı konusunun en önemli bölümlerinden biri olan, ana bilgisayar içinde koşturulacak gerçek zamanlı işletim sistemi (GZİS) seçimi ve seçilen işletim sistemine uydunun görevlerinin entegre edilmesi amaçlanmıştır.

İ.T.Ü Uçak ve Uzay Bilimleri Fakültesi bünyesinde gerçekleştirilmekte olan "ITU-pSAT-I" projesi temel alınarak GZİS seçilmiş ve yine bu piko uydunun görev yükleri göz önünde tutularak GZİS' ne görev entegrasyonu yapılmıştır. Piko uydunun görev yükleri başlıca 3 tanedir. Birincisi düşük çözünürlüklü kameradır. İkincisi güç sistemi, sensörler ve pasif manyetik dengeleyiciyi barındıran yapıdır. Üçüncü sistem ise dünyadaki yer istasyonu ile irtibat sağlayan modem sistemi olarak kabul edilebilir.

Uydu içinde yukarıda belirtilen görev yükleri ana bilgisayar ile 2 çeşit veri yolu kullanarak haberleşmektedir. Bu veri yolları I^2C ve RS232 dir. RS232 veri yolu, ana bilgisayar ile modem arasında kullanılırken, I^2C veri yolu, ana bilgisayar ile kamera sistemi ve ikinci görev yükü arasında kullanılmaktadır.

Yukarıda kısaca özetlenen piko uydunun ana bilgisayarında kullanılacak olan GZİS olarak SALVO işletim sistemi seçilmiştir. İşletim sistemi oluşturulurken üç temel iş için ayrı ayrı üç görev oluşturulmuştur. En yüksek öncelik durumu dünya ile haberleşmeye atanmıştır. Daha sonra kamera işlemleri ve güç sistemi ile sensör sisteminin ana bilgisayar ile etkileşim görevleri oluşturulmuştur.

İşletim sistemi doğrudan uydunun üzerinde koşacak şekilde hazırlanmıştır. Ancak tezin hazırlanması sürecinde, gerçek donanımların hepsi tümleşik halde bulunmamaktadır. Bundan ötürü görev yükleri, bilgisayar ve mikro işlemciler ile benzetim yapılmıştır. Uydu ana bilgisayarı ile tezde kullanılan mikro işlemci birebir aynıdır. Aynı şekilde veri yolu yapılarına da sadık kalınmıştır.

Yapılan çalışmalarda ana bilgisayar içinde koşturulan işletim sisteminin ve benzetim elemanlarının görevlerini başarıyla yaptığı görülmüştür. Sistem piko uydu üzerinde gerçekleştirildiği için aşırı karmaşık bir yapı seçilmemiştir. Bununla birlikte geliştirmeye açık olup sonraki süreçlerde daha karmaşık yapıların uygulanması mümkündür.

INTEGRATION OF THE PICO-SATELLITE FUNCTIONS INTO THE RTOS

SUMMARY

Satellite design has become one of the most popular subjects among the universities during the last years. In this aspect, operating system selection and software have gained importance. In this piece of written research, a real time operating system (RTOS) selection and integration of pico-satellite functions into the RTOS are aimed.

Students of Istanbul Technical University, Faculty of Aeronautics and Astronautics have been designing a pico-satellite. RTOS has been selected, based on "ITU-pSAT-I" project. Additionally, missions have been integrated by considering payloads.

The pico-satellite has mainly three blocks. The first one is a low resolution camera system. The second block consists of a power supply, sensors and a passive magnetic stabilizer. The third one is as a modem system that provides communication with the ground station.

Two types of data buses are used for communication on the pSAT. These data buses are called I²C and RS232. RS232 data bus is used between on-board computer and modem while I²C data bus is used between on-board computer and the camera system and also the second system.

SALVO operating system has been selected as RTOS of the pico-satellite. During the operating system design, three jobs have been assigned seperately in order to execute the three main functions. The highest priority has been assigned for communication with ground station. After that, interconnection functions between the camera systems, power supply, sensors and on-board computer were defined.

Operating system has been prepared as directly running on the pico-satellite. Unfortunately, during the thesis study, real components have not been integrated in one system. So that, payloads have been simulated as personal computer and microprocessors. Main flight computer (MSP430) is the same as

satellite main flight computer. And data busses have been used with original structure as satellite.

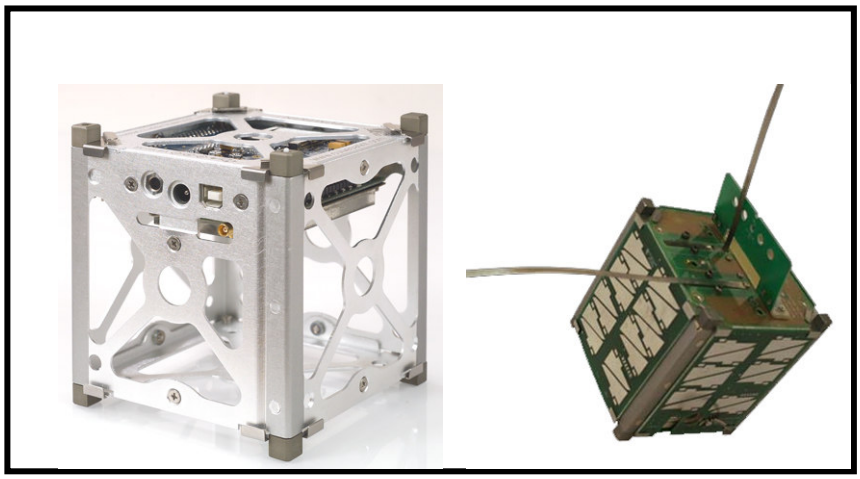
It is seen that, the operating system running on the main flight computer and simulation components were successful. Since the system is operating on a pico satellite, a very complex structure was not be used. Also for the future considerations, it is possible to develop the system in order to use it on more complex designs.

1. GİRİŞ

Çağımızda uydular, gerek günlük yaşantımızda gerekse özel amaçlı kullanımlarda (askeri, bilimsel vb.) insanoğlunun hizmetindedir. 1950 sonlarından itibaren hızla gelişen uzay teknolojileri sayesinde bugün uydu yapımı sadece devletler nezdinde değil, özel firmalar ve üniversiteler tarafından da üretilir hale gelmiştir.

Piko uydu üretimi de üniversitelerin uçak uzay mühendisliği bölümlerinde önemli hale gelmiştir. Bunun başlıca sebebi üretilen sistemin öğrenciler için bulunması güç bir tecrübeyi onlara kazandırmasıdır. Örnek uygulamalar Ek A da mevcuttur.

İstanbul Teknik Üniversitesi Uçak ve Uzay Bilimleri Fakültesi de bu gelişmeleri yakından takip ederek kendi uydusunu yapmaya başlamıştır. Proje uydusu ITU-pSAT-I olarak adlandırılmıştır. Uydu, boyutları ve görev yükleri göz önüne alındığında piko uydu sınıfına girmektedir.



Şekil 1.1 : Piko uydular [1]

Piko uydular yaklaşık boyut olarak 15x15x15 cm ebatlarındadır. Şekil 1.1 de bir piko uydunun temel gövde yapısı ve güneş panelleri ile birlikte son hali gözükmektedir. Piko uyduların fazla sayıda görev yükü bulunmamaktadır. En genel görev yükleri düşük çözünürlüklü kamera ve güç sistemi ile buna bağlı sensör yapılarıdır. Temel amaçları ise kamera ile uzayda görüntü yakalama, telemetri verilerini edinme

akabinde tüm bu bilgileri dünyadaki yer istasyonuna iletmek ve yer istasyonundan gelecek komutları anlayıp icra etmektir.

ITU-pSAT-I projesinde de üç adet görev yükü bulunmaktadır. Bunlar kamera sistemi, iletişim sistemi, manyetik dengeleyici ile sensör yapısıdır. Uydu, tüm sistemleri ana uçuş bilgisayarı (MSP430) ile kontrol etmektedir. İletişim ağında kullanılan veri yolları, I²C ve RS232 seri haberleşme yapılarıdır. Ana uçuş bilgisayarı içine gerçek zamanlı bir işletim sistemi olan Salvo işletim sistemi gömülmüştür.

İlerleyen bölümlerde GZİS'nin ne olduğu, seçiminde dikkat edilmesi gereken hususlar, Salvo işletim sisteminin yapısı ve benzer işletim sistemleriyle kıyaslanması, piko uyduların yapısı ve işletim sistemleri ve son olarak da bir piko uydunun işletim sistemine görev entegrasyonu uygulaması işlenmektedir.

2. GERÇEK ZAMANLI İŞLETİM SİSTEMİ NEDİR?

2.1 Gerçek Zamanlı Sistem Nedir?

Gerçek zaman gerçek hızı gerektirmez. Onun yerine belirlenen işin zamanında yapılmasını amaçlar. Bu durum gerçek zamanlı sistemlerin temel karakteristiğini oluşturan, işin hangi zamanda, hangi konumda olduğunu ve işlemlerin başlangıç ve bitiş anlarının tam olarak belirlenmesi özelliğini içerir. Gerçek zamanlılığın kaynak yönetimi ile ilgili olduğu söylenebilir.

Gerçek zamanlı sistemlerin yazılımlarında üç çeşit yol izlenmektedir.

2.1.1 Tek Parçalı Sistemler

Bu tip yazılımlarda bütün kod bir blok halinde yer alır. Basit sistemler için yazımı pratiktir. Ancak karmaşık sistemlerde bu yapı kullanışlı değildir.

2.1.2 Çekirdek Tabanlı Sistemler

Bu sistemler gerçek zaman çekirdeğini kullanırlar. Bu yapı üretici firma tarafından veya kullanıcı tarafından görevler ve kesmeler kullanılarak oluşturulur.

2.1.3 İşletim Sistemi Tabanlı Sistemler

İşletim sistemi tabanlı yapıyı çekirdek tabanlı yapıdan ayıran en önemli özellik, işletim sisteminin fonksiyon alanının sağladığı olanakların fazlalığıdır. Bu olanaklar dosya yapısı ve kullanıcı ara yüzü gibi özelliklerdir.

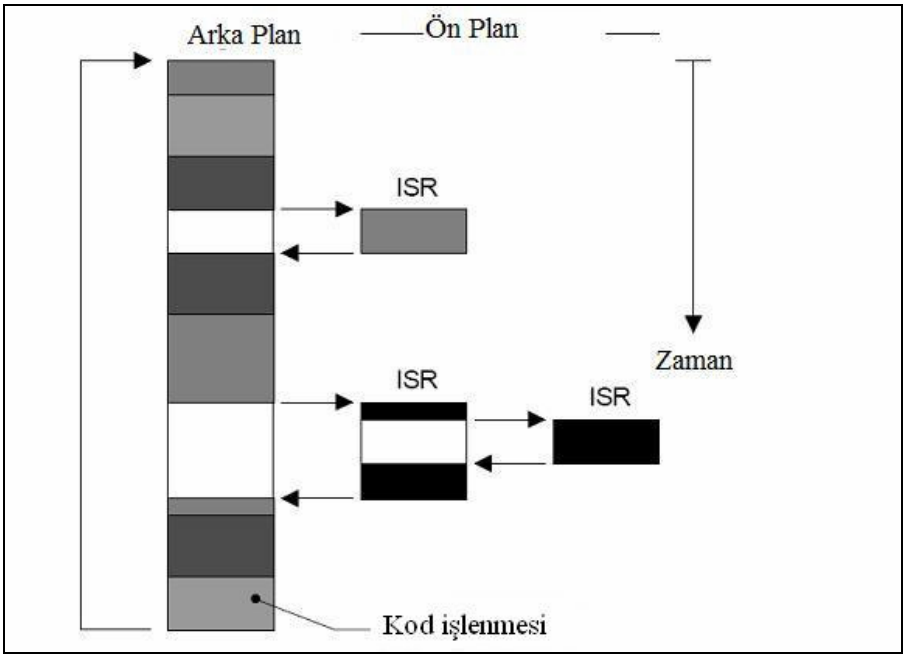
2.2 Gerçek Zamanlı İşletim Sistemi

Jean J. Labrosse' a [2] göre; GZİS, kritik zamanlı işlemlerin eş zamanlı olarak koşturulmasıyla oluşmaktadır. Yazılım kritik zamanlı olayları en etkin şekilde ve uygun zaman aralıklarında icra etmelidir. GZİS' nin kullanımı sistemin dizayn sürecini kolaylaştırdığı gibi hızlandırmaktadır da. Bunu sağlamak için birbirinden bağımsız elemanlar kullanılır ve bu elemanlar görev olarak adlandırılır. Görevler

basit program parçalarıdır. Her bir göreve önem derecesine göre öncelik atanır. Görevlerin öncelik sırası olması, programlamayı daha anlaşılır hale getirip basitleştirmekte ve işlemlerin kontrolünün kullanıcının eline geçmesini sağlamaktadır. GZİS' nin tam olarak anlaşılması için aşağıdaki terimlere açıklık getirilmelidir.

2.2.1 Ön Plan / Arka Plan

Bu yapı Bölüm 2.1.1 de belirtilen tek parçalı kod yapısıdır. Ön Plan / Arka Plan yapısı GZİS değildir. Tek bir sonsuz döngü içerisinden istenilen modüller çağrılmaktadır. Ardışık sıra ile modüllerin koşturulması arka planda olur. Düzenli olmayan kesme servis rutinleri (ISR) ise ön plan olarak adlandırılan yapıda meydana gelir. Ön Plan / Arka Plan yapısı şekil 2.1 de gösterilmiştir.



Şekil 2.1 : Ön Plan / Arka Plan Uygulaması [2]

Kritik işlemlerin gerçekleştirmesi ISR ler tarafından yapılmaktadır. Buda, ISR lerin normalden daha fazla zamana ihtiyaç duymalarını gerektirmektedir. Arka plandaki modüle bilgi ancak ISR nin işi bittikten sonra gitmektedir. Bu yapı basit sistemler için kullanışlı olmasına karşın karmaşık sistemlerde kullanışlı değildir. Sistem karmaşıklaştıkça kodu yazmak zorlaşır ve yeni durumların eklenmesine olanak vermez.

2.2.2 Çoklu Görev

Çoklu görev, sürecin planlanması ve bu doğrultuda mikro işlemcinin görevler arasında tetikleme yapmasıdır. Çoklu görev, işlemin daha kolay ve ufak halde şekillenmesine olanak verir. Bununla birlikte işlemci, uygun olan görevleri paylaştırır. Programcıya, karmaşık gerçek zamanlı sistem yapılarının kurulmasında esneklik sağlar.

2.2.3 Çekirdek

Çekirdek, görevlerin düzenlenmesinden ve görevler arası iletişimin sağlanmasından sorumludur. Çekirdek diğer bir görevi koşturmaya karar verdiğinde o an için koşan görevin içeriği o görev için oluşturulmuş bellekteki yığın içerisine yazılır. Yeni koşturulacak görev de kendine ait yığından bir önceki kaydedilmiş bilgilerini alarak işlemeye devam eder. Bu sürece genel durum anahtarlaması veya görev anahtarlaması denir. Durumların kaydedildiği yığın yapısına da görev kontrol bloğu (TCB) denir.

2.2.4 Kesmeler

Gerçek zamanlı sistemlerde önemli bir durum da kesme cevaplarının zaman gereksinmesidir. Bütün GZİS' ler kritik öncelikli kodları çalıştırırken kesmeleri geçersiz kılarlar. Kesme ihmali nekadar uzun olursa, kesmenin gecikme süresi de okadar yüksek olmaktadır. Bundan dolayı, GZİS' leri 50 µs den daha az bir süre için kesmeyi ihmal eder. Fakat asıl olan bunun olabildiğince az olmasıdır.

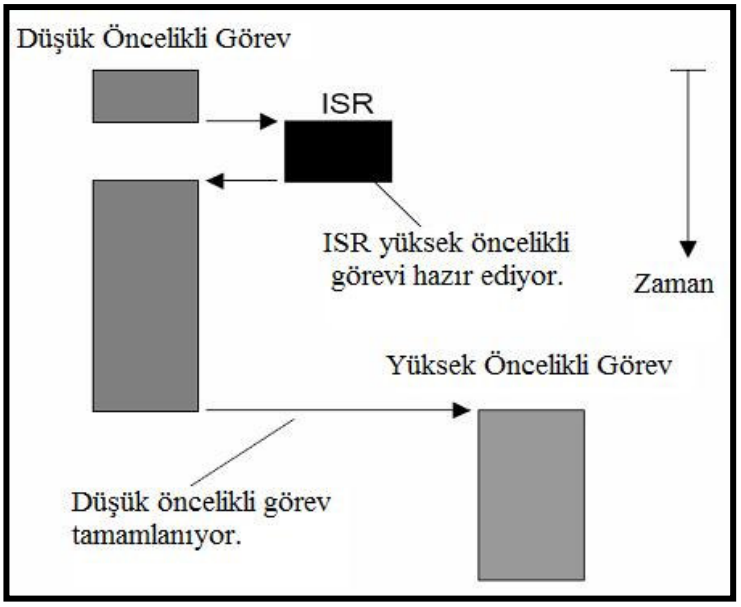
2.2.5 Planlayıcı

Planlayıcı çekirdeğin bir parçasıdır. Görevlerin işleyişi sırasında bir sonraki işlemde hangi görevin ve ne zaman koşacağını belirler. Birçok gerçek zamanlı çekirdek öncelik tabanlı çalışır ve görevlere öncelik atanır. Planlayıcılar iki çeşittir. Öne almayan planlayıcılar ve öne alan planlayıcılar.

2.2.5.1 Öne Almayan Planlayıcılar

Bu tarz planlayıcılara kooperatif çoklu görev de denir. Uygulamadaki görev tamamen işlenmeden sonraki göreve geçilmez. Görevin icrası sırasında daha öncelikli bir görev için ISR oluşsa bile önce koşan görev işlenir ve bitmesiyle

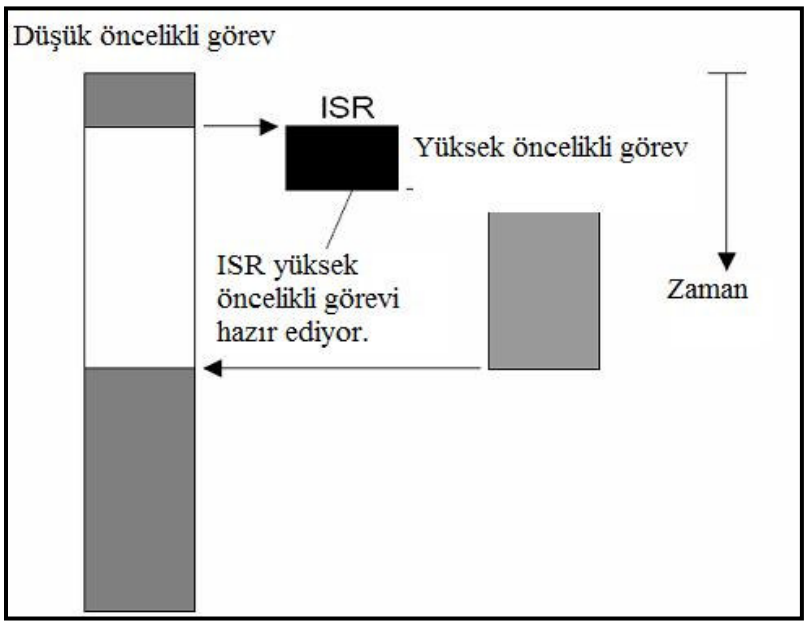
öncelikli görev işlenmeye başlanır. Bu yapı, öne alan planlayıcılara göre daha yavaştır. Ancak programcı için yapıyı oluşturmak kolaydır. Şekil 2.2 öne almayan planlayıcı yapısını göstermektedir.



Şekil 2.2 : Öne Almayan Planlayıcı yapısı [2].

2.2.5.2 Öne Alan Planlayıcılar

Bu yapıdaki planlayıcılarda ISR tarafından yüksek öncelikli görev hazır hale getirildiğinde, işlenmekte olan görev düşük öncelikli ise kesilir ve yüksek öncelikli görev koşturulur. Bu görevin işi bitince yarım kalan görev kaldığı yerden devam eder. Genelde GZİS' leri öne alan planlayıcı yapısı kullanırlar. Çünkü bu yapı daha hassastır. Şekil 2.3' te öne alan planlayıcı yapısı gösterilmiştir.



Şekil 2.3 : Öne Alan Planlayıcı yapısı [2]

2.2.6 Tekrardan Giriş

Bir fonksiyonun birden fazla görevde kullanılması özelliğidir. Kesme durumunda veri kaybına uğramadan kaldığı yerden devam edebilir. Öne almayan planlayıcı yapısı olan GZİS' lerinde, görev ve ISR arasında fonksiyon paylaşımı olmadığı sürece bu tarz fonksiyonlara ihtiyaç duyulmaz.

3. GERÇEK ZAMANLI İŞLETİM SİSTEMİ SEÇİMİ

GZİS seçiminde en önemli etken hiç şüphe yoktur ki belirlenen hedeflerin başarılmasını sağlayacak özellikleri bulundurmasıdır. Jean J. Labrose' a [2] göre bu temel özellikler şöyledir.

Maliyet yönünden düşük olmalıdır. Kesme gecikmelerindeki belirsizlik en az seviyede olmalıdır. Çekirdek servisleri, görevlerin koşturulmasında yeterli gerçeklik zamanını sağlamalıdır. Jean J. Labrose' a [2] göre orta seviye bir GZİS en az yirmi adet görevi düzenleyerek çoklu görev yapabilme yeteneğine sahip olmalıdır. Görevler dinamik olarak yaratılabilmeli veya silinebilmelidir. Semafor düzenleme servisleri bulunmalıdır. Zaman gecikmesi ve zaman aşımı servisleri çekirdek tarafından desteklenmelidir.

Michael Barr' a [3] göre ise bir GZİS seçiminde dikkat edilmesi gereken hususlar şöyle sıralanmaktadır.

- Dil ve mikro işlemci desteği : Kullandığımız GZİS'nin hangi programlama dillerini desteklediği ve hangi mikro işlemciler üzerinde koşabildiği kriteri.

- Alet uyumluluğu : GZİS' nin derleyici, bağlayıcı ve kaynak kod ayıklayıcısı ile uyumu önemlidir.

- Performans : Kullanılan donanım ile işletim sisteminin etkileşimi sonucu oluşan yapının verimliliği önemlidir.

- Araç Sürücüleri : Genel donanımların GZİS tarafından desteklenmesi çevre birimlerinin kontrolünü kolaylaştıracaktır.

- Kaynak kodu veya nesne kodu : Kaynak koda müdahale etmek kontrolün tamamen programcının elinde olmasını sağlarken bunun lisans gerektirmesi maliyeti artırmaktadır. Aksi takdirde hazır nesne kodlarını veya kütüphanelere bağlanmış kodları kullanmak gerekmektedir.

GZİS' ler uydu kontrol sistemlerinin dışında; robot endüstrisinde, füze güdüm sistemlerinde, iletişim ağlarında ve endüstriyel süreç kontrolünde kullanılmaktadır. Tablo 3.1 de en yaygın ticari GZİS'leri verilmiştir. Daha kapsamlı gösterim Ek B de belirtilmiştir.

VxWorks
pSOS
Nucleus PLUS
QNX
Windows C
Salvo
LynxOS
ThreadX

Tablo 3.1 : Ticari GZİS'leri

4. SALVO İŞLETİM SİSTEMİNİN YAPISI VE BENZER BİR SİSTEM İLE KIYASLANMASI.

4.1 Salvo İşletim Sistemine Genel Bakış

Salvo düşük maliyetli, az miktarda belleğe ihtiyaç duyan, birçok mikro işlemci tarafından desteklenen, yüksek performanslı bir GZİS' dir. Salvo' nun ilk oluşumu, yarış arabalarının veri toplama sistemleri için düşük maliyetli ve yüksek performanslı sistem oluşturma çabasıyla meydana gelmiştir. İlk halinde makine dili kullanılırken daha sonrasında C programlama diline geçilmiştir.

Salvo ticari bir yazılım olup açık kaynak kodlu değildir. Açık kaynak kod sadece profesyonel sürümünde verilmektedir. Diğer sürümlerinde ise nesne kodları ve derlenmiş kütüphane kodları kullanılmaktadır. Ücretsiz sürümünde ise durum ve görev sayısına sınır getirilmiştir.

Salvo işletim sisteminin çalışabilmesi için minimum gereksinimleri aşağıdaki üç maddede gösterilmektedir:

- 3. nesil ANSI-C uyumlu bir C derleyicisi.
- Donanımsal çağırma yapısına sahip bir mikro işlemci.
- En az 4 kademeli dönüş yığını olan bellek.

Salvo, öncelik tabanlı bir işletim sistemidir. Bu öncelikler 16 kademeye ayrılmıştır. Olaya bağlı sürüş yöntemi ile görevler arası geçişi sağlar. Bölüm 2.2.5.1 de belirtilen öne almayan yapıda planlayıcı kullanan işletim sistemidir. Salvo kesme oluşup ISR tarafından yüksek öncelikli görev hazır hale getirildiğinde koşturmakta olduğu göreve devam eder ve bu işlem bittikten sonra, hazır olan yüksek öncelikli görevi icra etmeye başlar. Salvo tüm GZİS'lerinin olmazsa olmazı çoklu görev yapısını desteklemektedir. Ücretsiz sürümü dışındaki tüm sürümlerinde görev oluşturma sayısına bir sınırlama getirilmemiştir. Ancak temel ayarlarında 255 görev sınırlaması bulunmaktadır.

4.2 Salvo İşletim Sisteminin Yapısı

Salvonun etkin bir biçimde kullanılabilmesi için Bölüm 4.1 ve bu bölümde yer alan konuların özümsenmesi gerekmektedir. Bunların yanında C programlama dilinin de etkin biçimde kullanılması gerekmektedir.

4.2.1 Öncelik Tabanlı Çoklu Görev

Görevlerin önceliklerinin doğru atanmaması sürecin akışını dramatik yönde değiştirebilir. Bu nedenle süreç içinde belirlenen görevlerin önceliklerinin kesin ve doğru olarak belirlenmesi gerekmektedir. Salvo işletim sisteminde 0 en yüksek öncelikli görevi gösterirken, 16 en düşük öncelikli görevi göstermektedir. İki farklı görev aynı önceliğe sahip olabilmektedir.

Salvo' nun en önemli özelliklerinden biri önceliklerin dinamik veya statik olarak atanabilmesidir. Statik öncelikler program derlenirken görevlere atanır ve program koştuğu sürece değişmez. Dinamik öncelikler ise program koşarken dahi değiştirilebilir. Bu yapı sürecin düzenlenmesinde önemli esneklikler getirmektedir. Tablo 4.1' de öncelik değerlerini ayarlamak için Salvo' da kullanılan komutlar belirtilmiştir.

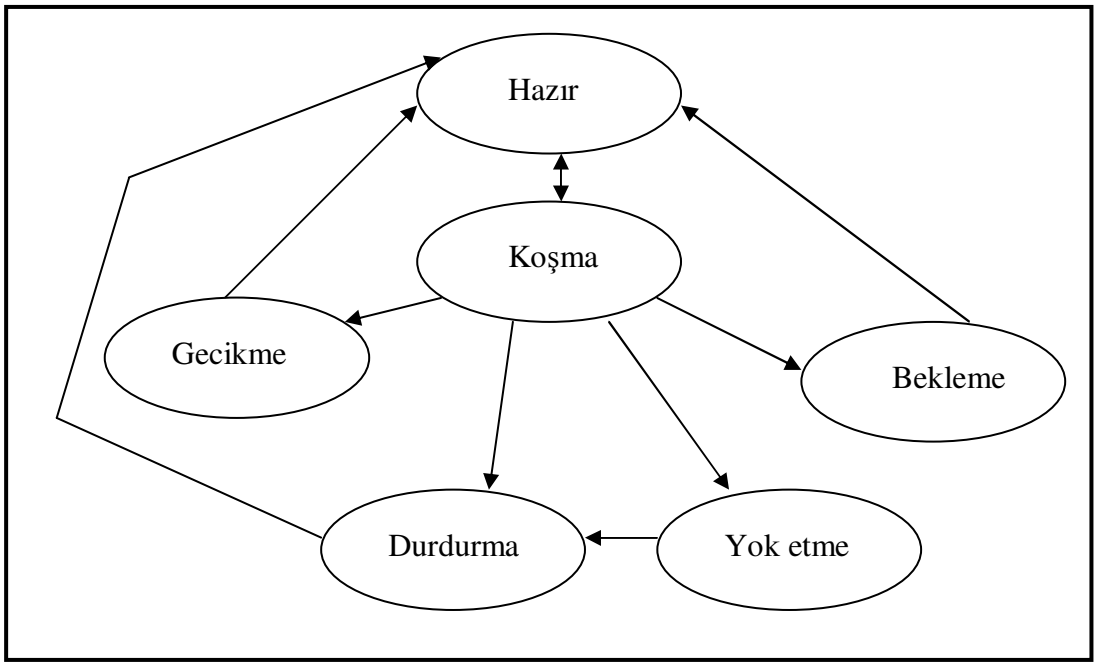
OS_SetPrio( )
OSGetPrio( )
OSGetPrioTask( )
OSSetPrioTask( )
OSDISABLE_TASK_PRIORITIES

Tablo 4.1 : Öncelik tanımlama komutları ve fonksiyonları [4]

4.2.2 Görev Durumları

Salvo işletim sisteminde planlayıcı, görevlerin durumlarına göre karar vererek gerekli işlemleri yapar. Bu durumlar altı çeşittir (Koşma, hazır, bekleme, gecikme, durdurma, yok etme). Görevin işlenme süreci koşma durumudur. Koşma durumuna yalnızca hazır durumundan geçilebilir. Koşma durumundan çıkan görev bekleme,

gecikme, durdurma, yok etme veya tekrardan hazır durumuna geçebilir. Bekleme, gecikme ve durdurma durumlarından; sadece hazır durumuna geçilebilir. Yok etme durumundan ise yalnızca durdurma durumuna geçilebilir. Şekil 4.1 bu durumların özetini vermektedir.



Şekil 4.1 : Salvo İşletim Sisteminde Görev Durumları

İki veya daha fazla görevin durumu hazır olduğunda planlayıcı önceliklerine bakarak hangisinin icra edileceğine karar verir. Program çalışırken yeni görevler yaratılabilir veya yok edilebilir. Tablo 4.2' de Salvo' daki görev durumlarını ayarlayan komutlar verilmiştir.

OSCreateTask( )
OS_DelayTS( )
OS_Destroy( )
OS_Stop
OSDestroyTask( )
OSStartTask( )
OSStopTask( )

Tablo 4.2 : Görevlerin durumlarını belirleyen komutlar

4.2.3 Gecikmeler ve Zamanlayıcılar

Gecikmeler ve zamanlayıcılar Salvo işletim sisteminde görevler arasında durum geçişlerini yapmak için kullanılan yöntemlerdendir. Bunun dışında belli döngülerin ayarlanmasında da kullanılmaktadır. Salvo işletim sisteminde saat yoktur. Zaman, mikro işlemci tarafından üretilen sistem tikleri kullanılarak oluşturulur. Bu durumda zaman akış hızı işlemcinin hızı ile doğrudan bağlantılıdır. Tablo 4.3'te Salvo işletim sisteminin gecikme ve zamanlayıcı komutları verilmiştir.

OS_Delay( )
OSGetTicks( )
OSStartCycTimer( )
OSTimer( )
OS_DelayTS( )

Tablo 4.3: Gecikme ve zamanlayıcı komutları

4.2.4 Olaylar ve Görevler Arası İletişim

GZİS' lerde görevler arası iletişimi sağlamak için çeşitli yollar kullanılır. Salvo işletim sisteminde görevler arası iletişim üç yolla gerçekleştirilir. Bunlar semafor yöntemi, mesaj yöntemi ve mesaj sırası yöntemleridir. Bütün bu yöntemler iki temel prensibi barındırır. Mesaj gönderme ve mesaj bekleme. Hangi iletişim yöntemi kullanılırsa kullanılsın Salvo içinde önce oluşturulması ve daha sonrasında başlangıç değerlerinin ataması gerekmektedir.

4.2.4.1 Semaforlar

Semaforlar iki çeşittir. İkili semaforlar ve sayma semaforları. İkili semafor sadece iki değer alabilir. Bu değerler 1 veya 0 dır. Sayma semaforları ise belirtilen boyutta değer aralığına sahip olacaktır. 8 bitlik bir sayma semaforu 0 dan 255 e kadar değer alabilir. Durum bayrakları ikili semaforlar için kullanılmaktadır. Durumun meydana gelip gelmediği ikili semafordan takip edilir. Eğer ikili semafor 0 ise durum gerçekleşmemiştir. Durum meydana geldiğinde ikili semafor değeri 1 olur ve işaret verilir.

4.2.4.2 Mesajlar

Mesajlar, görevlere rastgele bilgi gönderirler. Bu bilgi sayı, katar, dizi, fonksiyon veya işaretçi olabilir. Mesaj şekli değişir değişmez gönderen ve alan görevler bunun farkına varır. Mesajda belirtilen içerik görevlerin durumunu belirleyecektir.

4.2.4.3 Mesaj Kuyrukları

Bu yapı, mesaj yapısının uzatılmış halidir. Mesaj kuyruk yapısı ile görevler arası etkileşim mikro işlemcinin belleğine duyulan ihtiyacı arttıracaktır.

4.3 LynxOS GZİS Yapısı ve Salvo ile Karşılaştırılması

LynuxWorks firması tarafından 1988 tarihinde üretilmiştir. Kesin olarak yapılması gereken işin belirlendiği ve yüksek performans gerektiren uygulamalar için geliştirilmiştir [5].

Temel özellikleri 4 aşamadan oluşmaktadır [5]. Bunlar;

1. Sistem çekirdeği, öne alma planlayıcı yapısına sahiptir. Ayrıca tekrar giriş özelliği bulunmaktadır. Bu özellikleri ile kritik zamanlı görevler için önem kazanmaktadır.

2. Çekirdek yapısının çoklu görevi desteklemesi amaçlanmıştır. Bu sayede kullanıcı programları çevrebirim sürücüleri ve diğer çekirdek servisleri kendi görevlerini yaratabilmektedir.

3. Mikro işlemcinin sayfa bellek düzenleme ünitesini kullanmayı amaçlamıştır. Bu sayede çekirdekte dağınık adres boşluklarının oluşması engellenmiştir.

4. Unix ve POSIX API' lerini kullanmaktadır.

Birinci hedef olan öne alma yapısı ve tekrar giriş üç temele dayandırılmıştır. Birincisi, ilk giren ilk çıkar yapısıdır. İkincisi Round Robin yapısı. Üçüncüsü ise öncelik tabanlı kuantum yapısıdır. 256 adet öncelik sırası görevlere atanabilmektedir. Ayrıca bu önceliklerde 512 alt önceliğe bölünebilmektedir. İşletim sisteminin %90 ı C dilinde yazılmıştır.

Lynx_OS GZİS, Salvo ile karşılaştırıldığında temel özelliklerinin aynı görünmesine karşın önemli bir farklılık vardır. Planlayıcı yapıları farklıdır. Salvo öne almayan planlayıcı yapısına sahiptir. Bu, kolay programlanabilme özelliği kazandırmaktadır. Ancak Lynx_OS işletim siteminin de öne alan planlayıcı yapısıyla, Salvo' ya göre daha hassas çalışma yeteneğine sahip olduğu görülmektedir. Her iki işletim sistemi de C dilini kullanmaktadır. Salvo işletim sistemi görevlerine atayabildiği öncelik sayısı bakımında Lynx_OS karşısında zayıf kalmaktadır. Bununla birlikte her iki işletim sistemi de uzay uygulamalarında kullanılmaktadır.

5. PİKO UYDULARIN YAPILARI

Bir piko uydu oluşturulurken dağıtılmış bilgisayar mimarilerinden biri tercih edilir [5]. Bu mimariler sayesinde karmaşık araçların ve sistemlerin geliştirilmesi kolaylaşmaktadır. Bu avantajlar iyi tanımlı ara yüzleri, esnek kompozisyonu, doğrudan fonksiyon yapı eşleşmelerini, standart bileşenleri, artımlı testi ve diğer avantajları içerir.

Bu bağlamda, dağıtılmış bilgisayar kavramı, hesaplamanın; farklı bileşenlerde, alt sistemlerde, sistemlerde ve tesislerde yer alabilen bir dizi işlemci tarafından merkez dışında yapılması anlamında kullanılmaktadır.

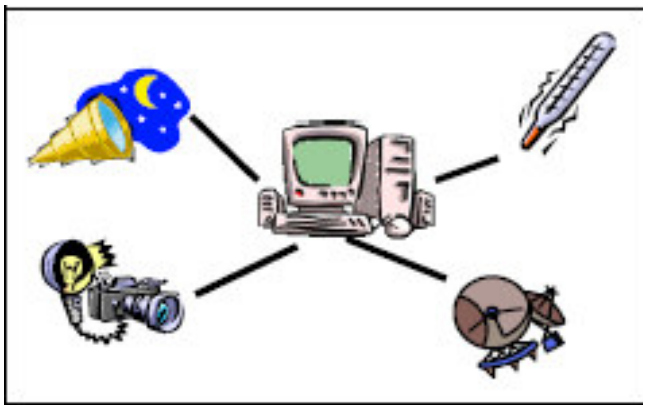
Bu işlemciler, veri/uygulama paylaşım özelliklerine sahip genel amaçlı bilgisayarlar veya mikro işlemciler olabilirler. Belli bir göreve odaklanmış ortak işletimi sağlayan mimariye sahip olabilirler, ve/veya her biri belirli görevleri etkili bir şekilde çalıştırmada veya belirli alt sistemleri kontrol etmede etkili kılınmış olabilir. (Ör: akıllı çevrebirimleri).

Bilgisayar sisteminin mimarisi (dağıtılmış veya merkezi) bileşenleri birbirine bağlayan fiziksel veya mantıksal çerçeveyi ifade eder. Mimari, alt-sistemler arası veri transferi için yolunu belirler ve hem kablolama donanımına hem de modülerliğe büyük ölçüde bağlı olabilir. Tipik mimari açıklamaları aşağıdaki gibidir:

5.1 Yıldız Mimarisi

Yıldız (merkezi veya dağıtılmış) mimarisi, tahsis edilmiş bağlantılar aracılığıyla diğer tüm sistem birimlerine bağlayan bir işlemciden oluşur. Bu yaklaşım genellikle büyük kablolama donanımlarını ve çevre birimlerinin dizaynının, merkezi bilgisayarın donanım/yazılımlarına bağlı olunmasını gerektirir. Kablolama yoğunluğu piko uydular için başlı başına bir sorundur. Yıldız konfigürasyonu bağlı bileşenlerin işlemcileri olmadığı zamanda merkezi mimariye örnektir. Yıldız konfigürasyonu, bileşenlerin işlemcileri olsa da dağıtılmış mimaride kullanılabilir.

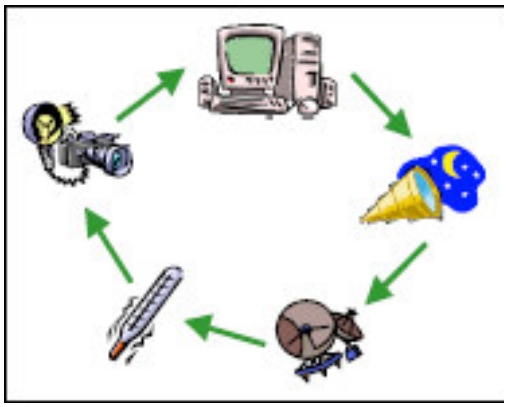
Böylece bileşenlerden birisinin arızalanması durumunda bile tüm sistem çalışmaya devam edecektir. Şekil 5.1 de yıldız mimarisi gösterilmektedir.



Şekil 5.1 :Yıldız Mimarisi.

5.2 Halka Mimarisi

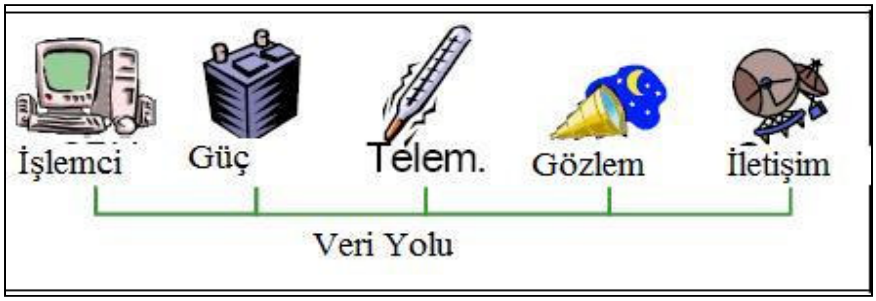
Halka biçiminde dağıtılmış bilgisayar mimarisinde işlemciler yakın bir halka ile bağlanmıştır. İletişim bir işlemciden diğerine geçen jeton tarafından gerçekleştirilir. Halkalar nispeten daha az kablolama donanımı gerektirir. Bu etken uydu yapısı için olumludur. Ancak alt sistem uygulamalarında küçük tasarım bağımlılığı görülebilmektedirler. Alt sistemler çöktüğünde veya kesildiğinde iletişim kesintisine de yol açabilmektedir. Bu durum ise uydu için yaşamsal önem arz etmektedir. Halka mimarisi uydularda kullanılmak için uygun değildir. Şekil 5.2 de halka yapısı gösterilmiştir.



Şekil 5.2 : Halka Mimarisi

5.3 Doğrusal Veri Yolu Mimarisi

Doğrusal dağıtılmış işlemci mimarisi tüm alt sistemlerin bağlı olduğu standart, paylaşımlı, doğrusal veri yolundan oluşur. Bu genellikle az kablolama donanımını gerektirir ve yolla problemsiz bağlantı sağlar. İşlemciler yoldan veri transfer ettiklerinde iyi tasarlanmış bir iletişim protokolü düzenlemesine ihtiyaç duyar. Bu veri yolu mimarisi kablonun iletim güvenliği sağlandığı sürece uydu için ideal bir mimari oluşturmaktadır. Şekil 5.3' de doğrusal dağıtılmış işlemci mimarisi şeması görülmektedir.



Şekil 5.3 : Doğrusal mimari

5.4 Melez Mimari

Melez mimari, yukarıda sözü edilen mimarilerin bir veya birden fazla farklı işlemci içeren birimlerin bir sistem altında bağlamasından oluşur. Bu, farklı işlemciler arası iletişim gereklilikleri çoğu zaman farklı mimarilerce en iyi şekilde karşılandığı için aranan bir özelliktir.

5.5 Katmanlı Mimari

Katmanlı mimariler, yukarıda sözü edilen mimarilerin aynı işlemci üzerinden tek bir sistemde bağlanmasıyla oluşur. Örneğin, aynı işlemci komut ve kontrol bilgisi için düşük hızlı veri yolu mimarisine sahipken, bilim verisini hızlı veri yolu bağlantısı ile sağlayabilir. Bu farklı işlemci fonksiyonları farklı mimariler tarafından en iyi şekilde adreslenmesini gerektiren bir yapıdır.

5.6 Sistem Tasarımı

Dağıtılmış bilgisayar yaklaşımının avantajları belirtildikten sonra, hedeflenen görev için en uygun mimari seçilmelidir. Basitlik ve maliyet açısından dengeli bir sistem

aranmalıdır. Aynı zamanda uydu yapımı için gerekli alt yapının sağlanıp uygun malzemelerin seçilmesi gerekmektedir.

Uydunun tasarım aşamasında temel olan kurallar aşağıda sıralanmıştır:

5.6.1 Alt Sistem Ana Kartları

Dağıtılmış mimarinin verimli olarak çalışması için, tüm alt sistemlerin aynı işlemciyi kullanmalarına gerek yoktur. Veri yolu ara yüzünün standart olma koşulu gözetildiği sürece, hiçbir alt sistem işlemci farklılıklarından etkilenmez.

5.6.2 Mikro İşlemciler

Her sistemin kendi işlemcisinin olması ile sistemin hızlanması, hata risklerini azaltacaktır. İşlemcilerde gözetilmesi gereken husus; sistemin karmaşıklığı karşısında yazılan programın işlemcinin bellek yeteneği tarafından karşılanıp karşılanamayacağıdır.

5.6.3 Yazılım

Üçüncü bölümde belirtilen hususlar göz önüne alınmalıdır.

5.6.4 Veri Yolu

Sistemin bileşenlerini en iyi şekilde birbirine bağlayarak iletişim sağlayacak yapı seçilmelidir. Yukarıda belirtilen hususlar doğrultusunda doğrusal veri yolu mimarisi ve melez yapı en uygun veri yolu mimarilerini oluşturmaktadır.

5.6.5 Telemetri Sistemleri

Uydunun veri edinimi sağlayan yapı olduğu için imkan elverdiğince yüksek çözünürlüklü olması gerekmektedir. Güç sarfiyatını minimum seviyede tutabilecek yapılar olmalıdır.

6. PİKO UYDUNUN İŞLETİM SİSTEMİNE GÖREV ENTEGRASYONU UYGULAMASI

Yukarıdaki bölümlerde belirtilen hususlar göz önünde tutularak. Piko uydunun işletim sistemine görev entegrasyonu yapılmıştır. Benzetim sisteminde yapılan testler sonucunda sistemin düzgün çalıştığı görülmüştür. Böyle bir uygulamada görevin tam olarak tanımlanması, görevi gerçekleştiren sistemlerin yapısının en ince ayrıntısına kadar bilinmesi gerekmektedir.

6.1 ITU-pSAT-I Uydusunun Özellikleri

Pumpkin firmasının ürettiği Cube Sat Kit olarak adlandırılan bileşenler kullanılarak uydunun temel yapısı oluşturulmuştur. Uydunun temel olarak 6 adet parçadan oluşması öngörülmüştür. Bunlar, alt sistemleri üzerinde taşıyan iskelet yapısı, iletişim sistemi, güç sistemi, kamera sistemi, uçuş kontrol bilgisayar sistemi ve tüm alt sistemlerin birbiri ile haberleşmesini sağlayacak veri yolu sistemidir.

Uydunun yaklaşık 650km irtifada güneşe eşzamanlı yörüngede çalışacağı varsayılmaktadır. Günde beş veya altı defa ve de her birinde yaklaşık on üç dakika süre ile yer istasyonunun uydu ile temas kurabileceği düşünülmektedir. ITU-pSAT-I de yörünge hesabı yapılmayacaktır. Bu sayede uydu üzerindeki sistemler fazla iş yükünden kurtulmuş olmaktadır.

6.2 Alt Sistemler

6.2.1 İskelet Yapı

Alüminyum 5052-H32 malzemesinden üretilmiştir.. İçerisindeki yuvalara görev yükleri yerleştirilmektedir. İskelet yapının hazır alınmasının nedeni işgücünden kazanmak ve denenmiş bir sistemi kullanarak zaman sarfiyatının azaltılmasını sağlamaktır. Uydu yörüngeye bırakıldığı anda köşe kısımlarındaki anahtarlar açılarak sistemin güç kaynaklarının aktif olması sağlanacaktır.[6]

6.2.2 Güç Sistemi

Güç sistemi, içerisinde iki adet enerji kaynağı bulundurmaktadır. Bunlar güneş panelleri ve pillerdir. Piller iskelet yapının içine monte edilmiştir. Güç sistemi 3.3 V ile 5V arasında gerilim üretmektedir.

Telemetri sensörleri güneş panellerinin üzerindeki sıcaklığı, üretilen gerilim düzeyi ve akım miktarını ölçmekte, pillerin sıcaklık, voltaj ve akım verilerini edinmektedir. Ayrıca veri yolunun üzerindeki akımı da ölçmektedir. Edinilen veriler I²C (Inter-Integrated Circuit) veri yolu üzerinden iletilmektedir.[6]

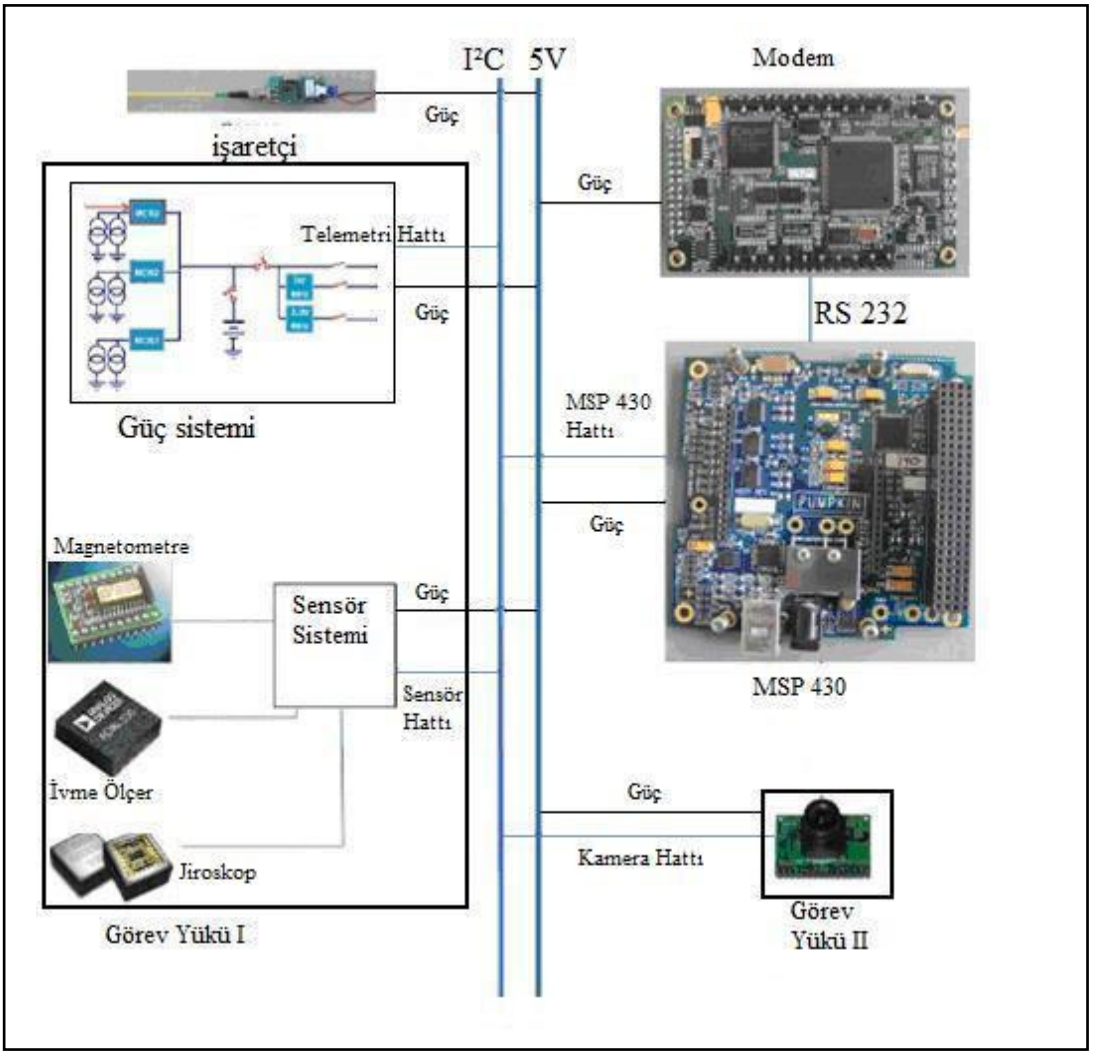
6.2.3 İletişim

İki çeşit iletişim cihazı kullanılmaktadır. Bunlar MHX-425 modem ve işaretçidir. Modem haberleşme veri yolu olarak RS232 (Recommended Standard 232) seri haberleşme yapısını kullanmaktadır ve doğrudan uçuş kontrol bilgisayarına bağlıdır. İşaretçi ise I²C veri yolunu kullanmaktadır.[6]

6.2.4 Ana Uçuş Kontrol Bilgisayarı

Pumpkin firmasının FM430 olarak adlandırdığı gömülü kart uçuş bilgisayarı olarak işlev görmektedir. Sistem Texas Instruments firmasının ürettiği MSP430F1611 işlemcisini kullanmaktadır. Bu işlemci 16 bitlik düşük güçte çalışabilen 8MHz lik bir mikro işlemcidir. 55 KB anlık belleği ve 10 KB RAM i bulunmaktadır. İki adet RS232 çıkışı, bir adet I²C ve bir adet de SPI (Serial Peripheral Interface) veri yoluna sahiptir.[6]

Şekil 6.2 de piko uydunun alt sistemleri ve bu sistemler arasındaki veri yolları gösterilmiştir.

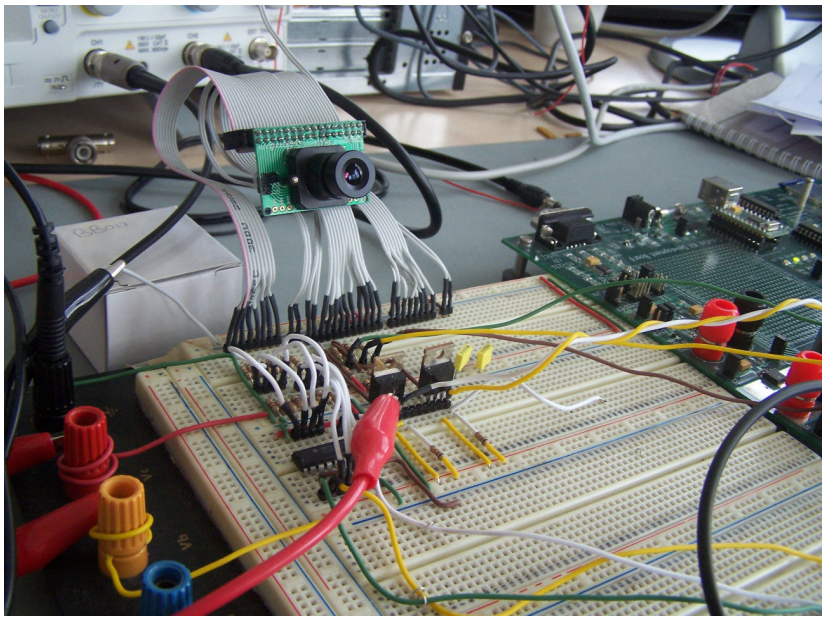


Şekil 6.2 : Uydunun Alt Sistemleri

6.3 Görev Yükleri

Sistemin üç görev yükü vardır.

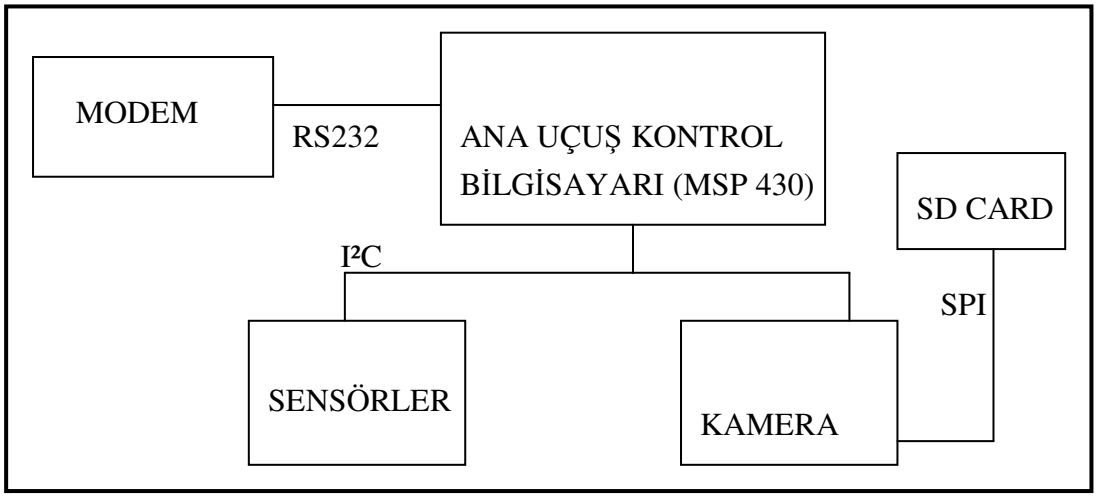
1. Yer istasyonu ile iletişimi sağlayan MHX-425 modem ve işaretçidir.
2. Düşük çözünürlüklü kamera ve bu kamerayı kontrol eden mikroişlemciden oluşmaktadır. Şekil 6.3' de bu yapı gösterilmiştir.
3. Telemetri sisteminin ve pasif mıknatısın bulunduğu yapıdır.



Şekil 6.3 : İkincil görev yükünün masa üstü modeli [6]

6.4 Benzetim Sisteminin Oluşturulması

Benzetim sisteminde ana uçuş kontrol bilgisayarı gerçek sistemde kullanılan mikro işlemci ile aynıdır. Diğer yapılar ise tezin hazırlanması esnasında tümleşik olmadığı için birebir kullanılamamıştır. Haberleşme görev yükü olan modem, yer istasyonu ile birleştirilerek kişisel bilgisayar ile benzeştirilmiş ve ana uçuş kontrol bilgisayarı ile RS232 veri yolu üzerinden bağlantı sağlanmıştır. Sekil 6.4' de benzetim sisteminin temel yapısı gösterilmiştir.



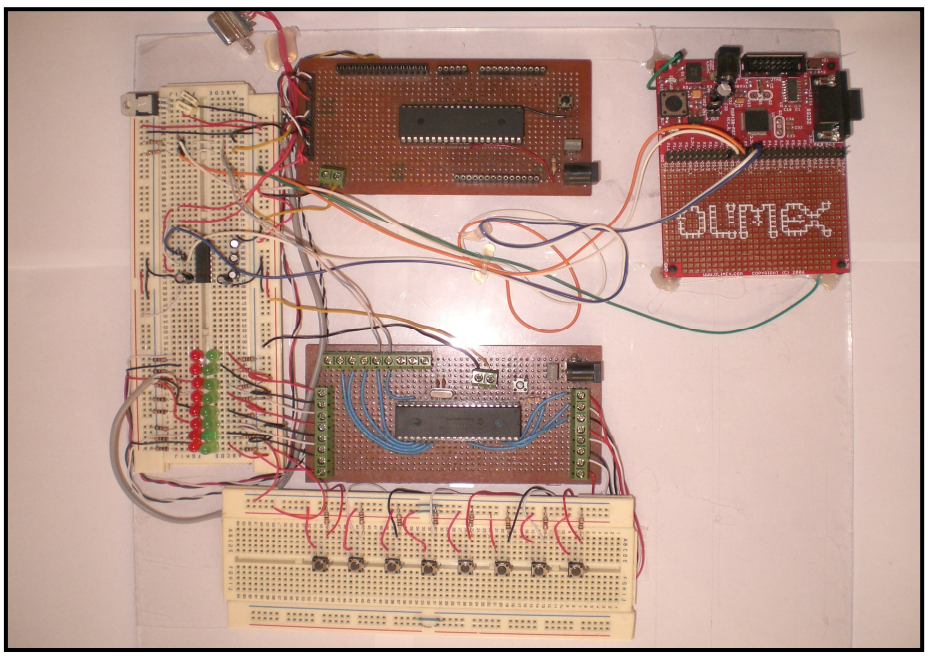
Şekil 6.4 : Benzetim Sisteminin Genel Yapısı

Telemetri verileri ve sensörlerin benzetimi ise PIC 16F877 mikro işlemcisi ile gerçekleşmiştir. Bu işlemcinin 8 bitlik sayısal girişi gerçek sistemin sensörlerine benzeştirilirken, aynı işlemcinin 8 bitlik sayısal çıkışı da ana uçuş kontrol bilgisayarından gelen komutların algılanıp algılanmadığını göstermektedir.

Benzetim sisteminde kamera donanım olarak sisteme eklenememiştir. Gerçek görev yükünde kullanılan kamera işlemcisinin bir alt modeli kullanılmıştır. Fiilen tüm kamera komutları sisteme gömülmüştür. Şekil 6.5' de sistemin gerçek fotoğrafı bulunmaktadır. Kamera ile ilgili komut listesi aşağıdaki gibidir.

//Komut Listesi:

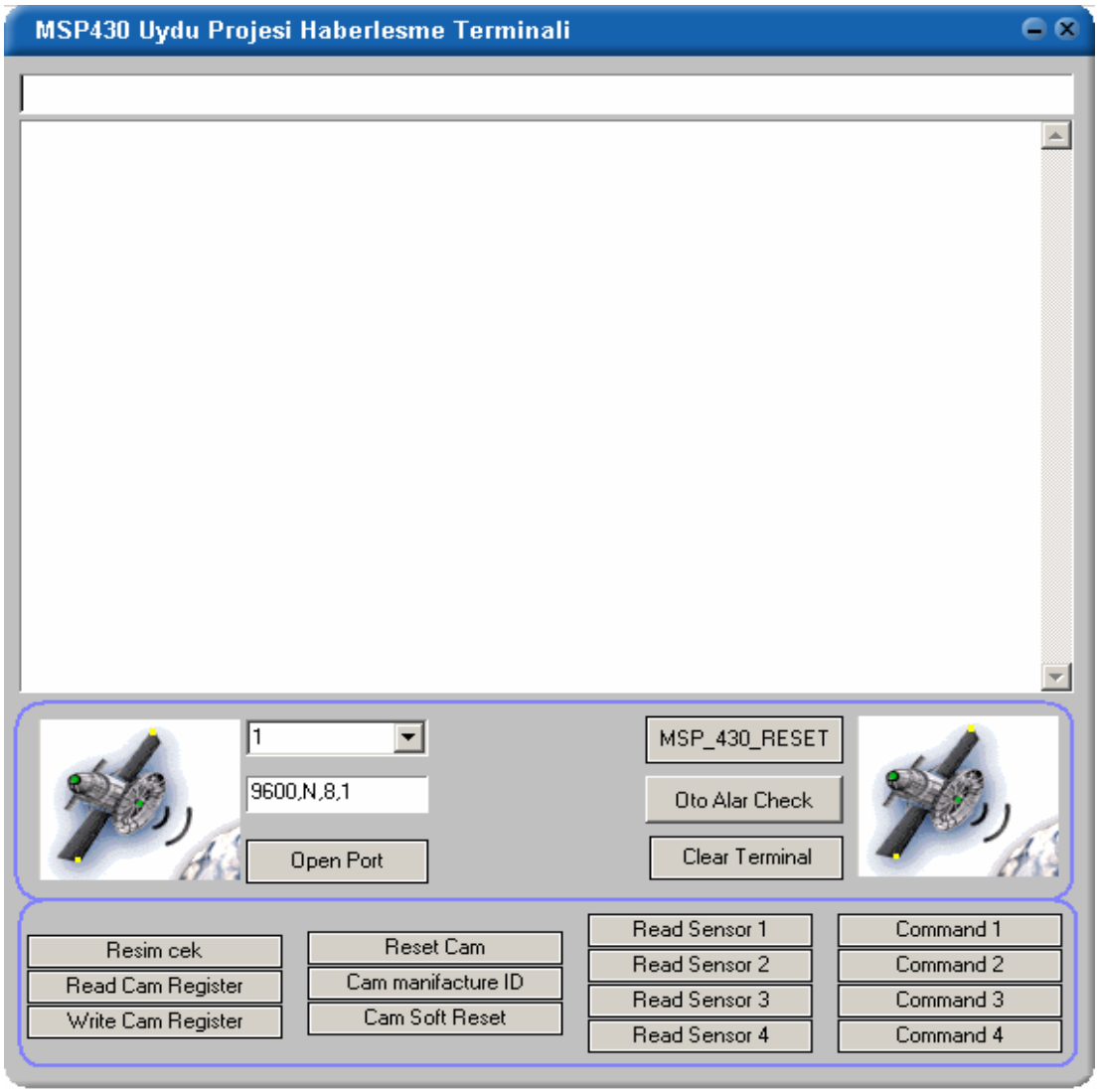
- //T: Bir kare al ve transfer et.
- //R: Yazmaçı oku.
- //W: Yazmaça yaz.
- //D: Kamerayı ilk ayarlandığı hale getir.
- //S: Üretici IDleri kontrol et.
- //X: Yazılımsal al baştan yap.



Şekil 6.5 : Gerçek sistem

6.4.1 Yer İstasyonu ve Modem Yapısı

Gerçek sistemde uydu ile dünya arasındaki iletişimi uydu üzerindeki MHX-425 modemi sağlamaktadır. Benzetim sisteminde ise yer istasyonunu dizüstü bilgisayar, modemi ise MSP430 ile benzetim yapılmıştır. Şekil 6.6 da yer istasyon programının kullanıcı ara yüzü verilmiştir.



Şekil 6.6 : Yer istasyonu kullanıcı ara yüzü.

Şekil 6.6 da gösterildiği üzere üst kısımda, işlenen komutlar ekrana gelmekte ve uydunun durumları belirtilmektedir. Alt kısmın sol üst bölümünde seri iletişim için gerekli ayarlamalar yapılmakta ve bu ayarlar gösterilmektedir. Sağ üst kısımda ise uyduyu al baştan yapma, otomatik alarm durumu ve komut ekranını temizleme sekmeleri gözükmektedir. Sol alttaki 2 küme kamera kontrol sekmeleridir. Sağ

taraftaki iki küme ise güç sisteminden veri okumak ve göndermek için kullanılan sekmelerdir.

Uydu benzetim sistemi çalışmaya başladığı anda tüm alt sistemlerini tek tek kontrol ederek çalıştırır. Bu bilgiyi de hemen dizüstü bilgisayara RS 232 veri yolu ile iletir. Sistem, çalıştığı ve dizüstü bilgisayardan aksi komut gelmediği sürece 4 adet sensör durum bilgisini sürekli olarak dizüstü bilgisayarına gönderir. Yer istasyonu programındaki otomatik alarm durumda sürekli okunan sensörden birinde arıza meydana geldiğinde Şekil 6.6 deki ekranın üstünde gösterilir. Kamera komutları yer istasyonundan MSP 430' a yollandığında cevap ilgili mikroişlemciye bağlı ledlerde görüldüğü gibi aynı komutun yer istasyonuna geri yollanmasıyla komutun algılandığı gösterilmektedir. Yer istasyonu program kodları Ek d de belirtilmiştir.

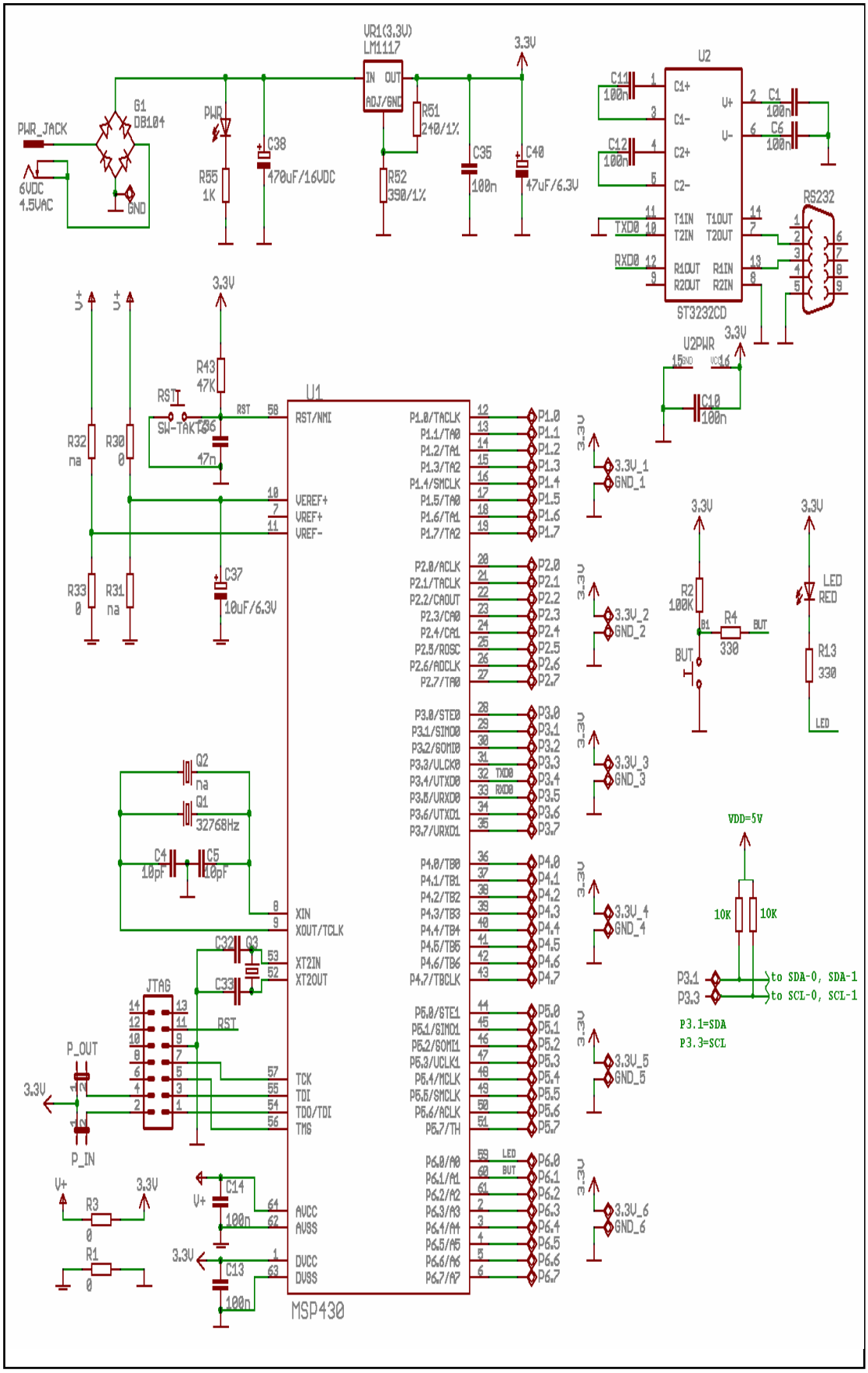
6.4.2 MSP 430 Uçuş Bilgisayarı

Benzetim sisteminde kullanılan mikroişlemci ile gerçek sistemde kullanılacak mikro işlemci aynıdır. MSP 430, I²C veri yolunu kullanarak kamera ve güç sistemi ile haberleşmektedir. I²C veri yolu haberleşmesinde MPS 430 yönetici konumunda, kamera ve güç sistemi ise yönetilen konumundadır. Uçuş bilgisayarı ile yer istasyonu arasındaki bağlantı ise bir önceki bölümde gösterildiği gibi RS 232 ile yapılmaktadır. Şekil 6.7 de MSP 430' un devre şeması verilmiştir.

MSP 430'un P3.1 ve P3.3 bacakları I²C iletişimi için kullanılmıştır. P3.1 seri veri yolu ucu, P3.3 ise iletişim saat üretecinin çıkışıdır. P3.6 ve P3.7 uçları ise RS 232 seri iletişim uçlarıdır.

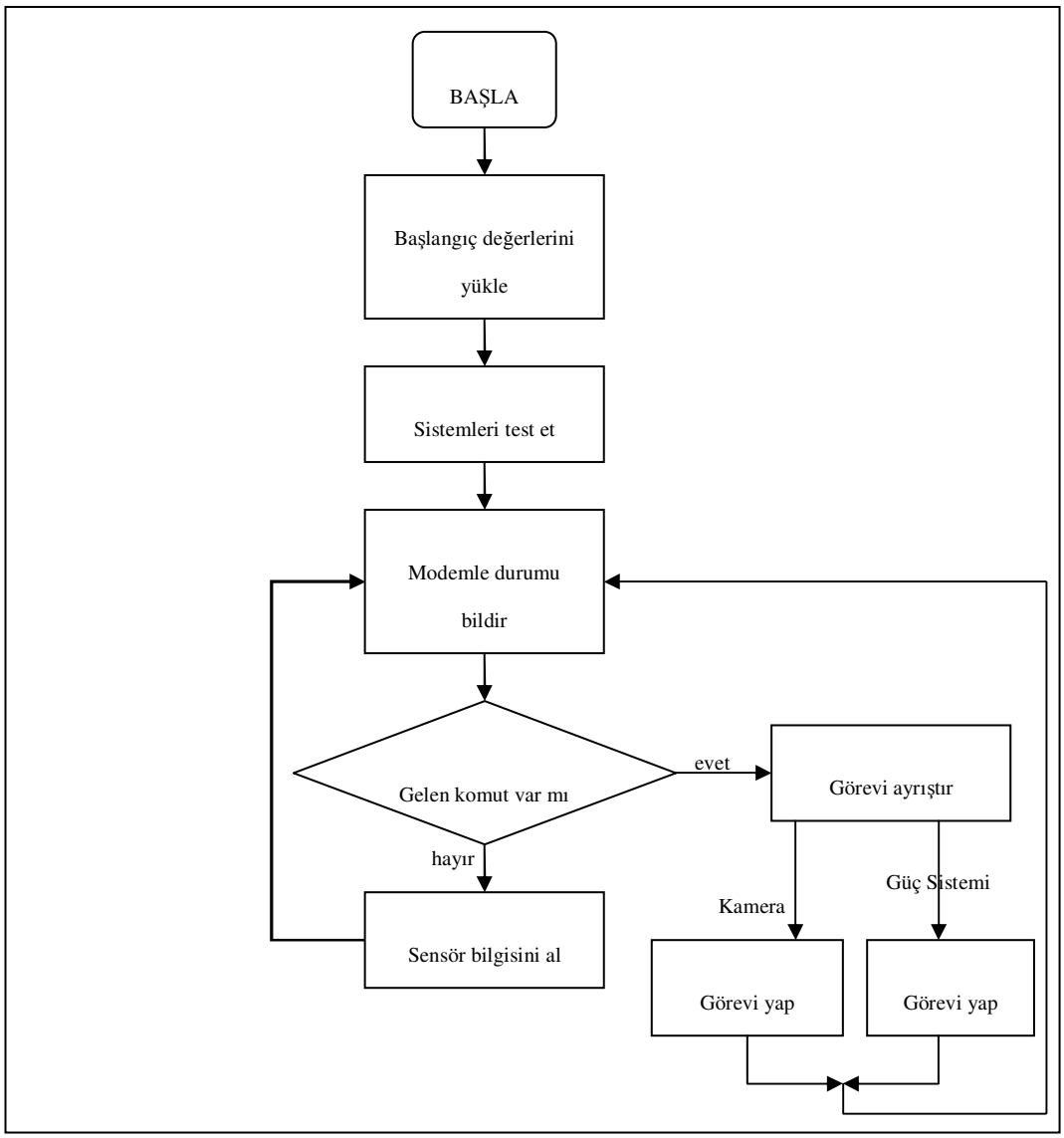
Mikro işlemci Salvo işletim sistemi ile programlanmıştır. Kullanılan dil ise Standart C dilidir. Yazılan program CrossWorks adlı derleyici kullanılarak düzenlenmiş ve yine aynı derleyici ile işlemciye yüklenmiştir.

MSP 430 5V doğru akım ile beslenmektedir. Ancak 3.3V beslemede sistemin çalışması için yeterli olabilmektedir.



Şekil 6.7 : MSP 430 devre şeması [7].

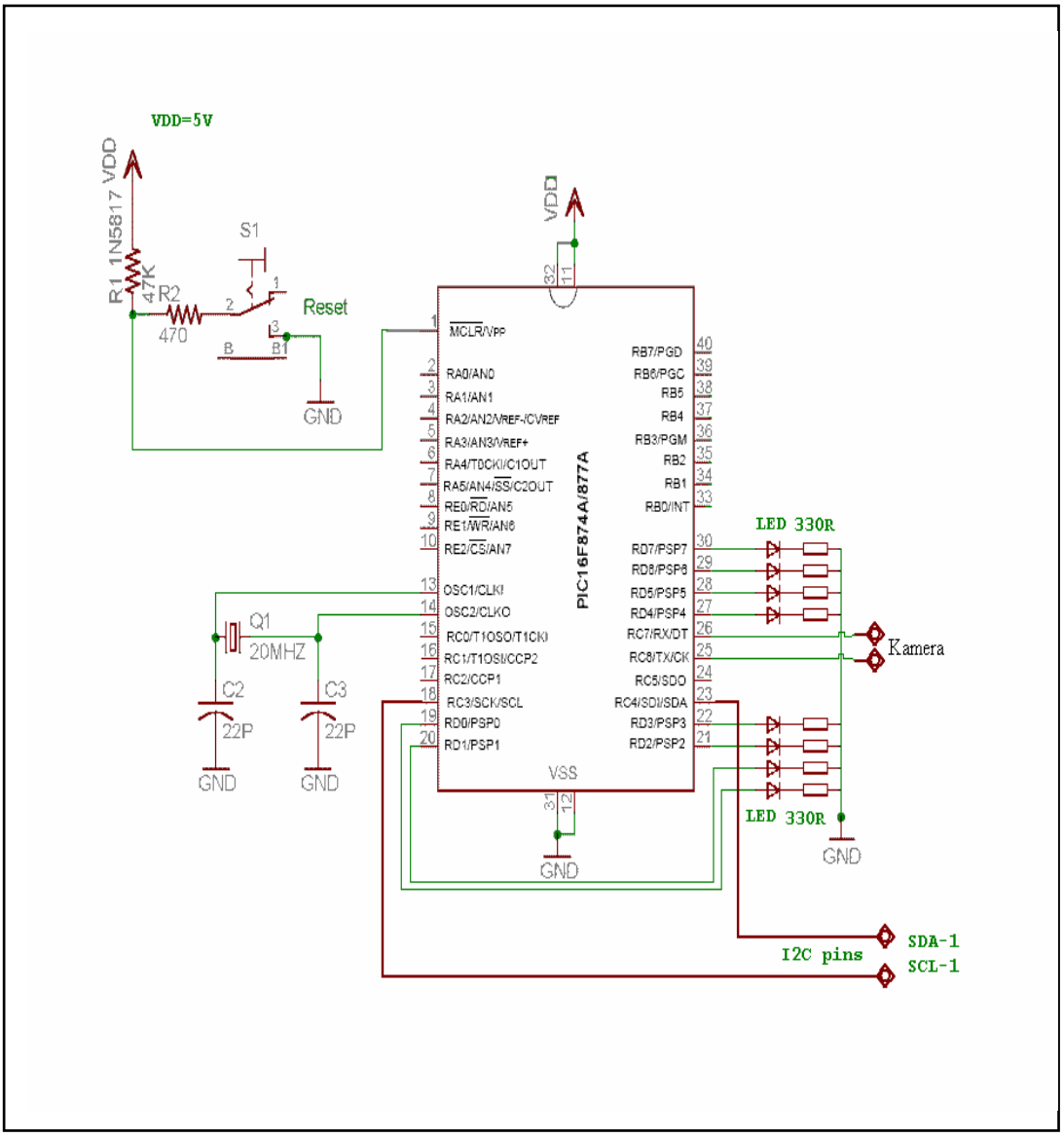
İşlemcinin içine yüklenmiş programda 3 görev koşmaktadır. Birincisi ve en öncelikli olanı yer istasyonu ile haberleşmeyi sağlayan iletişim görevidir. Bu görev yer istasyonundan herhangi bir komut geldiği takdirde bunu ilgili kısımlara iletir ve aynı şekilde görevin icra edildiğini ilgili kısımlardan öğrenerek yer istasyonuna iletir. Bunun dışında sürekli olarak sensör durum bilgisini yer istasyonuna iletir. İkincisi kamera görevidir. Bu görev, kamera benzetim sistemi ile iletişime geçip ilgili komutları iletir ve cevabını alıp iletişim görevine bildirir. Üçüncüsü ise güç sistemi görevidir. Bu görevde ilgili komutları alıp güç benzetim sistemine iletir ve cevabını iletişim sistemine yönlendirir. Dönüş cevabı yer istasyonun gönderdiği komutun aynısı olarak ayarlanmıştır. İkinci ve üçüncü görevler I²C veri yolunu kullanırlar. Şekil 6.8 de programın genel akış çizeneği verilmiştir.



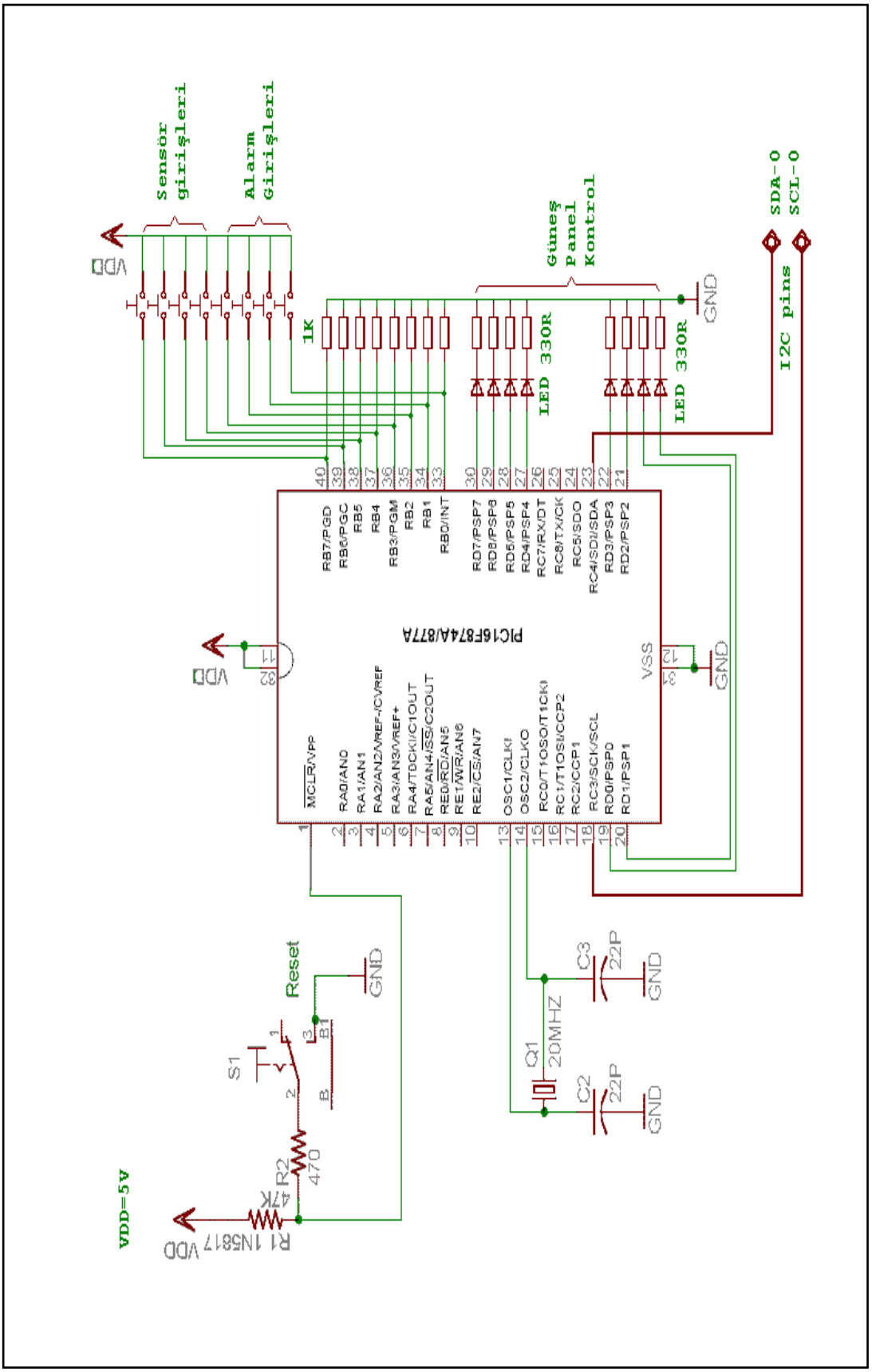
Şekil 6.8 : Benzetim sisteminin akış çizeneği

6.4.3 Kamera ve Güç Sistemlerinin Benzetimi

İki sistemde de işlemci olarak PIC 16F877 mikro işlemcileri kullanılmıştır. Her iki sistemde I²C ile yönetilen olarak MSP 430' a bağlanmıştır. Kamera sisteminde mikro işlemcinin C giriş/çıkışı çıkış olarak ayarlanmıştır. Bu çıkışlara bağlanan ledler vasıtasıyla yapılması istenen komutun durumu gösterilmektedir. Güç sisteminin işlemcisinde de aynı giriş/çıkış birimi çıkış olarak kullanılmıştır. Ayrıca güç sisteminde mikro işlemcinin B giriş/çıkışı giriş olarak ayarlanmıştır. İlk 4 bit alarm sensörlerinin, diğer 4 bit ise farklı sensör durumlarının girişi için atanmıştır. Şekil 6.9 da kamera sisteminin devre şeması, Şekil 6.10 da güç sisteminin devre şeması gösterilmiştir.



Şekil 6.9 : Kamera sisteminin devre şeması



Şekil 6.10 : Güç sisteminin devre şeması

7. SONUÇ

Görevlerin belirlenmesi, benzetim sisteminin oluşturulması ve Salvo işletim sisteminin yapısı araştırılmış, ana uçuş bilgisayarının içine yüklenmesi ile hedeflenen duruma gelinmiştir. Çalışma sonunda ulaşılan sonuçlar aşağıda sunulmuştur:

- GZİS açık kodlu olmadığı ve gerek Salvo işletim sistemi, gerek derleyici ücretsiz sürümler olduğu için ilk uygulamalarda programların derlenmesinde ve çalıştırılmasında problemler yaşanmıştır.

- Yukarıda bahsedilen problemlerin nedenlerinden biri de işletim sistemi ile derleyici arasındaki uyuşmazlık ve açıklayıcı doküman azlığıdır.

- İlk denemeler olabildiğince basit seviyede tutulup, olumlu sonuçlar alındıkça Salvo işletim sistemi daha etkin şekilde kullanılmaya yönelinmiştir.

- Normalde bu tarz bir sistemin görev yapısı çok karmaşık olması gerektiği akıldan hiç çıkarılmamıştır. Ancak projenin ilk kez gerçekleştirilmesinden ötürü sistem olabildiğince basit tutulmuştur.

- Tez uygulama ağırlıklı olduğu için çabaların büyük bölümü program ve mikro işlemci yapılarını öğrenip uygulamaların gerçekleştirilmesine sarf edilmiştir.

- Ana uçuş bilgisayarı MSP 430 üzerindeki işletim sistemi ve bunun içinde bulunan görev entegrasyon programı Ek C de verilmiştir. PIC 16F877 işlemcilerinin içindeki programlar ise veri yolundan gelen değeri çıkışa verdiği ve güç sisteminde giriş değerlerini alıp veri yoluna aktardığı için belirtilmemiştir.

8. KAYNAKLAR

[1] http://www.cubesatkit.com

[2] http://www.micrium.com

[3] http://www.embedded.com

[4] Salvo User Manual version 3.2.2

[5] **Palmintier B.** A Distributed Computing Architecture for Small Satellite and Multi-Spacecraft Missions. 16th Annual AIUU/USU Conference on Small Satellite.

[6] **Can Kurtulus, Taskın Baltacı, Murat Ulusoy, B. Tayfun Aydın, Bulent Tutkun Gokhan Inalhan, N.L. Oksan Cetiner-Yıldırım, Turgut Berat Karyot, Cuma Yarım, Fırat Oğuz Edis, Cingiz Hacıyev, A. Rustem Aslan, M. Fevzi Unal** ITU- pSAT I: Istanbul Technical University, Student Pico-Satellite Program. İstanbul Turkey.

[7] http://www.olimex.com

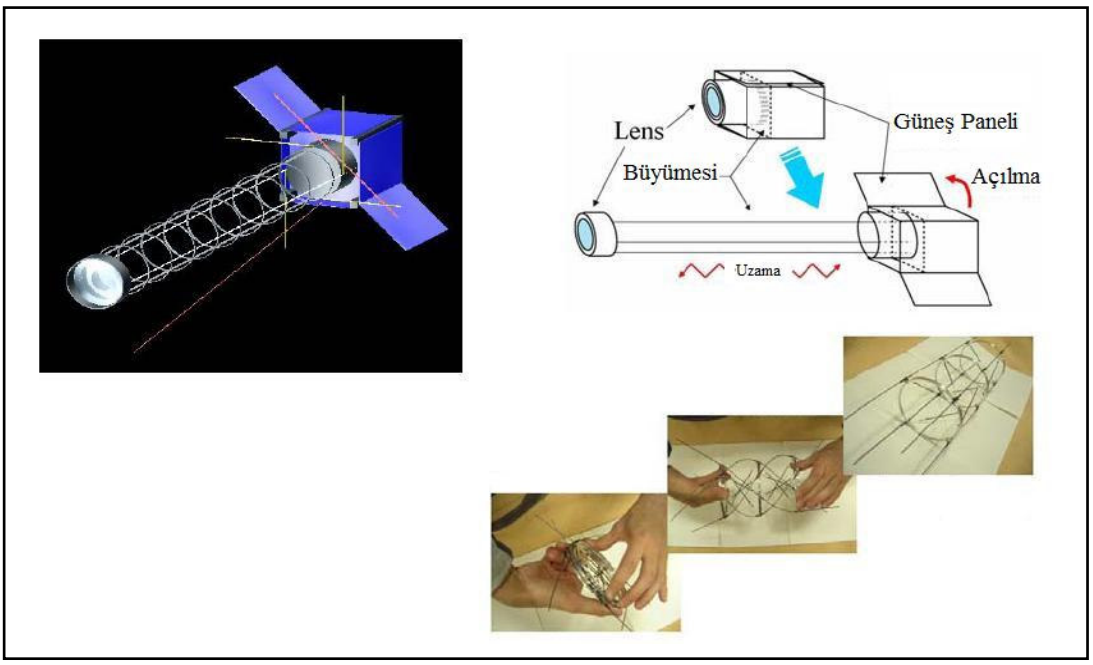
[8] **Akito Enokuchi, Masaki Nagai, Ryu Funase, Yuya Nakamura and Shinichi Nakasuka** Remote Sensing by University of Tokyo' s Pico-Satellite Project "PRISM".

[9] **L. N. Stras, D. D. Kekez, G. J. Wells, T. Jeans, R. E. Zee, F. M. Pranajaya, and D. G. Foisy,** "The Design and Operation of The Canadian Advanced Nanospace eXperiment (CanX-1)," *Proc. AMSAT-NA 21st Space Symposium*, Toronto, Canada, October 2003, pp.150-160**.**

[10] http://www.studentspace.aau.dk/publications/AAUCubesatProject.pdf

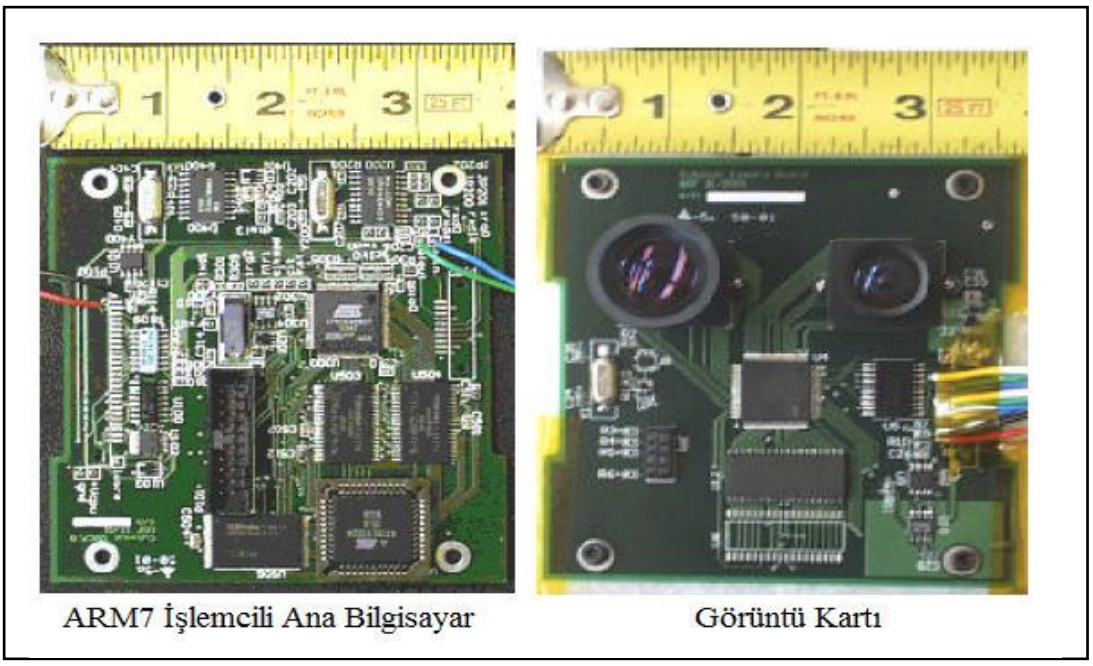
Yapılan Bazı Uygulamalar:

Tokyo üniversitesindeki projede [8], 17cm x 17cm x 25 cm ebatlarında 5 kg ağırlığında yüksek çözünürlüklü kameraya sahip bir piko uydu yapılmıştır. İlk uydu projelerinde başarı sağlandıkça amaç daha yüksek çözünürlüklü görüntü sağlayabilmek olmuş ve bunu başarmak için teleskopik mercek kullanılmıştır. Şekil A.1 de bu yapı teleskopik yapı gösterilmiştir. Kısalmış boyu 10cm iken uzatıldığında mercek 80cm ileri çıkmaktadır.



Şekil A.1 : Uzayan lens uygulaması [8].

Kanada üniversitesindeki CanX-1 projesinde [9], 1 kg ağırlığa sahip piko uydu 30 haziran 2003 tarihinde fırlatılmıştır. Amatör uydu frekansı üzerinden uydu ile iletişim sağlanmıştır. Üzerinde 1 adet yatay 1 adet yıldız izleyicisi olmak üzere 2 adet görüntüleme cihazı bulunmaktadır. 3 eksenli aktif manyetik dengeleyicisi, yerel konum sistemi tabanlı yer belirleyicisi ARM7 merkezi işlemcisi vardır. İşlemcinin çalışma hızı 40 MHZ olup 32MB harici belleği bulunmaktadır. Şekil A.2 de ana bilgisayar ve görüntüleme kartları görülmektedir.



Şekil A.2 : CanX-1 uydusunun bilgisayar ve görüntü kartları

Aalborg üniversitesinin projesinde [10], 10cm x 10cm x 10cm boyutlarında 1 kg ağırlığında piko uydunun yapımına 2001 senesinde başlanmış. 30 haziran 2003 yılında Rusya tarafından uzaya fırlatılmıştır. Fırlatma maliyeti 40.000 Amerikan doları tutmuştur. Projede güdülen amaç mühendislerin konu ile ilgili yeteneklerini geliştirmek, uydudan sinyal almayı başarmak, uzay ortamından veri toplayabilmek ve kamera kullanarak görüntü elde etmektir. Siemens C161 işlemcisi uçuş bilgisayarı olarak kullanılmıştır. Bunun seçim sebebi ise düşük güç tüketimidir. Proje piko uydu deneyiminde başlangıç olarak görülmektedir.

EK B

Muhtelif GZİS' leri, üretici firmaları ve internet adresleri listesi.

- AMX (KADAK) http://www.kadak.com/
- AvSYS Real-Time (Avocet Systems) http://www.avocetsystems.com/
- Blackhawk OS (Blackhawk) http://www.blackhawk-dsp.com/
- BlueCat Linux (LynuxWorks) http://www.lynuxworks.com/
- BSD/OS (Wind River) http://www.windriver.com/
- C Executive (JMI Software) http://www.jmi.com/
- CMX-RTX, CMX-Tiny+, CMX-RTXS (CMX Systems) http://www.cmx.com/
- Diamond (3L) http://www.shen.myby.co.uk/threel
- DR-DOS 7.03 (The SCO Group) http://www.sco.com/
- eCos (Red Hat) http://www.redhat.com/
- Embedix RT (Lineo) http://www.lineo.com/
- embOS (SEGGER) http://www.segger.com/
- ERCOSEK (ETAS) http://www.etasinc.com/
- eRTOS (JK Microsystems) http://www.jkmicro.com/
- EUROS (EUROS Embedded Systems) http://www.kaneff.de/
- Eyrx (Eyring) http://www.eyring.com/
- Fusion RTOS (DSP OS) http://www.dspos.com/
- icWORKSHOP (Integrated Chipware) http://www.chipware.com/
- INTEGRITY (Green Hills Software) http://www.ghs.com/

- iRMX III, iRMX/INtime for Windows (TenAsys) http://www.tenasys.com/
- Jbed (esmertec) http://www.esmertec.com/
- Linux for Real-Time (OnCore Systems) http://www.oncoresystems.com/
- LynxOS (LynuxWorks) http://www.lynuxworks.com/
- mC/OS-II (Micrium) http://www.micrium.com/
- Microwave OS-9 (RadiSys) http://www.radisys.com/
- MontaVista Linux (MontaVista Software) http://www.mvista.com/
- NetBSD Embedded (Wasabi Systems) http://www.wasabisystems.com/
- Neutrino (QNX Software Systems) http://www.qnx.com/
- Nucleus uiPLUS, Nucleus OSEK, Nucleus PLUS (Accelerated Technology/Mentor Graphics) http://www.acceleratedtechnology.com/
- On Time RTOS-32 (On Time Software) http://www.on-time.com/
- OnCore OS (OnCore Systems) http://www.oncoresystems.com/
- OSE RTOS (OSE Systems) http://www.ose.com/
- OSEKturbo (Metrowerks/Motorola) http://www.metrowerks.com/
- OSEKWorks (Wind River) http://www.windriver.com/
- PDOS (Eyring) http://www.eyring.com/
- pF/x (FORTH) http://www.forth.com/
- PharLap Real-time ETS Kernel (VenturCom) http://www.vci.com/
- pmDOS (Micro Digital) http://www.smxinfo.com/
- Precise/MQX (ARC International) http://www.arc.com/
- PSMX Portable smx (Micro Digital) http://www.smxinfo.com/
- pSOSystem 2.5/3 (Wind River) http://www.windriver.com/
- PSX (JMI Software Systems) http://www.jmi.com/

- PXROS (HighTec EDV-Systeme) http://www.hightec-rt.com/
- QNX (QNX Software Systems) http://www.qnx.com/
- Quadros (RTXC) http://www.quadros.com/
- QuickTask (Softools) http://www.softools.com/
- RAVEN (Aonix) http://www.aonix.com/
- REAL/IX PX (MODCOMP) http://www.modcomp.com/
- Realogy Real-Time Architect (LiveDevices) http://www.livedevices.com/
- REALOS (Fujitsu Microelectronics) http://www.fujitsumicro.com/
- Real-Time OS: DSP/BIOS (Texas Instruments) http://www.ti.com/
- Red Hat Embedded (Red Hat) http://www.redhat.com/
- REDICE-Linux (REDSonic) http://www.redsonic.com/
- ROM-DOS (Datalight) http://www.datalight.com/
- RTexec (Applied Dynamics International) http://www.adi.com/
- RTEMS (OAR) http://www.rtems.com/
- RTKernel (On Time Software) http://www.on-time.com/
- RTKernel-RISC (EBSnet) http://www.ebsnetinc.com/
- RTX for Windows (VenturCom) http://www.vci.com/
- RTX51/RTX51 Tiny, RTX166/RTX166 Tiny (Keil) http://www.keil.com/
- Salvo (Pumpkin) http://www.pumpkininc.com/
- SKYmpx (SKY Computers) http://www.skycomputers.com/
- smx/smx++ (Micro Digital) http://www.smxinfo.com/
- Spartos (Ardro Engineering) http://www.ardro.com/
- Starlight Linux (Auriga) http://www.auriga.com/

- SuperTask (Lantronix) http://www.lantronix.com/
- TargetOS (Blunk Microsystems) http://www.blunkmicro.com/
- ThreadX (Express Logic) http://www.expresslogic.com/
- Tics (TICS Realtime) http://www.cris.com/~Tics/
- TimeSys Linux/RT, Real-Time Mach (TimeSys) http://www.timesys.com/
- TronTask3.0 (Lantronix) http://www.lantronix.com/
- TTPos (TTTech Computertechnik) http://www.tttech.com/
- TurboTask (Softools) http://www.softools.com/
- TxOS - Titanic (Incantation Systems) http://www.incantationsystems.com/
- VRTX (Mentor Graphics) http://www.mentor.com/
- VSPWorks (Wind River) http://www.windriver.com/
- VxWorks, VxWorks AE (Wind River) http://www.windriver.com/
- Windows CE .NET, (Microsoft) http://microsoft.com/

Aşağıda MPS 430 için yazılmış Salvo işletim sisteminin kodları mevcuttur.

```
#include <__cross_studio_io.h>
#include  <msp430x16x.h>
#include <string.h>
#include <stdlib.h>
#include "main.h"
#include "salvo.h"
#include "msp430.h"
#define TASK_COUNT_P      OSTCBP(1) /* task #1        */
#define TASK_SHOW_P       OSTCBP(2) /*  ""  #2        */
#define TASK_BLINK_P      OSTCBP(3) /*  ""  #3        */
#define PRIO_COUNT        10        /* task priorities*/
#define PRIO_SHOW         10        /*  ""            */
#define PRIO_BLINK        2         /*  ""            */
#define BINSEM_UPDATE_PORT_P OSECBP(1) /* binSem #1    */
unsigned int counter;
_OSLabel(TaskA1)
_OSLabel(TaskB1)
_OSLabel(TaskC1)
int RX_BufLen;//RX bufferdaki bilginin uzunlugunu say.
char RX_Buffer[255];
char OK[6]={13,10,79,75,13,10};
```

```
char Error[10]={13,10,69,82,82,79,82,33,13,10};
unsigned const Gecikme=50000;
unsigned const SerialGecikme=1000;
char OperationType;
int OperasyonT=1;
int OperasyonR=1;
int OperasyonW=1;
int OperasyonD=1;
int OperasyonS=1;
int OperasyonX=1;
int Operasyon1=1;
int Operasyon2=1;
int Operasyon3=1;
int Operasyon4=1;
int Operasyon5=1;
int Operasyon6=1;
int Operasyon7=1;
int Operasyon8=1;
int Operasyon9=1;//alarm
int Resetaktifmi=1;
int Reset=0;
int a;
//T=54, R=52, W=57, D=44, S=53, X=58
unsigned char i2cRXbuffer[];
unsigned char i2cTXbuffer[];
void TaskA( void )
{
```

```
for (;;) {
C_Delay(1000);
OperationType= RXBUF1;//seri portu oku
OSSignalBinSem(BINSEM_UPDATE_PORT_P);
OS_Yield(TaskA1);
    }
}
void TaskB( void )
{
for (;;) {
OS_WaitBinSem(BINSEM_UPDATE_PORT_P, OSNO_TIMEOUT, TaskB1);
P6OUT ^=0x01;
Slave1();
C_Delay(10000);
OS_Yield(TaskB1);
    }
}
void TaskC( void )
{
for (;;) {
P6OUT ^=0x01;
Slave2();
C_Delay(10000);
OS_Yield(TaskC1);
    }
}
void main( void )
```

```
{
ResetSystem:
  WDTCTL = WDTPW | WDTHOLD;                 // Disable the Watchdog.
  P6DIR |=0x01;
  RS232init();
strcpy(RX_Buffer,"Communication Init...");
RS_232_Send(RX_Buffer);
SendOK();
C_Delay(Gecikme);
  Init();
strcpy(RX_Buffer,"Processor init...");
RS_232_Send(RX_Buffer);
SendOK();
C_Delay(Gecikme);
  OSInit();
strcpy(RX_Buffer,"OS init...");
RS_232_Send(RX_Buffer);
SendOK();
C_Delay(Gecikme);
    OSCreateTask(TaskA, OSTCBP(1), 10);
    OSCreateTask(TaskB,  OSTCBP(2), 10);
    OSCreateTask(TaskC,  OSTCBP(3), 10);
    OSEi();
OSCreateBinSem(BINSEM_UPDATE_PORT_P, 0);
CheckSystem();
 for (;;)
    {
```

```
        if (Reset==1)
        {
        Reset=0;
        goto ResetSystem;
         }
      OSSched();
   }
}
//*********************************************************************
**********
void i2c_init(unsigned int Adres)
{
 P3SEL |= 0x0A;                    // Assign I2C pins to module
 U0CTL |= I2C + SYNC;               // Switch USART0 to I2C mode
 U0CTL &= ~I2CEN;                   // Recommended init procedure
 I2CTCTL = I2CSSEL_2;                // SMCLK
 I2CNDAT = 0x02+1;                  // Transmit Three byte
 I2CSA = Adres>>1;                  // Slave address
 U0CTL |= I2CEN;                   // Enable I2C, 7 bit addr,
}
void RS232init(void)
{
//------USART 0 -------
//  P3SEL |= 0x30;                  // P3.4,5 = USART0 TXD/RXD
//  ME1 |= UTXE0 + URXE0;               // Enable USART0 TXD/RXD
//  UCTL0 |= CHAR;                  // 8-bit character
//  UTCTL0 |= SSEL0;                 // UCLK = ACLK
```

```
//  UBR00 = 0x03;                       // 32k/9600 - 3.33  //UDEA ve MHX icin
degerler degisiyor. MHX'e ayarli, UDEA 2400 istiyor

//  UBR10 = 0x00;                       //

//  UMCTL0 = 0x4A;                        // Modulation     //UDEA'da 6B MHX'de 4A

//  UCTL0 &= ~SWRST;                        // Initialize USART state machine

////  IE1 |= URXIE0;                       // Enable USART0 RX interrupt

//---------USART 1--------------------

P3SEL |= 0xC0;//P3.6,7

ME2 |= UTXE1 + URXE1; // Enable USART1 TXD/RXD

UCTL1 |= CHAR; // 8-bit character

UTCTL1 |= SSEL0; // UCLK = ACLK

UBR01 = 0x03; // 32k/2400 - 13.65

UBR11 = 0x00;

UMCTL1 = 0x4A; // Modulation

UCTL1 &= ~SWRST; // Initialize USART state machine

}

//*****************************************************************
***********

void i2cRead (unsigned int Adres)

{

unsigned int i;

i=0;

i2c_init(Adres);

  U0CTL |= MST;                       // Master mode

  I2CTCTL = I2CSTT+I2CSTP;                // Initiate transfer

  while ((I2CIFG & RXRDYIFG) == 0);         // Wait for Receiver to be ready

   i2cRXbuffer[i] = I2CDRB;               //ilk gelen bilgiyle ilgilenme
```

```
for (i=0;i<0x02;i++)                    //ikinci gelen bilgi veriuzunlugu
  {
   while ((I2CIFG & RXRDYIFG) == 0);         // Wait for Receiver to be ready
   i2cRXbuffer[i] = I2CDRB;                  //  verigeliyor
  }
  while ((I2CTCTL & I2CSTP) == 0x02);       // Wait for Stop Condition


}
void i2cWrite (unsigned int Adres)
{
unsigned int i;
i=0;
i2c_init(Adres);
  U0CTL |= MST;                             // Master mode
  I2CTCTL |= I2CSTT+I2CSTP+I2CTRX;          // Initiate transfer
  while ((I2CIFG & TXRDYIFG) == 0);         // Wait for transmitter to be ready
  I2CDRB = 0x00;                            // Load  Control Byte
for (i=0;i<0x02;i++)                    //ikinci gelen bilgi veriuzunlugu
{
  while ((I2CIFG & TXRDYIFG) == 0);         // Wait for transmitter to be ready
   I2CDRB =i2cTXbuffer[i];                  //  verigeliyor
}
  while ((I2CTCTL & I2CSTP) == 0x02);       // Wait for Stop Condition


}
void Slave1(void)
{
```

```
switch (OperationType)
 {
   case 0:{//T=54
                if (OperasyonT==0)
                {
                strcpy(RX_Buffer,"Bir kare al ve PC'ye transfer et=");
                RS_232_Send(RX_Buffer);
                OperasyonT=1;
                i2cTXbuffer[0]=0x54;
                icraet(0x38);
                }
         break; }
   case 1:{// R=52 ASCII 82
                if (OperasyonR==0)
                {
                strcpy(RX_Buffer,"Register oku, R'den hemen sonra gelen byte offset'i
temsil eder=");
                RS_232_Send(RX_Buffer);
                OperasyonR=1;
                i2cTXbuffer[0]=0x52;
                icraet(0x38);
                }
         break; }
   case 2:{//W=57
                if (OperasyonW==0)
                {
                strcpy(RX_Buffer,"Register'a yaz, W'dan hemen sonra gelen byte offset
arkasindan gelen byte ise yazilacak degerdir=");
```

```
                RS_232_Send(RX_Buffer);
                OperasyonW=1;
                i2cTXbuffer[0]=0x57;
                icraet(0x38);
                }
        break; }
    case 3:{// D=44
                if (OperasyonD==0)
                {
                strcpy(RX_Buffer,"Kamerayi ilk ayarlandigi hale sokar - 8 bit prog.
RGB QVGA=");
                RS_232_Send(RX_Buffer);
                OperasyonD=1;
                i2cTXbuffer[0]=0x44;
                icraet(0x38);
                }
        break; }
    case 4:{// S=53
                if (OperasyonS==0)
                {
                strcpy(RX_Buffer,"Manufacture IDl=");
                RS_232_Send(RX_Buffer);
                OperasyonS=1;
                i2cTXbuffer[0]=0x53;
                icraet(0x38);
                }
        break; }
    case 5:{// X=58
```

```
            if (OperasyonX==0)
            {
            strcpy(RX_Buffer,"Software reset atar=");
            RS_232_Send(RX_Buffer);
            OperasyonX=1;
            i2cTXbuffer[0]=0x58;
            icraet(0x38);
            }
      break; }
case 254:{// Rerset System
            if (Resetaktifmi==0)
            {
            Reset=1;
            Resetaktifmi=1;
            }
            else
            Reset=0;

      break; }

case 0xFF:{//komutgondericem hazir ol
            OperasyonT=0;
            OperasyonR=0;
            OperasyonW=0;
            OperasyonD=0;
            OperasyonS=0;
            OperasyonX=0;
```

```
                Resetaktifmi=0;
//                strcpy(RX_Buffer,"Ready");
//                RS_232_Send(RX_Buffer);
//                SendOK();
          break; }
    default :{

//                strcpy(RX_Buffer,"Access denied!");
//                RS_232_Send(RX_Buffer);
//                SendOK();
         break;}

  }
}
void Slave2(void)
{
switch (OperationType)
 {
   case 6:{// 31
                if (Operasyon1==0)
                {
                strcpy(RX_Buffer,"Sensor1_Value=");
                RS_232_Send(RX_Buffer);
                Operasyon1=1;
                i2cTXbuffer[0]=6;
                icraet(0x48);
                }
```

```
      break; }
case 7:{// 32
            if (Operasyon2==0)
            {
            strcpy(RX_Buffer,"Sensor2_Value=");
            RS_232_Send(RX_Buffer);
            Operasyon2=1;
            i2cTXbuffer[0]=7;
            icraet(0x48);
            }
      break; }
case 8:{// 33
            if (Operasyon3==0)
            {
            strcpy(RX_Buffer,"Sensor3_Value=");
            RS_232_Send(RX_Buffer);
            Operasyon3=1;
            i2cTXbuffer[0]=8;
            icraet(0x48);
            }
      break; }
case 9:{//34
            if (Operasyon4==0)
            {
            strcpy(RX_Buffer,"Sensor4_Value=");
            RS_232_Send(RX_Buffer);
            Operasyon4=1;
```

```
                i2cTXbuffer[0]=9;
                icraet(0x48);
                }
        break; }

case 10:{//34
                if (Operasyon5==0)
                {
                strcpy(RX_Buffer,"komut1=");
                RS_232_Send(RX_Buffer);
                Operasyon5=1;
                i2cTXbuffer[0]=10;
                icraet(0x48);
                }
        break; }
case 11:{//34
                if (Operasyon6==0)
                {
                strcpy(RX_Buffer,"komut2=");
                RS_232_Send(RX_Buffer);
                Operasyon6=1;
                i2cTXbuffer[0]=11;
                icraet(0x48);
                }
        break; }
case 12:{//34
                if (Operasyon7==0)
```

```
                {
                strcpy(RX_Buffer,"komut3=");
                RS_232_Send(RX_Buffer);
                Operasyon7=1;
                i2cTXbuffer[0]=12;
                icraet(0x48);
                }
        break; }
case 13:{
                if (Operasyon8==0)
                {
                strcpy(RX_Buffer,"komut4=");
                RS_232_Send(RX_Buffer);
                Operasyon8=1;
                i2cTXbuffer[0]=13;
                icraet(0x48);
                }
        break; }
case 14:{
                if (Operasyon9==0)
                {
                Operasyon9=1;
                Alarm_kontrol();
                }
        break; }
case 0xFF:{//komutgondericem hazir ol
              Operasyon1=0;
```

```
                Operasyon2=0;
                Operasyon3=0;
                Operasyon4=0;
               Operasyon5=0;
               Operasyon6=0;
               Operasyon7=0;
               Operasyon8=0;
               Operasyon9=0;
//                  strcpy(RX_Buffer,"Ready");
//                  RS_232_Send(RX_Buffer);
//                  SendOK();
          break; }
    default :{
//                  strcpy(RX_Buffer,"Access denied!");
//                  RS_232_Send(RX_Buffer);
         break;}


  }
}
void icraet(unsigned int adres)
{
char Tempbuf[10];
int i;
i2cWrite(adres);
C_Delay(100);
i2cRead(adres);
C_Delay(100);
```

```
//Tempbuf[0]=i2cRXbuffer[0];

i=(int)i2cRXbuffer[0];

itoa(i,Tempbuf,10);

RS_232_Send(Tempbuf);

SendOK();

}

void RS_232_Send(char *data)

{

     unsigned int i, j;

       for(i = 0; i < strlen(data); i++)//diziyi karakte karater TX buffera yükle.

       {//---------------------------------

          while (!(IFG2 & UTXIFG1));// USART1 TX buffer ready?

          TXBUF1 = data[i];// TX buffera bilgi yolla.

          C_Delay(SerialGecikme);

        }//---------------------------------

TXBUF1=0x00;


          for(j = 0; j <255; j++)//Bufer temizle.

          RX_Buffer[j]='\0';//bufer Null ile doldur.


          RX_BufLen=0;//sayaci sifirla cunki yeni RXbuferdaki bilgi uzunlugunu 0 dan
baslayarak saymak icin.

}

//*****************************************************************************
***********

void SendOK(void)

{

 unsigned int i;
```

```
      for(i = 0; i < 6; i++)//diziyi karakte karater TX buffera yükle.
      {//--------------------------------
         while (!(IFG2 & UTXIFG1));// USART1 TX buffer ready?
         TXBUF1 = OK[i];// TX buffera bilgi yolla.
         C_Delay(SerialGecikme);
       }//--------------------------------

}
//*********************************************************************
*****
//*********************************************************************
**********
void SendErr(void)
{
 unsigned int i;

      for(i = 0; i < 10; i++)//diziyi karakte karater TX buffera yükle.
      {//--------------------------------
         while (!(IFG2 & UTXIFG1));// USART1 TX buffer ready?
         TXBUF1 = Error[i];// TX buffera bilgi yolla.
         C_Delay(SerialGecikme);
       }//--------------------------------

}
//*********************************************************************
*****
void C_Delay(unsigned int cycle)
{
```

```
while (cycle !=0)
cycle--;
}
//*****************************************************************************
***********

void CheckSystem(void)
{
strcpy(RX_Buffer,"Tasks init...");
RS_232_Send(RX_Buffer);
SendOK();
C_Delay(Gecikme);

strcpy(RX_Buffer,"System Ready");
RS_232_Send(RX_Buffer);
SendOK();
C_Delay(Gecikme);

i2cTXbuffer[0]=0;
i2cWrite(0x48);
i2cTXbuffer[0]=0;
i2cWrite(0x38);
strcpy(RX_Buffer,"Power Supply init..");
RS_232_Send(RX_Buffer);
SendOK();
C_Delay(Gecikme);
strcpy(RX_Buffer,"1. Panel Activating...");
RS_232_Send(RX_Buffer);
```

```
i2cTXbuffer[0]=1;
i2cWrite(0x48);
hatavarmi(0x48,1,RX_Buffer);
C_Delay(Gecikme);
strcpy(RX_Buffer,"2. Panel Activating...");
RS_232_Send(RX_Buffer);
i2cTXbuffer[0]=3;
i2cWrite(0x48);
hatavarmi(0x48,3,RX_Buffer);
C_Delay(Gecikme);
strcpy(RX_Buffer,"3. Panel Activating...");
RS_232_Send(RX_Buffer);
i2cTXbuffer[0]=7;
i2cWrite(0x48);
hatavarmi(0x48,7,RX_Buffer);
C_Delay(Gecikme);
strcpy(RX_Buffer,"4. Panel Activating...");
RS_232_Send(RX_Buffer);
i2cTXbuffer[0]=15;
i2cWrite(0x48);
hatavarmi(0x48,15,RX_Buffer);
C_Delay(Gecikme);
strcpy(RX_Buffer,"Sensors initial...");
RS_232_Send(RX_Buffer);
SendOK();
C_Delay(Gecikme);
```

```
strcpy(RX_Buffer,"1. Sensor Activating...");
RS_232_Send(RX_Buffer);
i2cTXbuffer[0]=31;
i2cWrite(0x48);
hatavarmi(0x48,31,RX_Buffer);
C_Delay(Gecikme);
strcpy(RX_Buffer,"2. Sensor Activating...");
RS_232_Send(RX_Buffer);
i2cTXbuffer[0]=63;
i2cWrite(0x48);
hatavarmi(0x48,63,RX_Buffer);
C_Delay(Gecikme);
strcpy(RX_Buffer,"3. Sensor Activating...");
RS_232_Send(RX_Buffer);
i2cTXbuffer[0]=127;
i2cWrite(0x48);
hatavarmi(0x48,127,RX_Buffer);
C_Delay(Gecikme);
strcpy(RX_Buffer,"4. Sensor Activating...");
RS_232_Send(RX_Buffer);
i2cTXbuffer[0]=255;
i2cWrite(0x48);
hatavarmi(0x48,255,RX_Buffer);
i2cTXbuffer[0]=1;
i2cWrite(0x38);
C_Delay(Gecikme);
strcpy(RX_Buffer,"Camera init...");
```

```
RS_232_Send(RX_Buffer);
SendOK();
C_Delay(Gecikme);
i2cTXbuffer[0]=3;
i2cWrite(0x38);
C_Delay(Gecikme);
i2cTXbuffer[0]=7;
i2cWrite(0x38);
C_Delay(Gecikme);
i2cTXbuffer[0]=15;
i2cWrite(0x38);
C_Delay(Gecikme);
i2cTXbuffer[0]=31;
i2cWrite(0x38);
C_Delay(Gecikme);
i2cTXbuffer[0]=63;
i2cWrite(0x38);
C_Delay(Gecikme);
i2cTXbuffer[0]=127;
i2cWrite(0x38);
C_Delay(Gecikme);
i2cTXbuffer[0]=255;
i2cWrite(0x38);
C_Delay(Gecikme);
strcpy(RX_Buffer,"Camera  Activating...");
RS_232_Send(RX_Buffer);
SendOK();
```

```
}
void hatavarmi(unsigned int adres,unsigned int value,unsigned char *Message)
{
  i2cRead(adres);
  C_Delay(100);
  if ((i2cRXbuffer[0]==value)|i2cRXbuffer[0]>8)//8 e kadar olanlar alarm.>8 olanlar
sensor degerleri
  {
    RS_232_Send(Message);
    SendOK();
  }
else if (i2cRXbuffer[0]!=value)
  {
    RS_232_Send(Message);
    SendErr();
  }
}
void Alarm_kontrol(void)
{
i2cTXbuffer[0]=255;
i2cWrite(0x48);
C_Delay(100);
  i2cRead(0x48);
C_Delay(100);
 switch (i2cRXbuffer[0])
 {
 case 1:{
```

```
   strcpy(RX_Buffer,"1. Panel Failure");
   RS_232_Send(RX_Buffer);
   SendErr();
 break; }
 case 2: {
   strcpy(RX_Buffer,"2. Panel Failure");
   RS_232_Send(RX_Buffer);
   SendErr();
 break;}
 case 4: {
   strcpy(RX_Buffer,"3. Panel Failure");
   RS_232_Send(RX_Buffer);
   SendErr();
 break;}
 case 8: {
   strcpy(RX_Buffer,"4. Panel Failure");
   RS_232_Send(RX_Buffer);
   SendErr();
 break;}
default : {
          strcpy(RX_Buffer,"Running Normally");
          RS_232_Send(RX_Buffer);
          SendOK();
        break; }
```

ÖZGEÇMİŞ

Burak Sağlam 1979 yılında İstanbul'da doğdu. İlk, orta ve lise öğrenimini İstanbul' da tamamladı. 2004 yılında İstanbul Teknik Üniversitesi, Uzay Mühendisliği bölümünü bitirmiştir. Halen İstanbul Teknik Üniversitesi Uçak ve Uzay Mühendisliği Yüksek Lisans Programı'na kayıtlıdır.