BAŞKENT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

GERÇEK ZAMANLI GÖREV ZAMANLAYICI METOTLARININ UZAY ARAÇLARI SİMÜLASYONLARI ÜZERİNDE KARŞILAŞTIRILMASI

MEHMET EMİN GÜLLÜOĞLU

YÜKSEK LİSANS TEZİ
2018

GERÇEK ZAMANLI GÖREV ZAMANLAYICI METOTLARININ UZAY ARAÇLARI SİMÜLASYONLARI ÜZERİNDE KARŞILAŞTIRILMASI

A COMPARISON OF REAL-TIME TASK SCHEDULING METHODS IN SPACECRAFT SIMULATION

MEHMET EMİN GÜLLÜOĞLU

Başkent Üniversitesi
Lisansüstü Eğitim Öğretim ve Sınav Yönetmeliğinin
BİLGİSAYAR Mühendisliği Anabilim Dalı İçin Öngördüğü
YÜKSEK LİSANS TEZİ
olarak hazırlanmıştır.

2018

“Gerçek Zamanlı Görev Zamanlayıcı Metotlarının Uzay Araçları Simülasyonları Üzerinde Karşılaştırılması” başlıklı bu çalışma, jürimiz tarafından, 10/08/2018 tarihinde, **BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI 'nda YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Başkan (Danışman) : Prof. Dr. Mehmet Reşit TOLUN

Üye : Dr. Öğr. Üyesi Emre SÜMER

Üye : Dr. Öğr. Üyesi Atilla ERGÜZEN

ONAY

..../08/2018

Prof. Dr. Faruk ELALDI

Fen Bilimleri Enstitüsü Müdürü

TEŞEKKÜR

Her zaman desteklerini esirgemeyen aileme…

Bu çalışmanın gerçekleştirilmesinde, değerli bilgilerini benimle paylaşan, her zaman yol gösteren saygıdeğer danışman hocam; Prof. Dr. Mehmet Reşit TOLUN'a, çalışmam boyunca benden yardımlarını esirgemeyen arkadaşlarım Murat ÖNAL ve Alper KIRAL'a sonsuz teşekkürlerimi sunarım.

ÖZ

GERÇEK ZAMANLI GÖREV ZAMANLAYICI METOTLARININ UZAY ARAÇLARI SİMÜLASYONLARI ÜZERİNDE KARŞILAŞTIRILMASI

Mehmet Emin GÜLLÜOĞLU

Başkent Üniversitesi Fen Bilimleri Enstitüsü

Bilgisayar Mühendisliği Anabilim Dalı

Günümüzde gerçek zamanlı gömülü sistem uygulamaları önemli rol oynamaktadır. Uydular ise uzay çevresel şartlarına dayanıklı gerçek zamanlı gömülü uygulamalardır. Ticari veya askeri bir uydu projesi yaklaşık üç yüz milyon dolar seviyesinde maliyetleri vardır bu sebepten birçok uydu üreticisi fırlatmadan önce uydularını doğrulama ihtiyacı duyarlar ve uydu simülatörleri en çok tercih edilen doğrulama altyapıları olarak öne çıkarmıştır. Özellikle Uydu merkezi bilgisayarında koşan uydu uçuş yazılımlarını doğrulamak önem kazanmıştır. Bu tezde gerçek zamanlı görev zamanlayıcılarına odaklanılmıştır. Altyapımıza uygun tek işlemcide koşan sabit öncelikli görev zamanlayıcılardan Round Robin (RR), Rate Monotonic (RM) ve Event Driven (ED) seçilmiştir. Çalışmamızda bu görev zamanlayıcılar işlemci kullanım performanslarına göre karşılaştırılmıştır. Görev zamanlayıcıları RTEMS işletim sisteminde 10 Hz ile çalışan bir kapalı döngü simülasyon altyapısında koşturulmuştur. Görev zamanlayıcıların performans karşılaştırılması için iki adet görev belirlenmiştir bunlar Yönelim Belirleme ve Kontrol Sistemi kontrolcüsü ile MIL-STD 1553 veri yolu kontrolcüsü görevleridir. Yapılan testlerde üç görev zamanlayıcısı ile bu iki görev koşturulmuş ve elde edilen sonuçlar birbirine yakın değerler çıkmıştır. Değerlendirme sonucu RR ve ED görev zamanlayıcıları seçilmiştir. RR uygulama kolaylığı ve ED'nin tasarımcıya tam kontrol sağlaması bu görev zamanlayıcılarını seçmemize büyük etkendir.

ANAHTAR SÖZCÜKLER: Gerçek zamanlı gömülü sistemler, gerçek zamanlı işletim sistemi, Round Robin görev yöneticisi, Rate Monotonic görev yöneticisi, Event Driven görev yöneticisi, uydu simülatörü.

Danışman: Prof. Dr. Mehmet Reşit TOLUN, Başkent Üniversitesi, Bilgisayar Mühendisliği Bölümü.

ABSTRACT

Embedded real-time applications play an important role today. Satellites are also robust embedded real-time applications. A commercial or military satellite project can cost well over three-hundred million dollars. Since many satellite manufacturers need to validate their satellites before launching, satellite simulators play the most valuable role in validation infrastructures. Specifically, satellite flight software validation has become more important. In this work, we focused on the round robin (RR), rate monotonic (RM), and event driven (ED) real-time scheduling task methods with respect to their CPU usage performance for satellite simulator infrastructures. The tasks are evaluated and tested under the real-time executive for multiprocessor systems (RTEMS). Those scheduling tasks are used in polling mode in the simulation setup. In this study, we compared three task scheduler methods for orbit altitude control system tasks and MIL-STD 1553 bus data distribution controller tasks in a spacecraft simulator environment. The results were close and the values were not segregated, thus, RR and ED can be chosen, as RR was easy to implement and ED allowed for full control of the tasks.

KEYWORDS: Real-time embedded systems, Real-time operating system, Rate monotonic task,Round robin task, Event driven task handling, and Satellite simulations

Supervisor: Prof. Dr. Mehmet Reşit TOLUN, Başkent University, Department of Computer Engineering

İÇİNDEKİLER LİSTESİ

Sayfa

ŞEKİLLER LİSTESİ

TABLOLAR LİSTESİ

SİMGELER VE KISALTMALAR LİSTESİ

A	Amper
V	Volt
MHz	Mega Hertz
AHB	AMBA High-Speed Bus
AMBA	Advance Microcontroller Bus Architecture
CAN	Controller Area Network
DUS	Dinamik Uydu Simülatörü
ED	Event Driven: Olay Tabanlı
EDF	Earliest Deadline First: En Erken Tamamlanma Zamanı
ESA	Avrupa Uzay Ajansı
FPGA	Field Programmable Gate Array
FDIR	Fault Detection Isolation and Recovery: Hata denetimi izolasyon ve iyileştirme
GZIS	Gerçek Zamanlı İşletim Sistemi
ID	Tanımlama
Idle	Idle Time: İşlemci Bos Zamanı
MRG	Messaging Real-Time Grid
NATO	North Atlantic Treaty Organization
PR	Preemption: Önceliklendirme
RAM	Random Access Memory
RR	Round Robin
RM	Rate Monotonic: Frekans Monotonik
RTEMS	The Real-Time Executive for Multiprocessor Systems
SMP2	Simulation Model Portabilty
SoC	System on Chip
TCB	Task Control Block: Görev kontrol Bloğu
UMB	Uydu Merkezi Bilgisayarı
VHDL	Verilog Hardware Description Language
YBKS	Yönelim Belirleme ve Kontrol Sistemi

1 GİRİŞ

Mobil araçlar günümüzde çok yaygın kullanılmaktadır. Mobil araçların pil ile çalışma, az enerji tüketimi ve hafif olma gereksinimleri bulunmaktadır. Uydular ise dünya dışı mobil cihazlar olup; enerji tüketimi, ağırlık ve yapılan görevlere cevap verme süreleri önem kazanmıştır. Sözü edilen kısıtlarda uydularımız için az enerji harcayan, gerçek zamanlı çalışma yapabilen, az yer kaplayan, ağır olmayan çeşitli donanımlar ve yazılımlar geliştirilmiştir.

Milyon dolarlık uzay projelerinde bütçelerin yarısını fırlatma maliyetleri oluşturmaktadır. Diğer bir nokta da uzay araçlarının bir defa fırlatılması ve geri dönmemesidir. Bu durum uydu simülatörlerinin ne kadar önemli olduğunu göstermektedir. Uyduları dünyada çalıştırmak ve çevreselini modellemek durumunda kalınmıştır.

Uzay araçları için tasarlanan ekipmanların hafif, yetenekli ve radyasyon dayanımlarının yüksek olması gerekmektedir. Büyük bilgisayar çekirdeği ve büyük hafıza demek, hem ağır hem de uzayda bulunan radyoaktif yüklü parçacıkların hafızada ya da bilgisayar çekirdeğinde bulunan veri değerlerini değiştirme olasılığını arttırması demektir. Ekipman tasarımında uydunun görevini yapması için gerekli minimum ağırlıkta ve büyüklükte donanımlar tasarlanmaktadır. Oluşturulan yazılımların bu hassasiyetle tasarlanmış ekipmanlara sığma zorunlulukları bulunmaktadır; küçük, işlevsel, sağlam ve dayanıklı olmak zorundadırlar.

Uyduların yörüngelerde durabilmeleri için dünyanın çekim kuvveti ile uydu hızının merkez kaç kuvvetinin dengede olması gerekmektedir. Böyle bir durumda uydu, 7500 m/s hız ile 700 kilometrelik yörüngede hareket etmektedir. Uydu görevlerinde sensörlerden alınan ölçüm sonuçları gerekli hesaplamalardan sonra görevin getirdiği ister ile eyleyicilere komutlar gönderir, buna görev (task) denir. Uydunun hızı göz önüne alınırsa bütün bu işlemlerin çok hızlı olması gerekmektedir. Gerçek zamanlı çalışma, bu belirtilen görevlerin tanımlanan zaman aralığında ve gecikmelerle yapılmasıdır. Tasarımcı bu zaman aralığını belirleyerek gerekli

donanımı, gerçek zamanlı işletim sistemini ve görev zamanlayıcısını kısıtlı kaynaklar ile tasarlamalıdır.

Uydu yazılımlarının doğrulanması ve çeşitli testlerin yapılabilmesi için en iyi platformların uydu simülatörleri olduğu değerlendirilmektedir. Bu simülatörlerin çalıştığı uydu merkezi bilgisayarında (UMB) Leon3FT işlemcili RTEMS gerçek zamanlı işletim sistemi üzerinde görev zamanlayıcılarımızı karşılaştırmak için uzay uygulamalarında çok kullanılan Yönelim Belirleme Kontrol Sistemi (YBKS) kontrolcüsü ve veri alışverişi yapan MIL-STD 1553 veri yolu kontrolcüsü kullanılmıştır.

Tek işlemcide çalışabilen ve sabit öncelikli görev zamanlayıcıları içinden üç çeşit zamanlayıcı seçilmiştir. Bu sebepten elimizde bulunan düzeneğe uygun Round Robin, Rate Monotonic ve Event Driven görev zamanlayıcıları ile çalışılmıştır.

Görev zamanlayıcılarının çalışma performansları uydu performansını doğrudan etkilemektedir. Görevlerin çalışma hızları ve sıralanması görev zamanlayıcıları tarafından yapılmaktadır. Duruma göre hangi görev zamanlayıcısının daha etkili performans gösterdiği önem kazanmaktadır. YBKS ve MIL-STD 1553 seçilen görev zamanlayıcıları ile koşturulmuş ve sonuçlar karşılaştırılmıştır. Elde edilen sonuçlar birbirine yakın değerlerde çıkmış, avantajları ve dezavantajları değerlendirilmiştir.

Yapılan çalışmanın organizasyonu aşağıda paragraflar halinde verilmektedir.

Giriş bölümü; uydu projelerinde karşılaşılan sorunlar, bu sorunların neden olduğu alt gereksinimler ve çalışmayı yapmamızdaki sebepler ortaya konulmaktadır. Uydu simülasyonlarının uydu üreticileri için önemi, kullanım alanları, çalışılan alt yapının teknik özellikleri, uydu ile görev zamanlayıcıların ilişkisi, görev zamanlayıcıların önemi, görev zamanlayıcıların seçimi ve sonuçlar hakkında özet bilgi verilmektedir.

Uzay aracı simülatörleri bölümü; uzay aracı simülatörlerinde kullanılan altyapılar açıklanmıştır. Uyduların doğrulanma amacı verilmiş hangi altyapı hangi seviyede doğrulama yaptığı açıklanmaktadır.

Gömülü gerçek zamanlı sistemler bölümü; bu bölümde gerçek zamanlı sistemlerin tanımı, amacı, mimarisi, çeşitleri ve çalışmada kullanılan RTEMS gerçek zamanlı işletim sistemi hakkında bilgi verilmektedir.

Çalışma ortamı bölümü; çalışma ortamının tanımı ve bileşenleri verilmektedir. Uydu merkezi bilgisayarı, MIL-STD 1553 haberleşme veri yolu ve simülasyon çerçevesi ayrıntılı bir şekilde anlatılmaktadır. Çalışma ortamında bulunan bileşenler arası ilişkiler verilmekte ve arayüzlerin ayrıntıları açıklanmaktadır.

Görev zamanlayıcıları bölümü; çalışmadaki odak nokta bu bölümde açıklanmaktadır. Görev zamanlayıcıların tanımı, özellikleri, çeşitleri ayrıntılı olarak verilmiştir ve görev zamanlayıcı çeşitlerini örnekler üzerinden açıklamalarda bulunulmuştur. RTEMS'de bulunan görev zamanlayıcılarının yapılandırılması ve yeni bir görev zamanlayıcı geliştirilmesi anlatılmıştır.

Yürütülen çalışmalar bölümünde; bu bölümde UMB'de koşacak görevlerin görev zamanlayıcıları ile çalıştırılması ve sonuçları verilmiştir. UMB üzerinde RTEMS işletim sisteminde YBKS ve MIL-STD 1553 görevleri kullanarak Round robin, Rate monotonic, Event driven görev zamanlayıcılarının performans sonuçları grafikler ile verilmiştir.

Sonuçlar bölümünde; YBKS ve MIL-STD 1553 görevleri kullanarak Round robin, Rate monotonic, Event driven görev zamanlayıcılarının performans sonuçları tartışılmış, avantaj ve dezavantajları ortaya konulmuştur. Gelecek çalışmalarda ne yapılacağından bahsedilmiştir.

2 UZAY ARACI SİMÜLATÖRLERİ

Uydular zorlu ortamda çalışmak için tasarlanmaktadırlar. Zorlu uzay koşulları için yoğun radyasyon, çok hızlı uzay çöpleri ve küçük asteroit parçacıkları örnek verilebilir. Bunun yanında uydular bir defalığına fırlatırlar, üretimleri bitip fırlatıldıktan sonra bir daha dokunma ve parçalarını değiştirme gibi bir şans bulunmamaktadır. Bu kısıttan dolayı tasarımcıların istenilen performansta istenilen uyduları yapmaları, sıkı doğrulama ve testler ile başarılmaktadır. Uyduda oluşabilecek sorunlar ve durumlar belirlenir, testler sırasında her senaryo denenmelidir. Uzay ortamında karşılaşılan sorunlardan çevresel faktörler dışında, fonksiyonel olarak uydunun çalışması ve görevi başarması en önemlisidir. Fonksiyonel olarak çalışmayı test etmek ve doğrulamak denilince akla simülatörler gelmektedir [36].

Uzay aracı simülatörlerinde amaç doğrulamak, test etmek, isterleri geçerli kılmaktır. Doğrulama faaliyetleri üç ana başlıkta incelenebilir.

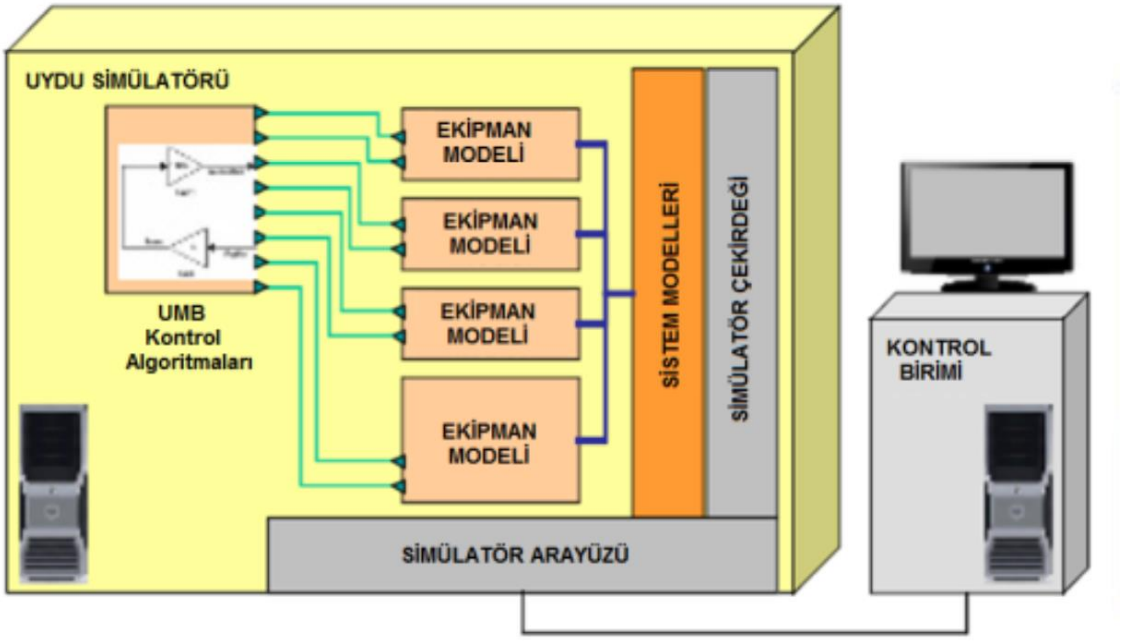
- Ekipman seviyesi
- Alt sistem seviyesi
- Uydu seviyesi

Ana doğrulama faaliyetleri dışında uçuş bilgisayarlarında iç iletişim, kayıt tutma (loglama), hata denetimi izolasyonu ve iyileştirme (FDIR) [34], alarm oluşturma gibi uydu sağlığını denetleyecek ek görevler oluşturulup doğrulanmalıdır.

Uyduyu her seviyede test etmek için çeşitli altyapılar bulunmaktadır. Amaçlarına göre doğrulama altyapıları alt başlıklar olarak verilmiştir.

2.1 İşlevsel Doğrulama Altyapısı

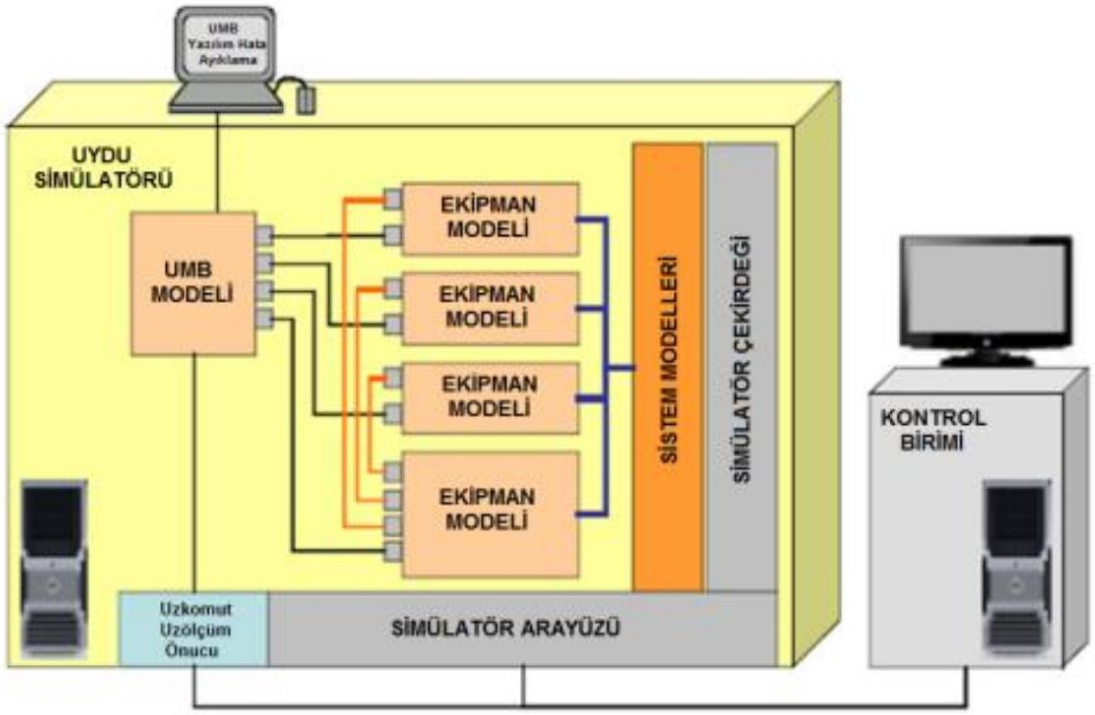
İşlevsel doğrulama altyapısı [1] ile UMB'de koşacak olan algoritmaların işlevsel doğruluğu test edilmektedir. Bu modelde UMB temsil edilmemektedir. Ayrıca işlevsel doğrulama altyapısı, YBKS algoritmalarının limitli doğrulanması için kullanılmaktadır. (Şekil 2.1)



Şekil 2.1 İşlevsel Doğrulama Altyapısı

2.2 Yazılım Çevrimde Doğrulama Altyapısı

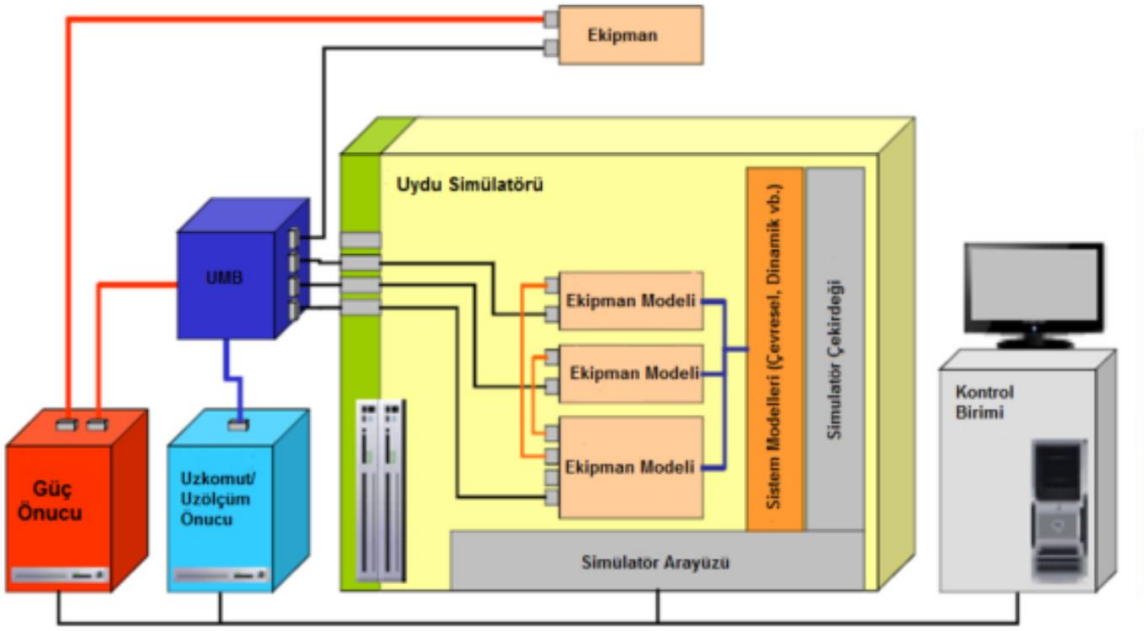
Bu altyapıda arayüzlerin, bağlantıların ve standartların birbirleri ile uygun şekilde çalışmaları test edilmektedir [1]. UMB model ya da emülatör olabilir. Emülatör UMB'nin mükemmel yazılımsal kopyasıdır [36]. (Şekil 2.2)



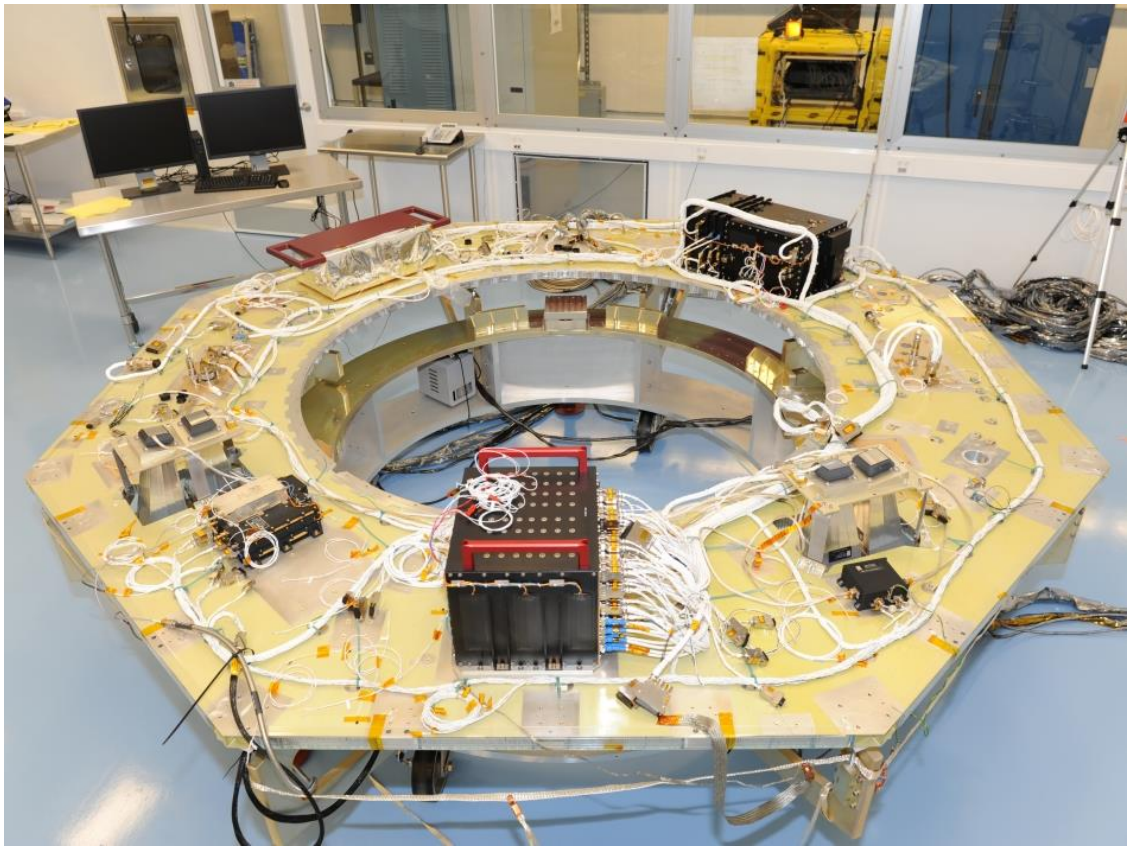
Şekil 2.2 Yazılım Çevrimde Doğrulama Altyapısı

2.3 Aviyonik Test Altyapısı

Aviyonik test altyapısı [1] uzay araçları için en gelişmiş test altyapısıdır. Doğrulama mühendisleri tasarımlarını bu altyapıda çözümlemektedirler. Ekipmanlar, benzetim veya gerçek ekipman olabilmektedir. Benzetimler ile gerçek ekipmanlar test için birbirlerinin yerine geçebilmektedirler [36]. (Şekil 2.3)



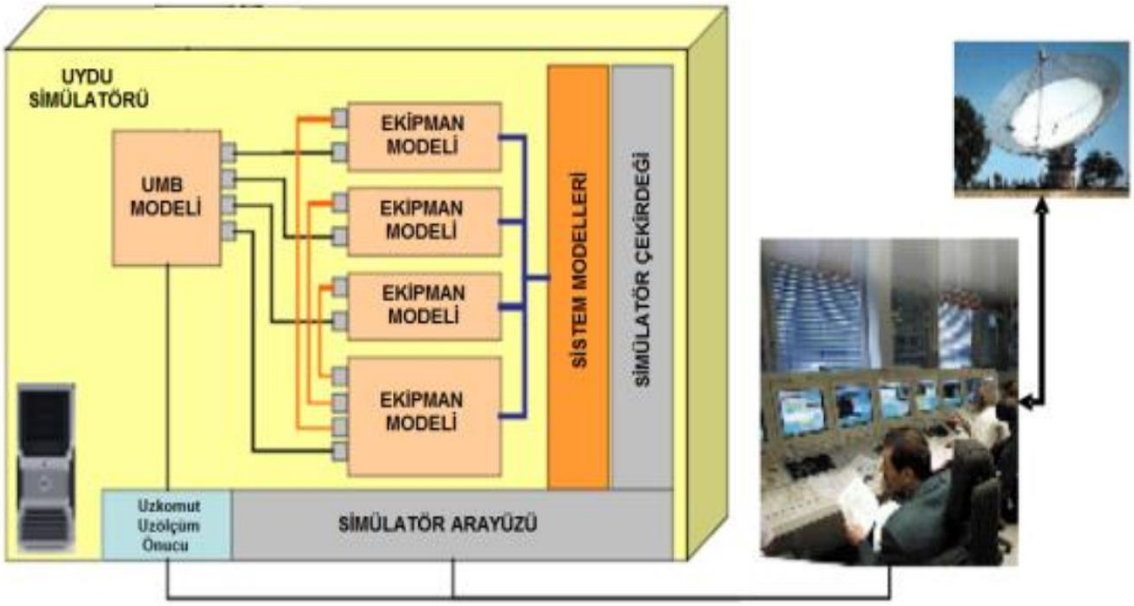
Şekil 2.3 Aviyonik Test Altyapısı



Şekil 2.4 Örnek Aviyonik Test Altyapısı

2.4 Dinamik Uydu Simülatörü

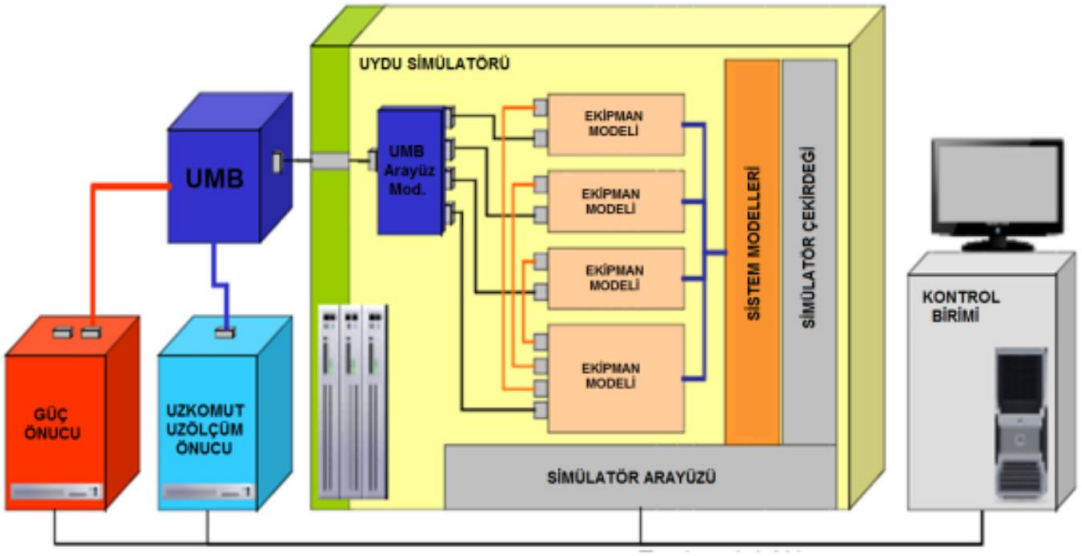
Dinamik uydu simülatörü (DUS) altyapısı [1] genelde telemetri, telekomut doğrulamaları için kullanılmaktadır. Uyduya telekomut gönderilmeden önce uyduyu tehlikeye sokmamak için gönderilecek komutlar ilk olarak DUS'ta denemektedir. Aynı zamanda uydu operatörlerinin eğitimlerinde de DUS kullanılmaktadır. Fırlatma ve erken yörünge fazında (LEOP) DUS ile yapılan senaryolar uyduyu kaybetmemek için önem arz etmektedir. DUS, erken yörünge fazında ve nominal fazda ön çalışmaya imkan sağlamaktadır. (Şekil 2.5)



Şekil 2.5 Dinamik Uydu Simülatörü

2.5 Hibrit Sistem Simülasyon Altyapısı

Hibrit sistem simülasyon altyapısında [1]; UMB gerçek donanım, diğer uydu ekipmanları ise simülasyon olarak çalıştırılmaktadır. Amaç gerçek UMB üzerinde uçuş yazılımı doğrulaması yapmaktır. Bu alt yapının girdi/çıktıları direk olarak test etme üstünlüğü bulunmaktadır. Çalışmamızı yaptığımız altyapı bu altyapıya örnek gösterilebilir.(Şekil 2.6)



Şekil 2.6 Hibrit Sistem Simülasyon Altyapısı

3 GÖMÜLÜ GERÇEK ZAMANLI SİSTEMLER

Bir amaca yönelik görev veya iş yapabilen donanımlarda koşan, belirli zaman penceresinde görevlerini tamamlayabilen sistemlere gömülü gerçek zamanlı sistemler denilmektedir. Gerçek zamanlı işletim sistemleri bu tür sistemlerin performanslarını artıran yazılımlardır. Günümüzde karmaşık görevler bir amaca yönelik gömülü sistemlerde koşturulmaktadır. Bu durum, uygun gömülü sistem için uygun işletim sistemini seçmemizi gerektirmektedir [21].

Katı gerçek zamanlı sistemler: Meydana gelebilecek bir gecikme için tolerans derecesi oldukça düşüktür. Zamanında yapılmayan görevler sistem için yıkıcı etki oluşturabilmektedir.

Sıkı gerçek zamanlı sistemler: Görevlerin zamanında bitirilememesi kalite azalmasına neden olabilmektedir.

Gevşek gerçek zamanlı sistemler: Görev için atanan sürenin dışına çıkılması çok önemli olmayabilir, atanan süredeki kayıp ve sistem kalitesindeki azalma göz ardı edilebilmektedir.

Gerçek zamanlı sistemler, yapılacak işin gereksinimlerine göre katı veya sıkı gerçek zamanlı olarak tanımlanmaktadırlar. Serbest bırakılma zamanı (release time) ile işin bitirme zamanı (deadline) arasındaki süre, icra zamanı (execution time) olarak adlandırılmaktadır. İşin bitirilme zamanının aşılması istenmiyorsa, sistem katı gerçek zamanlı olarak tasarlanmalıdır. Görevlerin koşacağı ortam gömülü bir sistem olabilir, yeterli işlemci gücü ve hafıza kaynağı var ise sistem işin bitirme zamanını aşmaz. Ayrıca bu sistem her periyotta tutarlı ise katı gerçek zamanlı sistem olarak tanımlanmaktadır [10].

3.1 Gerçek Zamanlı İşletim Sistemleri (GZIS)

Gerçek zamanlı işletim sistemleri [7], donanım ile uygulama programları arasında arayüz oluşturan bir sistem programıdır. Bu programın oluşturduğu soyutlama katmanı, kullanıcının donanıma popüler yazılım dillerini kullanarak daha kolay ulaşmasını sağlamaktadır.

İşletim sistemlerinin ilk görevi uygulama programlarının ihtiyaçlarını karşılamak için donanım kaynaklarının yönetimini sağlamaktır. Bunu ise aşağıdaki özellikleri sayesinde sağlarlar.

GZIS, belirli bir zaman aralığında mantıksal olarak doğru sonuç vermesi gereken uygulamalarda ve gömülü sistemlerde kullanılmaktadır. GZIS'in zamanlama davranışı ve kaynakları etkin kullanımı niteliği, gömülü sistemlerde kullanılmasında tercih sebebi olmaktadır.

Gerçek zamanlı işletim sistemleri yazılım mimarisi katmanları arasında köprü görevi görmektedirler ve katmanlar arasındaki hizmetlerin iletişimini kurmaktadırlar. Farklı dilde yazılmış uygulamalar aynı işletim sistemi üzerinde çalıştırılabilmektedir.

Şekil 3.1 Yazılım Mimarisi

3.2 Gerçek Zamanlı İşletim Sistemleri Çeşitleri

GZIS, birçok firma tarafında piyasaya ticari olarak sunulmakla birlikte açık kaynaklı kod [23] şeklinde de bulunabilmektedir. Tasarımcılar tarafından en çok tercih edilenleri aşağıda verilmiştir [7]:

- RTEMS
- VxWorks
- QNX
- Mobilinux
- Nucleus RTOS
- Prex
- ECos
- FreeRTOS
- Gömülü Linux
- JavaOS
- LynxOS
- Windows CE
- Windows XP Embedded

GZIS'in tercih edilme nedenleri, gömülü sistem mimarisi için destekleniyor olması ve donanım destek yazılım paketlerinin (Board Support Packages – BSP) ilgili işletim sistemi için yazılmış olmasıdır. Örneğin; LEON3 çekirdeği gömülü sistem mimarisi için RTEMS işletim sisteminin destekleniyor olması, MIL-STD 1553B seri haberleşme protokolü donanım destek yazılım paketlerinin RTEMS işletim sistemi için yazılmış olması söylenebilir. İşletim sistemleri görevleri itibari ile tasarımcılar tarafından tercih edebilirler, buna örnek olarak RTEMS'in uzay uygulamalarında en çok tercih edilen işletim sistemi olması gösterilebilir. Bunun sebebi, RTEMS'in hafızada az yer kaplaması, katı gerçek zamanlı işletim sistemi olması ve uzay projelerinde tarihçesi olmasıdır. Bu çalışmada bizim de odaklanacağımız işletim sistemi RTEMS olacaktır.

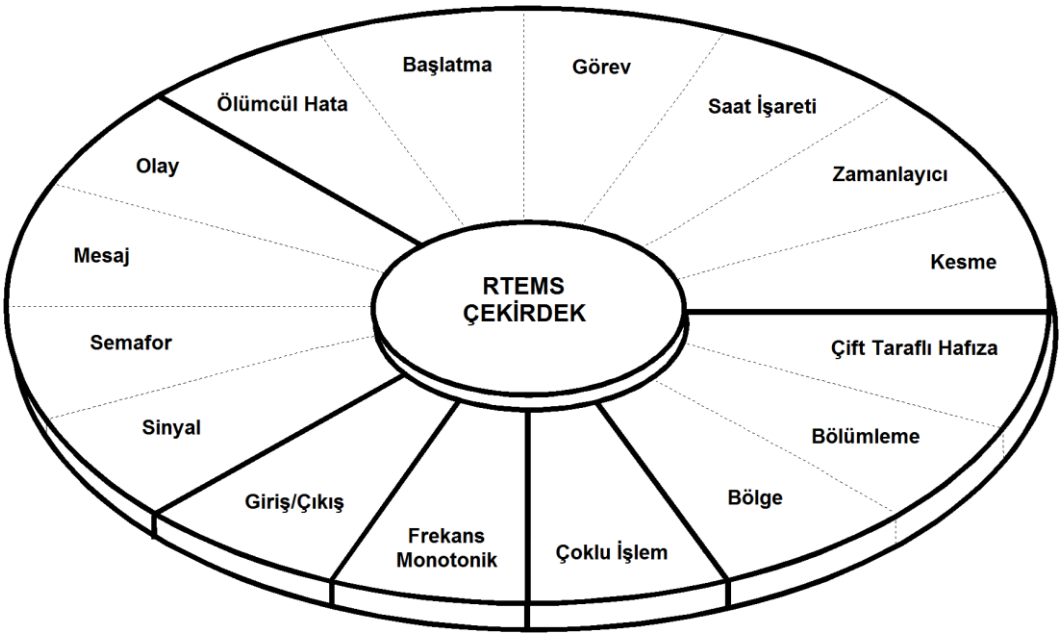
3.3 RTEMS

RTEMS, uzay uygulamalarında en çok tercih edilen açık kaynak kodlu, katı gerçek zamanlı işletim sistemidir. “The Real-Time Executive for Multiprocessor Systems” kelimelerinin baş harfleri ile RTEMS kısaltması oluşturulmuştur [8].

Yapılacak göreve göre tekrar düzenlenebilir bir işletim sistemidir ve genelde askeri uygulamalarda kullanılır. Bunun yanında, sadece hesaplamalara göre değil sonuçlara göre de düzeltme yapan işletim sistemidir.[22]

RTEMS birçok yöneticiden oluşur, bunlar aşağıda verilmiştir:

- Başlatma (initialization)
- Görev (task)
- Kesme (interrupt)
- Saat işareti (clock)
- Zamanlayıcı (timer)
- Semafor (semaphore)
- Mesaj (message)
- Olay (event)
- Sinyal (signal)
- Bölümleme (partition)
- Bölge (region)
- Çift taraflı hafıza (dual ported memory)
- Giriş/Çıkış (I/O)
- Ölümcül hata (fatal error)
- Frekans Monotonik (rate monotonic)
- Kullanıcı uzantıları (user extensions)
- Çoklu işlem (multiprocessing)

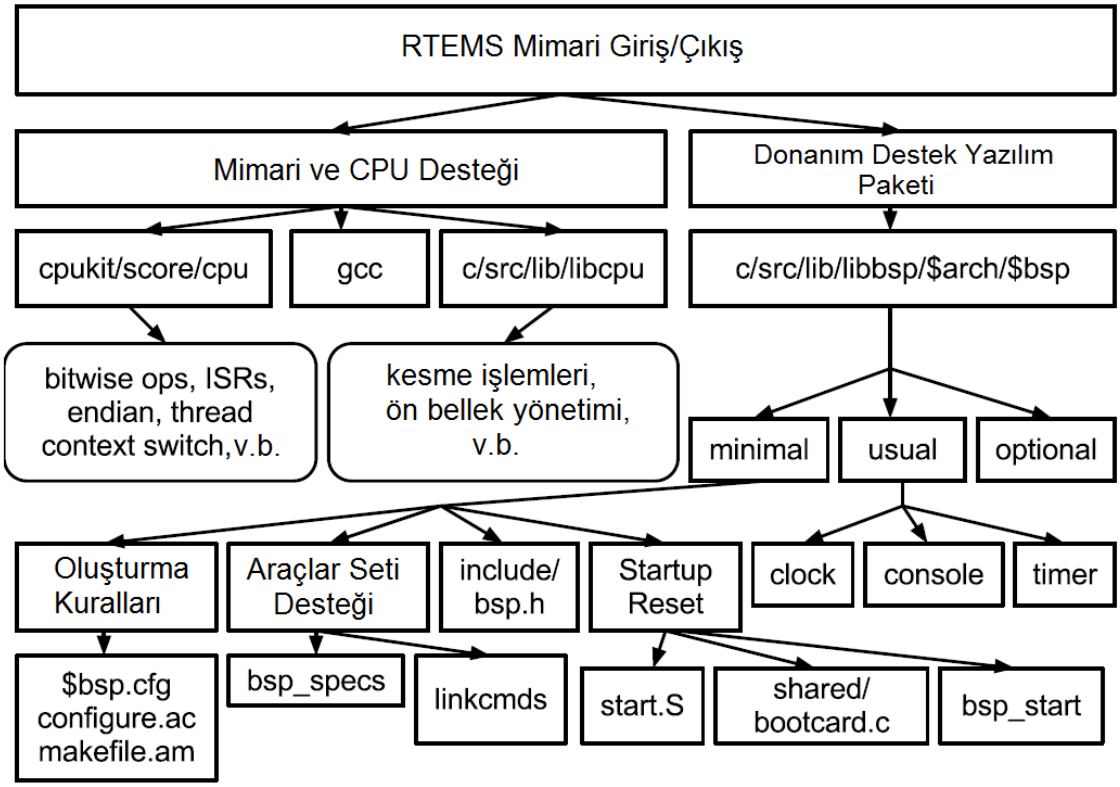


Şekil 3.2 RTEMS Yönetici Mimarisi [8]

Gömülü sistemlerde 32-bit işlemcilerin yaygın kullanımı sayesinde yan ürünleri bulmak kolay hale gelmiştir. 32-bit işlemciler çeşitli mimarilerde olabilirler, RTEMS birçok 32-bit mimari düşünülerek mimariden bağımsız şekilde yazılmış bir GZİS'dir. Bu özellik RTEMS'e işlemci kartından bağımsız saat işareti [28], kesme ve G/Ç cihazlarını kolaylıkla entegre imkânı vermektedir. RTEMS sistem tasarımcılarına çeşitli imkanlarla tasarımlarını genişletebilme kabiliyeti sunmaktadır [8].

RTEMS, işlemci kartı bağımsızlığına ve donanımlardan izole bir çekirdek yapısına sahip olduğu için başka bir işlemcide veya farklı donanımda çalışabilmektedir. Gerçek zamanlı sistem rahatlıkla başka bir işlemci kart sistemine taşınabilmektedir. Ayrıca RTEMS işlemciden bağımsız çalışabilmektedir [8].

Birçok gömülü gerçek zamanlı sistemde hafıza kritik bir kaynaktır. RTEMS, çalışma zamanı derlemelerinde kullanılmayan sistem yöneticilerini dışarıda bırakma özelliği ile küçük hafıza kullanımı sağlamaktadır. Hafızada kapladığı yer sistemden sisteme değişen bir yapıda olmaktadır. Kullanılan hafıza RAM veya ROM kullanımına imkân vermektedir. Ayrıca hem işletim sistemi hem de veri depolamak için gerekli hafızayı ayırabilmektedir [8].



Şekil 3.3 RTEMS Mimarisi [8]

RTEMS mimarisi yukarıda görülmektedir. İşlemci çekirdeği ve donanım destek yazılım paketlerinin (Board Support Packages – BSP) mimarideki yerleri açıklanmıştır.

RTEMS birçok işlemci mimarisinde koşabilen bir GZIS'dir Bunlardan birkaç örnek aşağıdaki gibidir:

- Arm - ARMV9, ARMV8, ARMV7
- C4xx - Texas Instruments - DSPs, C3x ,C4x
- Hitachi H8 ailesi - H8300
- Hewlett-Packard PA-RISC - hppa1.1
- i386 - Intel i386, Intel i486, Intel Pentium, AMD
- Intel i960 - Intel i960 ailesi
- m68kx - Motorola m680x0, m683xx, CPU32, Cold-fire CPU
- MIPS - MIPS ISA 32 ve MIPS 64 bit CPU

- or32 – Open-Cores, Open-Risc32 CPU
- Power-PC - IBM ve Motorola Power-PC 4xx, 5xx, 6xx, 7xx, 8xx, 74xx, ve 75xx24
- Sh serisi - Hitachi SH1, SH2, SH3, ve SH4
- SPARC - SPARC V7 ve SPARC V8 Leon2, Leon3, Leon3 FT

RTEMS işletim sisteminde uygulama programlama katmanı yazılımları görev zamanlayıcıları ile çağrılmaktadırlar.

Görev zamanlayıcılar bu tezin odak noktası olduğu için Bölüm 5'te ayrıntılı bir biçimde açıklanacak ve çalışmamız ile olan bağlantıları anlatılacaktır.

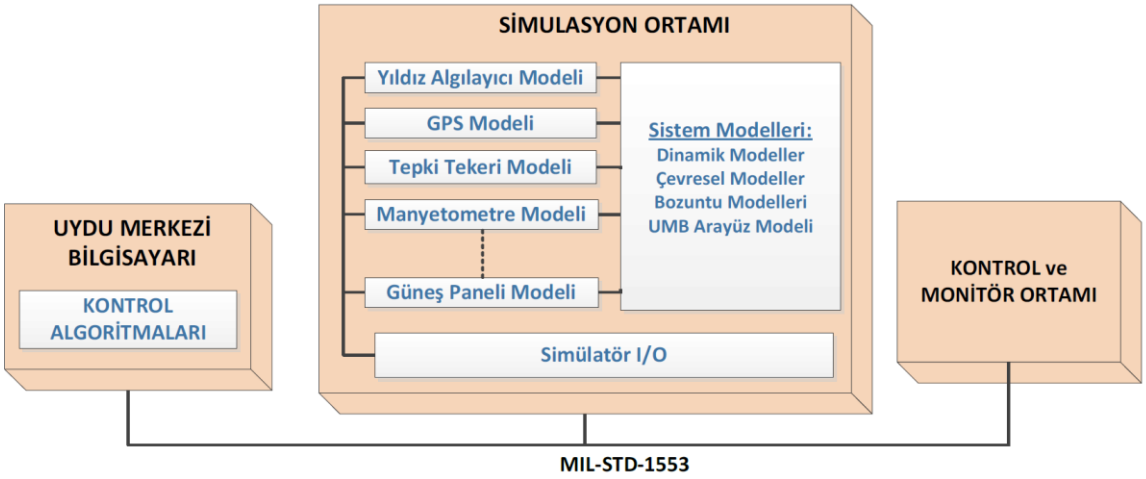
4 ÇALIŞMA ORTAMI

Uydular zorlu görev ortamları ve fırlatma gibi birçok zor şartlara dayanıklı olmak zorundadırlar. Uydu tasarımları yüksek teknoloji ürünler barındırmaktadır. Bu sebeple uyduların fonksiyonel olarak çalışabilirliğini test etmek gerekmektedir. Karmaşık arayüzlerin yüksek teknoloji ürünler ile uyumlu bir şekilde çalıştırılması, uzayda karşılaşılacak her senaryonun denenmesi, hatta anomali oluşturularak uydunun en kötü durumlara karşı sağlam olması hedeflenmektedir. Simülasyon ortamları bu tür testlerin yapılabileceği altyapılardır.

Uydu simülatörlerinin başarımı, sistemin ne kadar iyi modellendiği ile ilgili bir durumdur. Model gerçek sisteme ne kadar çok benzetilirse, testlerde elde edilen sonuçlar da o kadar gerçekçi olmaktadır. Bunun yanında hangi benzetim altyapısına ihtiyaç duyulduğu da önem arz etmektedir. Yukarıda anlatılan uydu simülatörleri (Bölüm 2) sistemin ihtiyaçlarına göre hangi altyapıyı kullanacağınız hakkında bilgi vermektedir. Bizim çalışmamızda uydunun sistem seviyesinde işlevselliğini test etmemiz yeterli olacaktır. Bu yüzden çalışmamızda Hibrit Sistem Simülasyon Altyapısı (Bölüm 2.5) kullanılmıştır.

Hibrit sistem simülasyon altyapısı; UMB gerçek donanımının ve simülasyon çerçevesinin, bir seri haberleşme standardı ile birbirine bağlandığı simülasyon altyapısıdır. Burada amaç uydu uçuş denetleyicileri ile ekipman, çevresel, dinamik modellerin bir uyum içinde kapalı döngü çalıştırılmasıdır.

TUSAŞ Uzay Sistemleri Müdürlüğü Sistem Entegrasyon Laboratuvarı'nda bulunan ve Şekil 4.1'de görünen sistem, UMB'nin uydularda kullanılan uçuş bilgisayarının radyasyon dayanımı olmayan ve donanımsal açıdan birebir aynısı olan kopyasıdır. Diğer ekipmanlar ve sistem modelleri benzetimdir. Bu yüzden çalışmada kullanılan altyapı hibrit sistem simülasyon altyapısıdır. Kapalı döngü altyapı katı gerçek zamanlı sistem olarak tasarlanmıştır ve 10 Hertz çalışmaktadır. Ancak istenilirse 20-30 Hertz seviyelerinde test yapmak da mümkündür.



Şekil 4.1 Çalışmada Kullanılan Hibrit Sistem Simülasyon Altyapısı

Ek olarak Şekil 4.1'de görülen monitör ortamında MIL_STD 1553 veri analizi gerçekleştirmek için kullanılmaktadır.

4.1 Uydu Merkezi Bilgisayarı (UMB)

UMB uydunun uçuş bilgisayarıdır. Bütün algoritmalar ve sistem için gerekli işleyiş programları tek bir işlemci çekirdeği (uniprocessor) üzerinde burada koşmaktadır. Uçuş yazılımı ve veri kotarma algoritmalarını en yüksek verimle çalıştırmak için gerçek zamanlı gömülü bir yapı kullanılmaktadır. Bu çalışmada Leon3 FT [5] çekirdeği mimarisine sahip bilgisayar üzerinde RTEMS [25] gerçek zamanlı işletim sistemi kullanılmıştır. Gaisler [13] firmasının RASTA [32] adlı Leon3 mimarili test altyapısı Şekil 4.2'de gösterilmektedir.

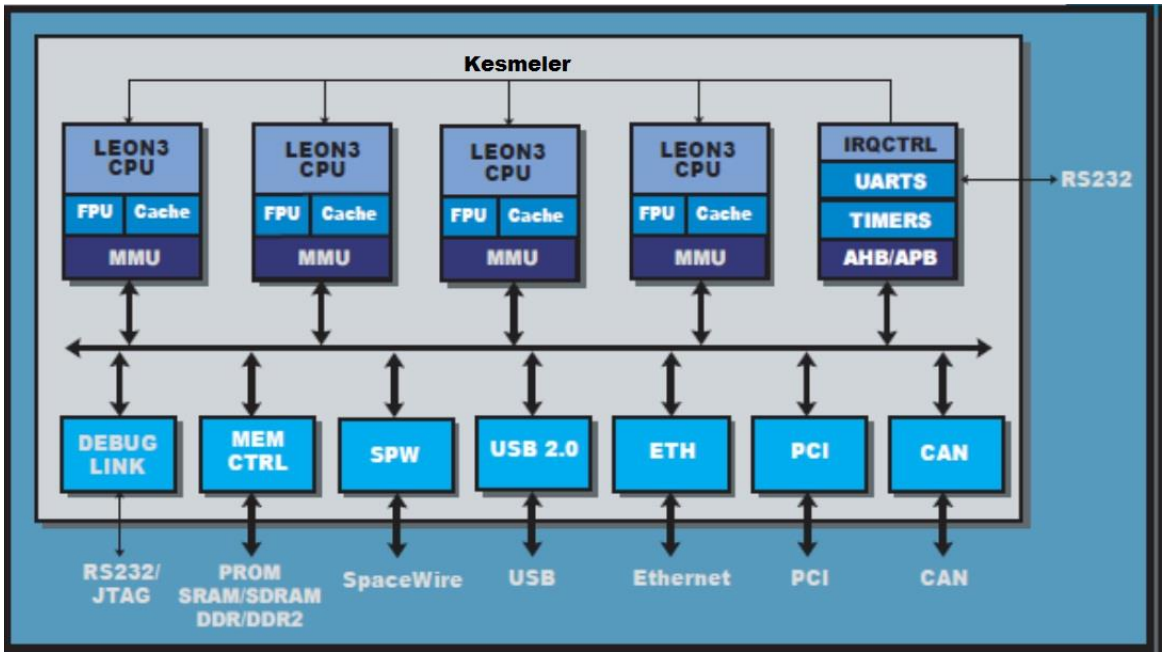
Şekil 4.2 RASTA - Leon3 Mimarili Test Altyapısı

4.1.1 LEON3 FT işlemci mimarisi ve kartı

Leon3FT, SPARC V8 mimarisi ile tasarlanmış [5], çoklu işlemi destekleyen (multiprocessor), 32 bitlik açık kaynak kodlu mikroişlemcidir. VHDL(Verilog Hardware Description Language) ile sentezlenebilir modeller ile oluşturulmuş 7 evre iş hattı (pipeline) mimarisine sahiptir. Simetrik ve asimetrik çoklu işlemcileri desteklemektedir, 4 işlemciye kadar çoklu çekirdek ile gerçekleme yapabilmektedir. Yonga üzerinde sistem (SoC) [31] özelliği ile kolay uygulamalar sunmaktadır. Leon3FT işlemci içeriği aşağıda sıralanmıştır, dört gerçek çekirdekli mimari Şekil 4.3'te verilmiştir [30].

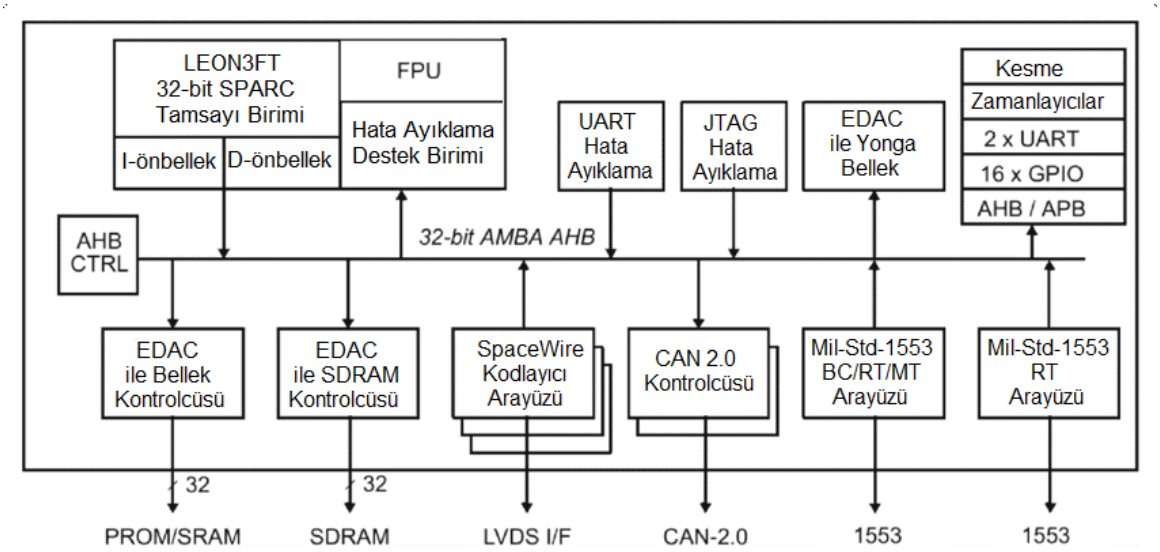
- A SPARC V8 komut seti ile V8 mimarisi
- 7 evre iş hattı (pipeline)
- İleri yonga üzeri hata ayıklama desteği ve veri izleme tamponu
- RAM hafıza alanı 1 den 512 KB kadar destekler
- Tutarlı, gürbüz ve tam eşzamanlı tek kenar saat işareti
- Yüksek performanslı 1.4 DMIPS/MHz, 1.8 CoreMark/MHz (gcc –4.1.2),

- Derleyiciler, sistem kabukları (kernel), simülatörler ve hata ayıklama monitörler gibi çok kullanılan yazılım araçlarına uyum.
- AMBA-2.0 AHB veri arayüzü
- 125 MHz FPGA hızı ve 400 MHz 0.13 um ASIC teknoloji hızı



Şekil 4.3 LEON3 FT Mimarisi [33]

AMBA-2.0 ve AHB veri arayüzleri (Şekil 4.4) ise Gaisler kütüphanelerini [13] kullanarak yonga üzerinde sistem [35] (SoC) yapısı ile erişim sağlamaktadır. Gaisler Kütüphanesi'ni geliştiren Gaisler firması, tasarımlarda kolay bir şekilde hata ayıklama yapılabilmesi için bazı araçlar geliştirmiştir. Gaisler Monitor [13] (GRMON), donanımın yonga FPGA'ya yüklenmesi ve üzerinde hata ayıklama yapılabilmesini sağlamaktadır [33].



Şekil 4.4 LEON3FT Mimarisi Üzerinde AMBA-AHB Veri Yolu [13]

4.1.2 UMB'nin test düzeneği içeriği

Bu çalışmanın test içeriği aşağıda maddeler halinde verilmiştir.

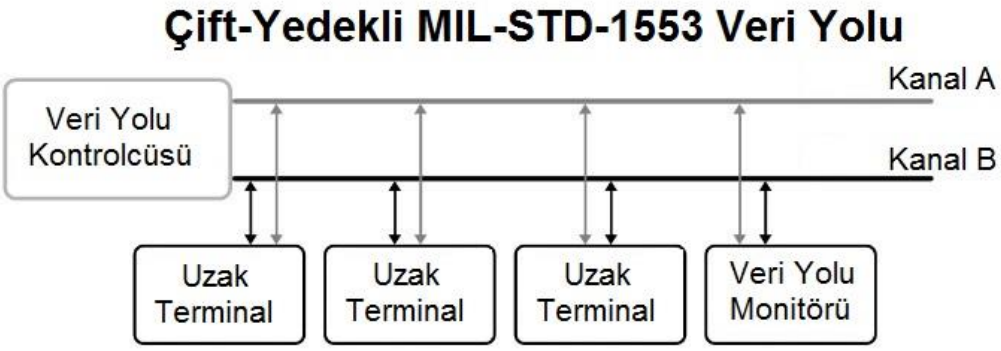
- LEON-3 FT çekirdeği
- RTEMS gerçek zamanlı işletim sistemi (OS)
- Gaisler kütüphanesi [13]
- MIL-STD-1553, CAN [20], Ethernet
- Kontrol Algoritma entegrasyonu:
  - AOCS Algoritması
  - Veri yönetimi
  - Sensor veri analizi
  - Eyleyici kontrol
  - Mod yönetimi

4.2 MIL-STD 1553 Seri Veri Yolu

MIL-STD 1553B [6] veri yolu askeri standartlarda tasarlanan bir seri çoklayıcı haberleşme veri yoludur. Bu standart; veri yolunun karakteristiğini, donanımını, mekanik yapısını ve veri paylaşım fonksiyonlarını tanımlamaktadır. İlk olarak F16 uçaklarında, sonrasında hava araçları üzerinde bulunan sistemler arasında veri iletişimi sağlamak için kullanılmıştır. Ayrıca bu standart NATO tarafından STANAG

3838 adıyla anılmaktadır. Çeşitli ülkeler tarafından askeri-sivil havacılıkta, otomotiv ve otomasyon sanayinde kullanılmaktadır. Protokolü, “Fiziksel Katman” ve “Protokol” olarak iki ana başlıkta toplanmaktadır.

MIL-STD-1553’te veri yolu için genelde çiftli dönen kablo (twisted shielded pair) veya dairesel kesitli (coaxial) tip kablo kullanılmaktadır. 1Mhz’de 70-85 Ohm değerinde direnç gösterir. Gerilimi 18V-27V arası değişmektedir.

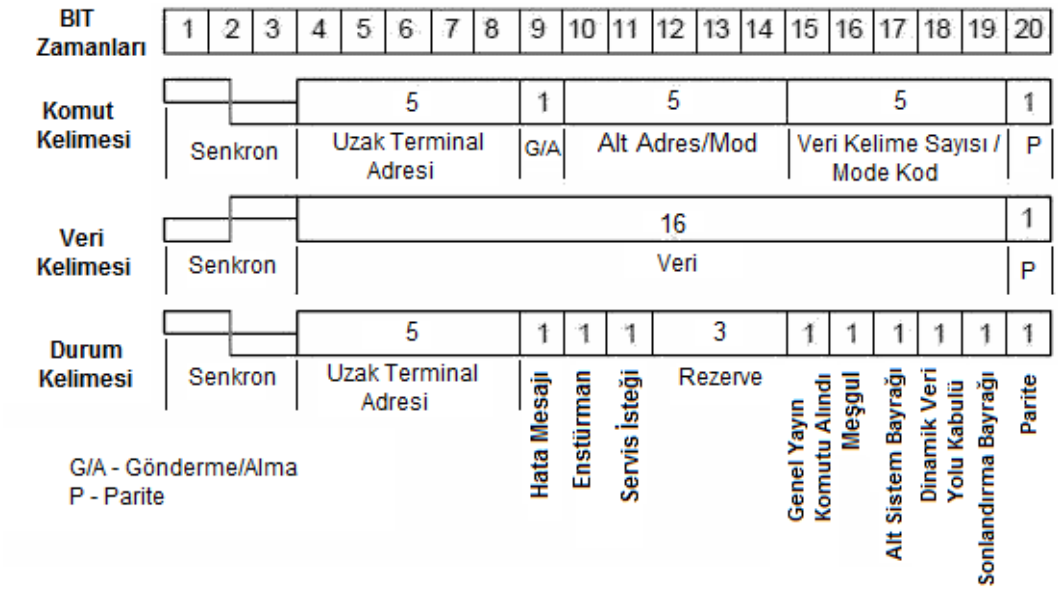


Şekil 4.5 MIL-STD-1553B Veri Yolu [6]

MIL-STD 1553 sistemi (Şekil 4.5) bir veri yolu kontrolcüsü (Bus Controller), uzak terminal (Remote Terminals), iletim hattı ve veri yolu monitöründen oluşmaktadır. İletim hattında haberleşme Manchester Kodlaması ile sağlanmaktadır. Bu kodlamada işaretler sinyal geçişleri ile kodlanmaktadır.

Protokolde ise veri yolundaki haberleşme, kelimeler (Word) ile yapılmaktadır. Veri yolunun kontrolü, veri akışı, durum raporlama ve yönetim üç farklı tip kelime ile sağlanmaktadır. Şekil 4.6‘da haberleşme protokolü kelime yapısı verilmiştir.

- Komut kelimesi (Command Words)
- Veri kelimesi (Data Words)
- Durum kelimesi (Status Words)



Şekil 4.6 MIL-STD-1553B Haberleşme Protokolü [6]

MIL-STD 1553 haberleşme protokolündeki tüm haberleşmeyi veri yolu kontrolcüsü yönetmektedir. Veri yolu kontrolcüsünün bir şekilde çalışması durursa, haberleşme sağlanamaz. Veri yolu monitörü pasiftir sadece kayıt yapar.

Şekil 4.7 MIL-STD-1553B Bağlantı Elemanları

MIL-STD 1553 kablo, çoklayıcı, sonlandırıcı ve diğer bağlantı elemanları Şekil 4.7'de gösterilmektedir.

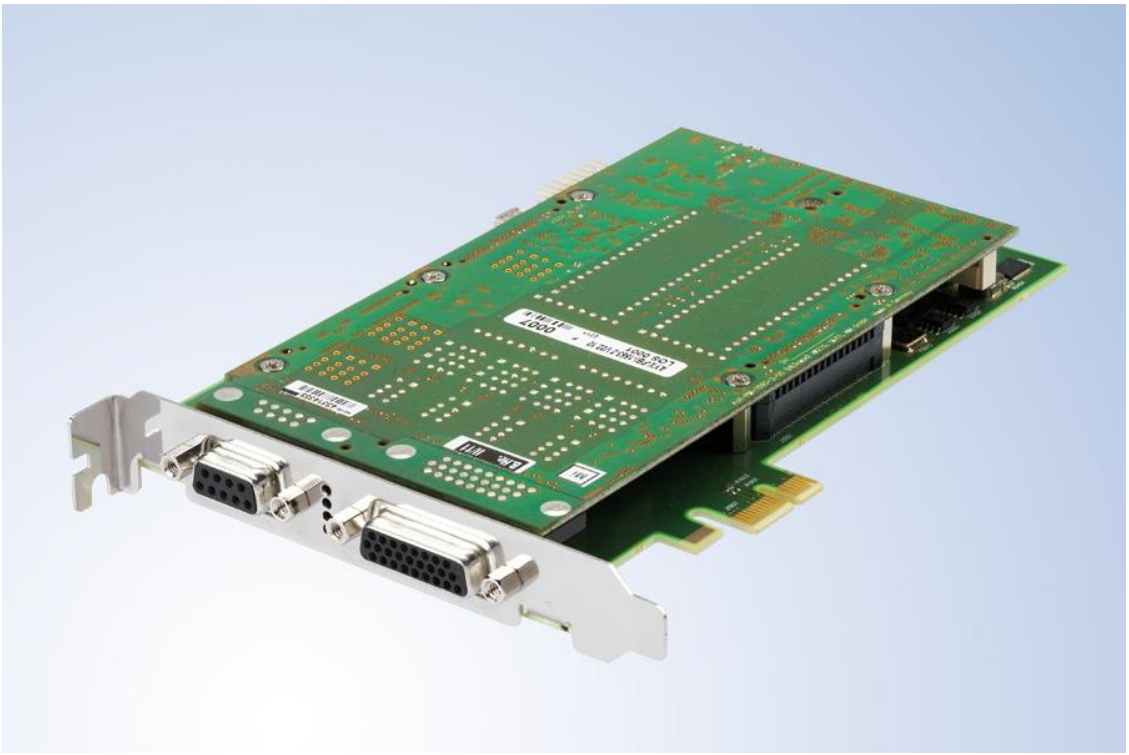
4.3 Simülasyon Çerçevesi (Simulation Framework)

Bir yazılım projesinde sonradan ekleyerek kullanabilen hazır kodlardır. Amaç belirli bir koda ait iş yükünün, hazır kod parçaları ile tekrar kullanılabilir hale getirilmesidir.

Simülasyon çerçeveleri, simülasyonumuzda bulunan ekipman ve çevresel modelleri bir çağırma sıklığı ile koşturmaktadır. Modeller değişen frekanslarla çağırabilmektedirler. Tüm bir sistem simülasyonu yapabilme kabiliyeti vardır. Sonuçları değerlendirmek için kayıt altına alarak, sonuçlar arasında araştırma yapmayı sağlamaktadır. Grafik arayüzler ile çıktılar gözlemlenebilmektedir. Bu yeteneklerinin yanı sıra asıl görevi dinamik bir simülasyon sunabilmesidir.

Eğer sistem gerçek zamanlı çalışacak ise simülatör çerçevesi de gerçek zamanlı çalışmak ve gerçek zamanlı sisteme cevap vermek durumundadır. Bu gerçek zamanlı yazılım çerçevelerini çalıştırabilmek için Linux tabanlı işletim sistemlerine ihtiyaç duyulmaktadır. Örneğin, bu çalışmada simülasyon çerçevesi Centos 7.3 Linux işletim sisteminde çalışmaktadır. Bunun yanında MRG (Messaging Real-time Grid) eklentisi (add-in) eklenerek gerçek zamanlı bir işletim sistemi kabuğu oluşturulmaktadır.

Bu simülasyon çerçevesinde veri alışverişi MIL-STD 1553 veri yolu üzerinden yapılmıştır. Bunun için PCI üzerinden çalışan bir MIL-STD 1553 kartı temin edilmiştir, bu kart Şekil 4.8'de gösterilen AIM- APE 1553-4 kartıdır [14].



Şekil 4.8 AIM-APE 1553-4 kartı

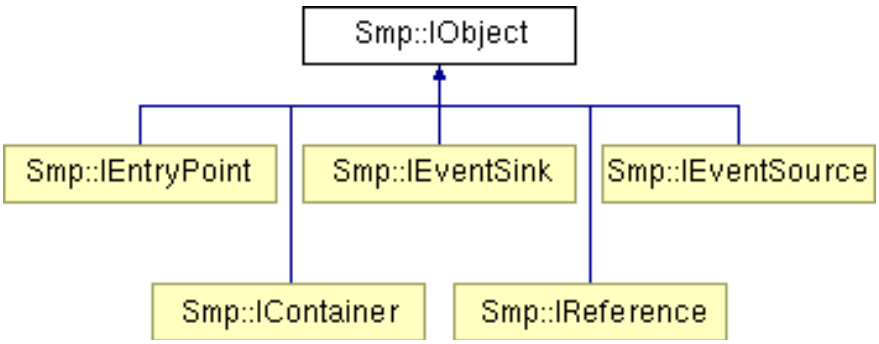
Uzay alanında birçok simülasyon çerçevesi bulunmaktadır. Bunun yanında sektördeki firmalar kendi firma içi yazılımlarını yapmaktadırlar. Bazı bilinen simülasyon çerçeveleri aşağıdaki gibidir:

- SimSat: Terma firmasının ESA için yaptığı üründür [15].

- EuroSim: DutchSpace tarafından geliştirilmiş ve ESA tarafından desteklenen Simülasyon çerçevesidir [16].

- Matlab: Mathwork firmasının genel simülasyon ve hesaplama çerçevesidir. Uzay uygulamalarında kullanımları bulunmaktadır.

Bu çalışmadaki bütün modeller EuroSim üzerinde çalıştırılmaktadır. Ayrıca yine bütün modeller SimSat ve EuroSim tarafından desteklenen SMP2 (Simulation Model Portabilty) [17] standardında yazılmıştır. Simülasyon çerçevesi değişse de modeller her çerçevede çalışacak şekilde oluşturulmuştur. Belirli bir nesne tipi bulunmaktadır (Şekil 4.9). Her modelde bu nesne oluşturulur ve bu nesneler bir meta-model ile parametreler olarak aktarılırlar.



Şekil 4.9 SMP2 Nesne Yapısı

5 GÖREV ZAMANLAYICILARI

Görev zamanlayıcıları bir işletim sistemindeki görevlerin (task) koşturulmasından sorumlu işletim sistemi programlarıdır. Uydu uçuş bilgisayarı tüm aracın uyumlu bir şekilde çalışmasından sorumludur. Uyumun sağlanması için sistemin katı gerçek zamanlı bir sistem olması gerekmektedir. Uydu bilgisayarında koşan uçuş yazılımı görev zamanlayıcılarının görevlerini zamanında bitirmeleri, uydunun görevinin başarılması adına çok önemlidir. Bu yüzden görev zamanlayıcılarının performansları uydunun performansı ile doğru orantılıdır.

5.1 Görev Zamanlayıcısı Tanımı

Kompleks sistemlerde çok girişli çıkışlı zaman kısıtları bulunan paralel görev yapma ihtiyaçları ortaya çıkmaktadır. Paralel görev tasarımları en küçük görev parçalarını ve planlamalarını göz önünde tutmak zorundadırlar. Bu yüzden birçok GZIS çekirdeği görev (task) yönetimi ve görev nesneleri sağlamaktadır.

Her görev zamanlayıcısı için gerekli bazı parametreler bulunmaktadır. Bu parametreler; adı, ID, önceliği, Task Control Block (TCB) ve görev kodudur. Kabukta (kernel) sistem görevleri de mevcut olabilmektedir. Bu sistem görevlerine boşta (IDLE) durum, görevlerin bilgi düşmesi (loglanması), olağandışı durum işleme (exception handling) ve hata ayıklama (debug) görevleri örnek olarak verilebilir.

5.2 Görev Zamanlayıcısı Özellikleri

Çok çeşitli görev zamanlayıcıları bulunmaktadır. Görev zamanlayıcıları özelliklerine göre aşağıdaki gibi sınıflandırmıştır.

İşlemci kullanımına göre: Tek işlemcili (uniprocessor) veya çok işlemcili (multiprocessor) [2] mimarileri kullanabilen görev zamanlayıcıları [4].

Önceliklendirmeye göre: Önceliklendirmeye (preemptive) [19] ya da önceliklendirme (nonpreemptive) olmaksızın kullanılabilen görev zamanlayıcıları. Preemptive yapılarda bir görev koşarken önceliği daha yüksek bir görev geldiğinde

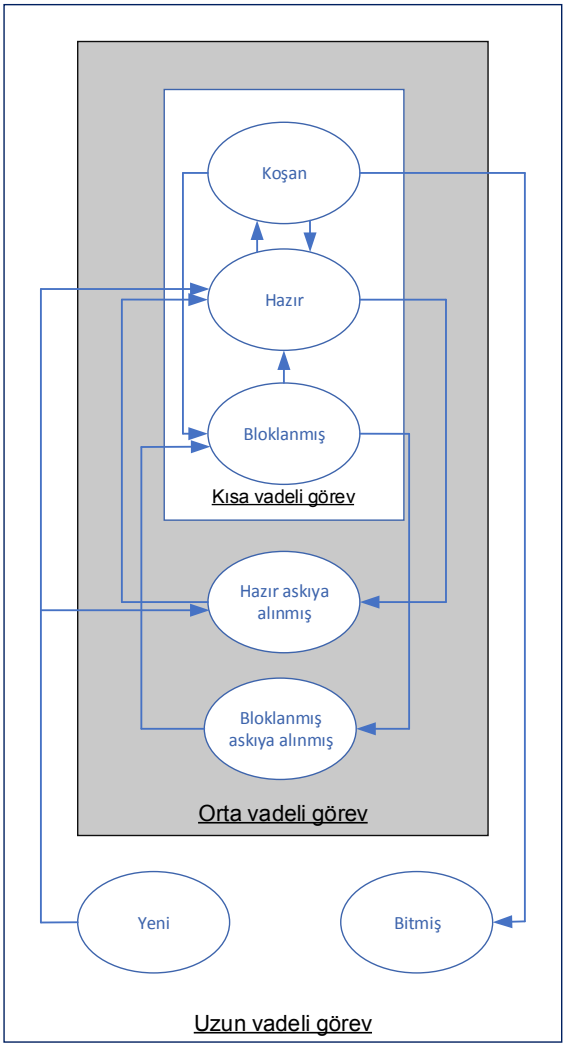
kesilir ve yüksek öncelikli görev koşmaya başlar. Kesilen görev öncelik sırası onda ise kaldığı yerden koşmaya başlar.

İşlemciye gelme süresine göre: İşlemciye gelme süresi görevin nerede olduğuna bağlıdır [18] (Şekil 5.1).

Kısa vadeli görev: Görev koşuyor, hazır ya da bloklanmışsa kısa vadeli görevdir.

Orta vadeli görev: Hazır askıya alınmış veya bloklanmış askıya alınmış görev ise orta vadeli görevdir.

Uzun vadeli görev: Yeni veya bitmiş görev ise uzun vadeli görevdir.



Şekil 5.1 İşlemciye Gelme Süresine Göre Görev Zamanlayıcıları

Önceliğine göre: İşlemciye gelme önceliği (priority) en yüksek olan görev gelir. Görev zamanlayıcılar sabit öncelik atamalı ya da dinamik öncelik atamalı olabilirler. Bu çalışmada sabit öncelik atamalı görev yöneticileri tercih edilmiştir [7].

5.3 Görev Zamanlayıcısı Çeşitleri

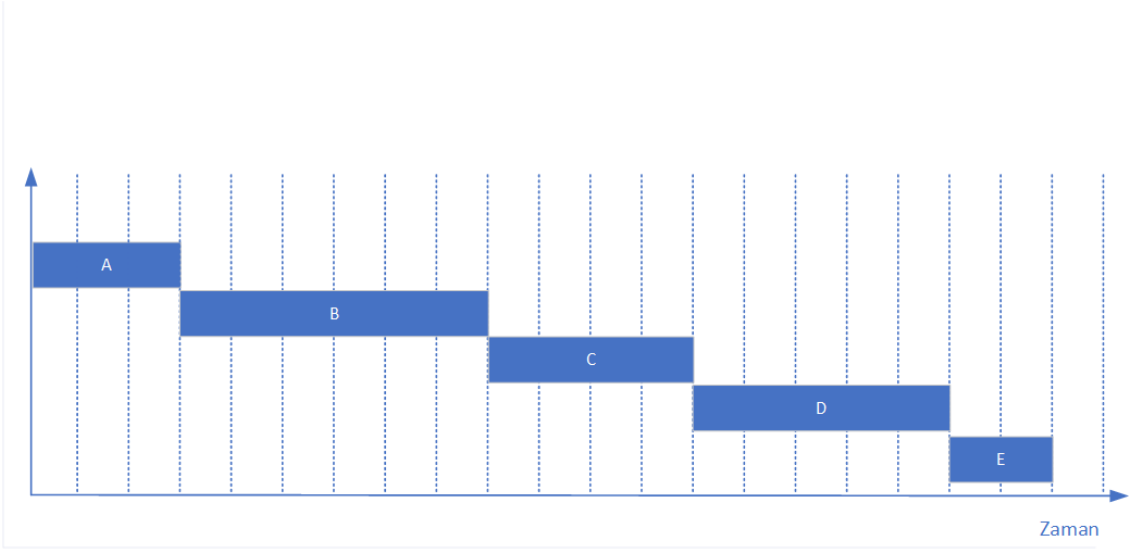
Görev zamanlayıcıları hangi görevi nasıl ve ne zaman çalıştıracağına karar verirler. Bunun için bir dizi kuralları uygularlar, buna görev zamanlayıcısının algoritması denir. Genel bir bakış açısıyla çok kullanılan görev zamanlayıcıları aşağıda verilmiştir.

Tablo 5.1 Örnek Görev Zamanlayıcılar Listesi [18]

Görev	Geliş Zamanı	Servis Zamanı	Öncelik	Bitme zamanı (Deadline)
A	0	3	En yüksek	3
B	2	6	En yüksek	6
C	4	4	En yüksek	4
D	6	5	En yüksek	5
E	8	2	En yüksek	2

First Come First Served (FCFS):

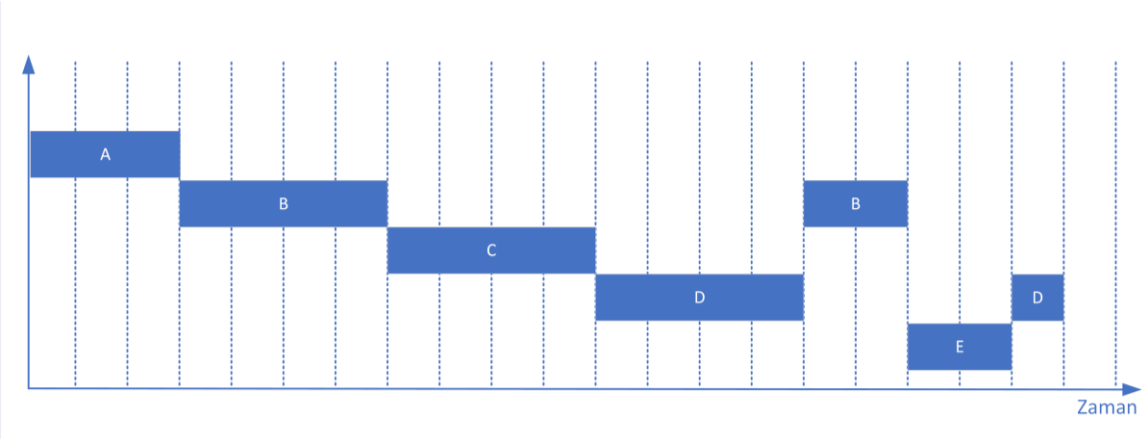
En basit görev zamanlayıcısı algoritmasıdır. Hazır olan görev, kuyruğa gelerek koşmaya başlar. İşlemci görevleri ilk giren ilk çıkar (FIFO) mekanizması çalıştırılır. Bir önceliklendirme (preemptive) ve öncelik (priority) bulunmaz (Şekil 5.2) [18].



Şekil 5.2 First Come First Served

Round Robin (RR):

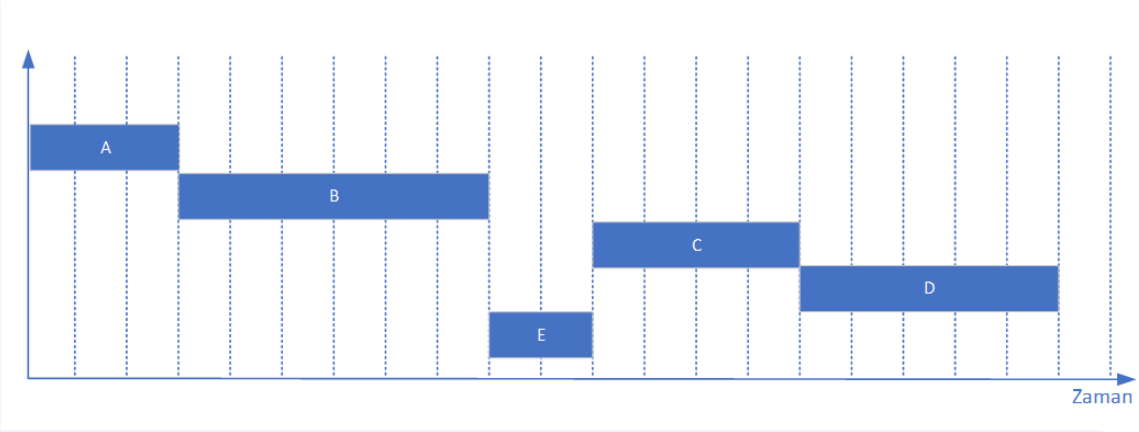
Zamanlama sorunlarını azaltmak için en çok kullanılan yöntemlerdendir. Hazır olan görevlerden önceliği en yüksek olan çalışır. Bunun yanında adil çalıştırma için her görevin, zaman dilimi (timeslice) [27] denilen bir çalışma süresi vardır. Önceliği en yüksek olan ve koşmaya başlayan görev, belirlenmiş zaman dilimi süresince koşar. Bu süre aşılınca görev durdurulur (preemptive) ve bir sonraki önceliği yüksek görev koşturulur. Bu görev bitince ve başka önceliği yüksek görev yok ise durdurulmuş olan görev kaldığı yerden çalışmaya devam eder. Bir önceliklendirme (preemptive) ve öncelik (priority) vardır (Şekil 5.3) [18].



Şekil 5.3 Round Robin, zaman dilimi=4

Shortest Process Next (SPN):

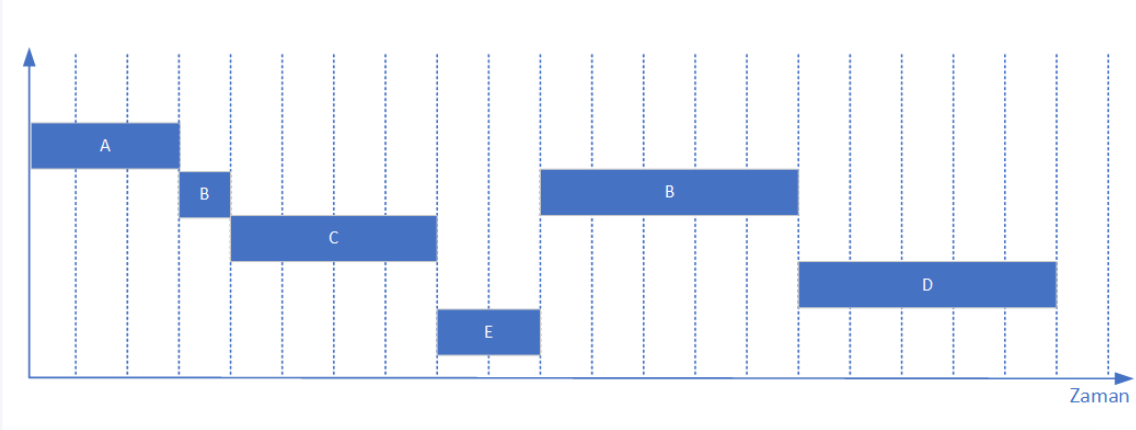
Bu görev zamanlayıcı çeşidinde hazır kuyruğuna girmiş en kısa görev çalıştırılır. Görev çalıştırmalarında ilk giren ilk çıkar (FIFO) mekanizması kullanılır. Bir önceliklendirme (preemptive) ve öncelik (priority) bulunmaz (Şekil 5.4) [18].



Şekil 5.4 Shortest Process Next

Shortest Remain Time (SRT):

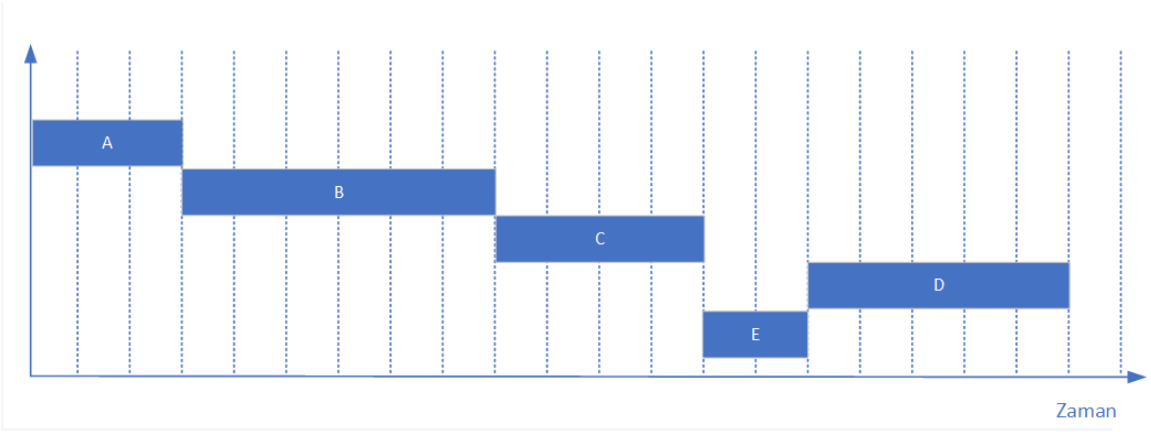
SRT pratik olarak SPN'nin önceliklendirilmiş (preempted) olanıdır. Görev zamanlayıcı çeşidinde hazır kuyruğuna girmiş en kısa görev çalıştırılır, eğer kuyruğa girmiş daha kısa görev var ise görev durdurulur ve bir sonraki kısa görev koşturulur. Bu görev bitince ve başka kısa görev yok ise durdurulmuş olan görev kaldığı yerden çalışmaya devam eder. Bir önceliklendirme (preemptive) vardır fakat öncelik (priority) bulunmaz (Şekil 5.5) [18].



Şekil 5.5 Shortest Remain Time

Highest Response Ratio Next (HRRN):

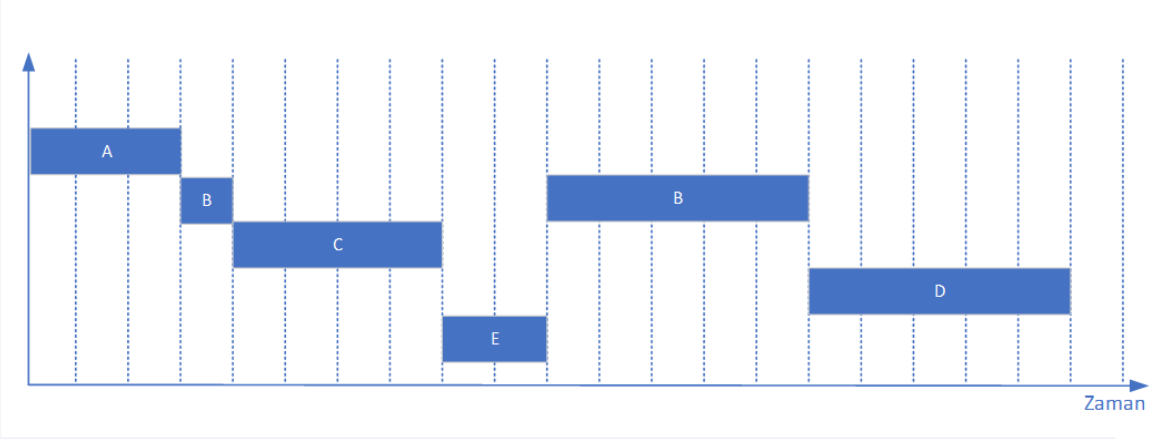
En yüksek cevap verme oranına bakan görev çalıştırma algoritmasıdır. Koşan görev biter bitmez veya durdurulduğunda hazır kuyruğundaki cevap verme oranı en yüksek olan görev çalıştırılır (Şekil 5.6) [18].



Şekil 5.6 Highest Response Ratio Next

Rate Monotonic (RM):

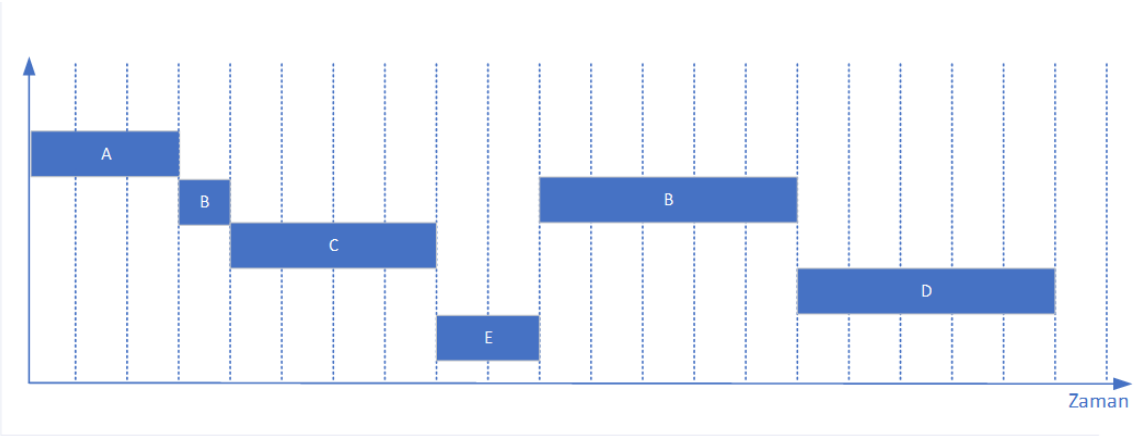
Frekans Monotonik de denilen bu görev zamanlayıcı işlemciyi kullanma süresine bakar, önceliği yüksek görevleri koşturur, öncelikleri aynı görevlerde ise periyodu kısa olan görevi ilk koşturur. Gömülü gerçek zamanlı sabit öncelikli sistemlerde çok tercih edilen bir görev zamanlayıcısıdır. Bir önceliklendirme (preemptive) ve öncelik (priority) vardır (Şekil 5.7) [2][4].



Şekil 5.7 Rate Monotonic

Earliest Dead Line First (EDF):

Dinamik öncelikli olarak bilinmektedir. Görevlerden en erken bitme zamanına (dead line) sahip olanı ilk çalıştıran görev zamanlayıcısıdır. Bir önceliklendirme (preemptive) ve öncelik (priority) vardır (Şekil 5.8) [3].



Şekil 5.8 Earliest Dead Line First

Interrupt and Event Driven (ED):

Olay (event) veya kesmeler (interrupt) ile sürülen görev zamanlayıcı yöntemidir. Bu yöntem, görevlerin ne zaman çalışacağının kararını tamamen tasarımcıya bırakır. Görevlerin çalışmaları kesmeler veya olaylar ile tetiklenir. Tasarımcının görev periyotlarına hakim olması ve çalışma çerçevesini aşmayacak şekilde

görevleri düzenlemesi gerekir. Bir görev, kesme veya olay ile tetiklendiğinde bir sonraki görev bir önceki görevin sonunda yeni bir olay gönderme (event_sent) ile tetiklenir. Bir sonraki görev bir önceki görevde gönderilen olayı olay alma (event_recieve) alır. Bu sayede art arda görev çalıştırması yapılır [9].

ED'yi işletim sisteminin diğer yöntemleri ile de yapmak mümkün olabilir. Bu yöntemlere aşağıda değinilmiştir [24]:

- Senkronizasyon - Semafor ve Muteks
- Kesme ve Olay Yönetimi (Interrupt and Event Handling) [7]
- Giriş/Çıkış (Input/Output)
- İç Görev iletişimi (Inter Task Communication)
- Zamanlama ve saat (Timers ve Clocks)
- Bellek Yönetimi (Memory Management)

Birçok görev zamanlayıcısı bu listede bulunmamaktadır. En çok kullanılan görev planlayıcılar listelenmiştir.

5.4 Görev Durumları

Hazır görev: Görev koşturulmaz çünkü daha yüksek öncelikli görevler koşturulmaktadır.

Bloklanmış görev: Görev kaynak ya da bir olay(event) bekler.

Koşan görev: Yüksek öncelikli görevdir ve o an çalışmaktadır.

Şekil 5.9 Görev Durumları

5.5 Görev Operasyonu

GZIS çekirdekleri, görev zamanlayıcısı servisleri sağlamaktadır. Burada amaç görevi yaratmak, çalıştırmak ve silmektir. Görev zamanlayıcıları rutin şekilde çağrılırlar. Bu rutinler genelde görevi askıya almak (suspend), bir gecikme sağlamak, devam etmek, önceliklerini değiştirmek, devreye almak, devreden çıkartmak için çekirdeğin sağladığı görev zamanlayıcı servislerini kullanmaktadırlar.

GZIS görevlerinin genel çalışma prensipleri, çalış tamamlan ve sonsuz döngüde çalış şeklindedir. Sonsuz döngüde çalışma görevlerinde askıya alma, bloklama gibi yöntemler kullanılmak zorunda kalınabilmektedir. Aksi halde düşük öncelikteki görevlerin hiç çalıştırılmama olasılıkları bulunmaktadır.

Görevlerin senkronizasyonu ve haberleşmesi için kullanılan bazı yapılar aşağıdaki verilmektedir;

- Semaforlar
- Muteks
- Mesaj kuyrukları
- Sinyaller

- Boru (pipe) yöntemleri

5.6 RTEMS Görevlerine (Task) Giriş

RTEMS işletim sistemi yukarıda bahsedilen GZIS'e ek olarak iki adet daha görev durumu barındırmaktadır.

RTEMS'in görev durumları [8];

Hazır görev: Görev koşturulmaz çünkü daha yüksek öncelikli görevler koşturulmaktadır.

Bloklanmış görev: Görev kaynak ya da bir olay(event) bekler.

Koşan görev: Yüksek öncelikli görevdir ve o an çalışmaktadır.

Uykuda görev (dormant): Henüz başlamamış ancak yaratılmış görevdir.

Var olmayan görev: Yaratılmamış veya silinmiş görevdir.

RTEMS ile 255 adet görev önceliği tanımlanabilmektedir. 1 en yüksek öncelik, 255 ise en düşük öncelik olarak kullanılır. Öncelik atamasında aynı sayıya atama yapmanın bir sınırı belirlenmemiştir [8].

RTEMS'de yapılandırma (configuration) işlemleri için kullanıcılara kolaylık olması adına makro yapılar kullanılmıştır. Bu makro yapılar açık kaynak kodlu olan RTEMS işletim sisteminin kapalı taraflarını oluşturmaktadır. Her RTEMS yapılandırması için bu makrolar çağırılmalıdır[8].

```
#define CONFIGURE_APPLICATION_NEEDS_CONSOLE_DRIVER
#define CONFIGURE_APPLICATION_NEEDS_CLOCK_DRIVER
#define CONFIGURE_MICROSECONDS_PER_TICK 1000 /* 1 millisecond
*/
#define CONFIGURE_TICKS_PER_TIMESLICE 50 /* 50 milliseconds */
#define CONFIGURE MAXIMUM TASKS 4
```

RTEMS'in ana fonksiyonu (main()) RTEMS'e has bir fonksiyondur init_task() olarak geçer.

```
rtems_task init_task (rtems_task_argument ignored);
```

```
name = rtems_build_name( 'A', 'P', 'P', '1' )
status = rtems_task_create(
name, 1, RTEMS_MINIMUM_STACK_SIZE,
RTEMS_NO_PREEMPT, RTEMS_FLOATING_POINT, &tid
);
if ( status != RTEMS_STATUS_SUCCESSFUL ) {
printf( "rtems_task_create failed with status of %d.\n", status
);
exit( 1 );
}
status = rtems_task_start( tid, user_application, 0 );
if ( status != RTEMS_STATUS_SUCCESSFUL ) {
printf( "rtems_task_start failed with status of %d.\n", status );
exit( 1 );
}
status = rtems_task_delete( SELF ); /* should not return */
printf( "rtems_task_delete returned with status of %d.\n", status
);
exit( 1 );
```

init_task altında her oluşturulacak görev için rutin yapılması gereken işlemler vardır. Bu işlemler; bir görev oluşturmak, başlatmak ve silmek için gereklidir [8].

Bir sonsuz döngüde kontrol algoritmalarımız koşacaktır, aşağıdaki kod parçasında while(1) ile sağlanır [29].

```
while ( 1 ) {                                                    /
infinite loop */
/* APPLICATION CODE GOES HERE
*
* This code will typically include at least one
* directive which causes the calling task to
* give up the processor.
*/
```

Bu çalışmada RTEMS'de koşan bütün görevler yukarıda belirtilen yapıda oluşturulmuş, başlatılmış ve silinmiştir.

5.7 RTEMS İle Yeni Bir Görev Zamanlayıcısı Geliştirmek

RTEMS ile bilinen görev planlayıcıların dışında yeni bir planlayıcı geliştirilmek istenirse, işletim sisteminin kabuğuna ve/veya çekirdeğine algoritmalar eklenmesi gerekmektedir [12]. Çeşitli yapılar kullanarak RTEMS'i tasarıma göre tekrar düzenlemek mümkündür. Bunun için aşağıda bulunan komutlar kullanılabilmektedir.

- `CONFIGURE_SCHEDULER_EDF`
- `CONFIGURE_SCHEDULER_CBS`
- `CONFIGURE_SCHEDULER_SIMPLE`
- `CONFIGURE_SCHEDULER_SIMPLE_SMP`

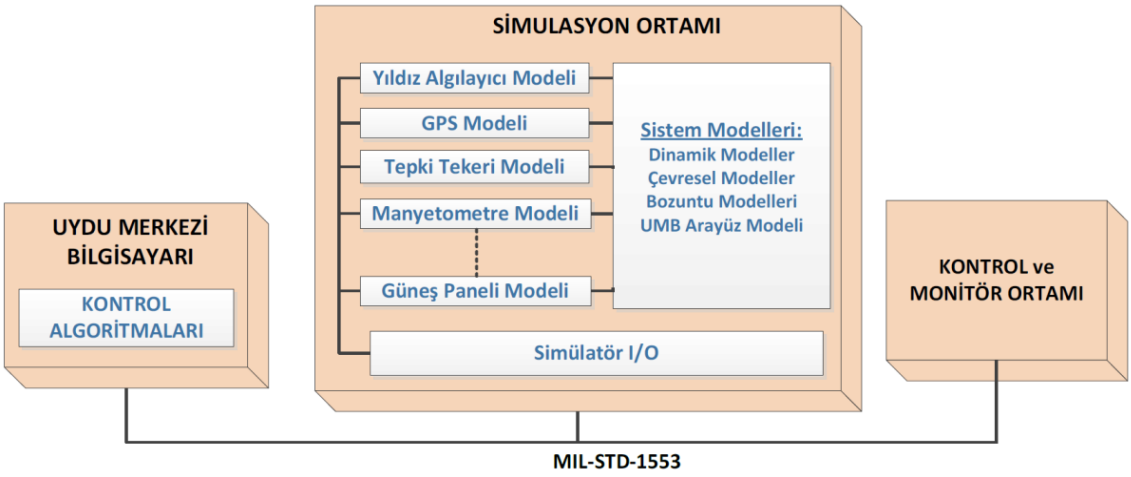
Bu komutlar ile görev zamanlayıcısı tasarlama imkanı bulunmaktadır. Tasarımcının işletim sistemlerine hakim ve yazılımcı yeteneğine sahip olması gerekmektedir, çünkü bu tür kodlar işletim sisteminin çekirdeğinde bulunmaktadır [12].

Bu konudaki zorluklar bilindiği için RTEMS'in hazır klasik API katmanı görev zamanlayıcıları kullanılarak bir karşılaştırma yapılması tercih edilmiştir.

6 YÜRÜTÜLEN ÇALIŞMALAR

Bu çalışmanın odaklandığı konu uydu uçuş bilgisayarında koşan denetleyicilerin en iyi performansı verebilmeleri için görev zamanlayıcıların karşılaştırılmasıdır. Uydu uçuş bilgisayarında koşan denetleyicilerin kontrol ettikleri uyduların gerçek ortamlarının uzay olması, bizi uzay ortamını benzetimler ile oluşturmak durumunda bırakmıştır. Yukarıda anlatılan bütün bölümler, simülasyon ortamı (Şekil 6.1) ve bu ortamı oluşturan parçalar, teknikler, standartlardır.

Uzay ortamını ve ekipmanlarını simülasyon çerçevesi ile benzetirken, gerçek zamanlı işletim sisteminde görevleri koşturarak bir uydunun uzaydaki performansını en yüksek seviyeye çıkarmak hedeflenmektedir.



Şekil 6.1 Hibrit Sistem Altyapı İçeriği

UMB'de [4.1.2] koşan algoritmaları test edebilmek için simülasyon ortamı (Bölüm 4.3) ile bir seri veri yolu olan MIL-STD-1553 (Bölüm 4.2) kapalı döngü çalıştırılmaktadır. Çalışma ve test ortamının ayrıntıları devam eden bölümlerde anlatılmaktadır.

6.1 Uzay Ortamı ve Uydu Ekipmanlarının Simülasyon Çerçevesinde Benzetilmesi (Simulation Framework)

Bu çalışmanın konusu olan görev zamanlayıcıların performanslarına yönelik karşılaştırma yapabilmek için uzay araçlarının benzetimine ihtiyaç duyulmuştur. Bu benzetimleri yapabilmek için ise EuroSim (Bölüm 4.3) [16] simülasyon

çerçevesinde koşan SMP2 (Bölüm 4.3) [17] uyumlu modeller maddeler halinde aşağıda verilmiştir:

- Yıldız algılayıcı modeli; bir yıldız haritasına bakarak uydunun konumunu belirleyen ekipmanın benzetim modelidir.
- Güneş sensörü modeli; güneşten gelen ışığı kullanarak yön ve/veya konum belirleyen ekipmanın benzetim modelidir.
- Manyetometre modeli; dünyanın manyetik alanını ölçerek manyetik tork çubuklarını sürmek için kullanılan ekipmanın benzetim modelidir.
- GPS modeli; konumlama uydularından gelen sinyaller ile konum tespitinde bulunan ekipmanın benzetim modelidir.
- Manyetik tork çubuğu modeli; dünyadaki manyetik alanını kullanarak uyduya yön veren eyleyicilerin ekipmanın benzetim modelidir.
- Tepki tekeri modeli; hızlanarak ve yavaşlayarak uydunun yönünü değiştiren eyleyicilerin ekipmanın benzetim modelidir.
- İtki sistemi modeli; bir tankta bulunan gaz sayesinde uydunun yönünü değiştiren ekipmanın benzetim modelidir.
- Haberleşme alt sistem modeli; uydu ile dünya arsında haberleşmeyi sağlayan ekipmanın benzetim modelidir.
- Güç alt sistem modeli; uyduya güç sağlayan güneş panellerinin ve gücün dağıtımını modelleyen ekipmanın benzetim modelidir.
- Uzay ortamı modeli; uydunun yörüngesi, güneş, ay ve manyetik alanı modelleyen ortam benzetim modelidir.
- Uydu dinamiği modeli; uydunun pozisyonunu ve uzaydaki yerini modelleyen ortam benzetim modelidir.

Simülasyon çerçevesinde UMB hariç bir uydunun bütün parçaları kendi görev bilgilerini yerine getirerek tam bir uydu gibi iç haberleşme sağlamaktadırlar. Ayrıca sistem 10 Hertz ile çalışmaktadır. UMB ile haberleşmek için MIL-STD 1553 veri yolu bir PCI haberleşme kartı (AIM-APE1553-4) [14] ile fiziksel olarak

sağlanmaktadır (Bölüm 4.3). Sistemde UMB ile haberleşme modeli bulunur, bu model gerekli verilerin gerekli modellere dağıtımını yapmaktadır.

6.2 UMB'de Koşan Görevler ve Görev Zamanlayıcıların Bu Görevler Üzerinde Performans Karşılaştırılması (Tartışma)

Uzay araçlarının zorlu çalışma ortamları, fırlatma öncesi uydunun sistem seviyesi doğrulamasını gerektirmektedir. Fırlatma öncesi doğrulama yöntemleri Bölüm 2'de verilmiştir. Bu çalışmada doğrulama altyapısı olarak hibrit sistem altyapısı (Bölüm 2.5) kullanılmıştır. Çalışma altyapısında gömülü gerçek zamanlı uçuş bilgisayarı olarak Leon3FT (Bölüm 4.1.1) işlemcili UMB (Bölüm 4.1), gerçek zamanlı işletim sistemi olarak da RTEMS (Bölüm 3.3) işletim sistemi kullanılmıştır. Simülasyon çerçevesi ve UMB arasında veri iletimi MIL-STD1553 (Bölüm 4.2) ile yapılmıştır.

Gerçek zamanlı işletim sistemi olan RTEMS'de koşacak görev zamanlayıcıları (Bölüm 5.3) arasından tezin konusu olan karşılaştırmayı yapabilmek için üç adet görev zamanlayıcı seçilmiştir, bunlar:

- Round Robin (RR)
- Rate Monotonic (RM)
- Event Driven (ED)

Yukarıda adı geçen görev yöneticilerinin seçilmesindeki motivasyon, tek işlemcili sistemlerde koşmaları, sabit öncelik [4] ile çalışıyor olmaları ve en önemlisi RTEMS işletim sisteminde uygulanmaları diğer görev zamanlayıcılarına göre kolay olmalarıdır. Diğer görev zamanlayıcılarının RTEMS'in kabuk yazılımlarına girilmeden uygulanması zor bulunmuştur [11].

Bu görev yöneticilerini test etmek için iki adet uydu kontrolcüsü seçilmiştir ve aşağıda bu görevler verilmiştir. Görevlerin işlemci kullanımına göre karşılaştırması yapılmıştır.

- YBKS algoritması: Uydudan gelen sensör verilerini alıp, yönelim ile ilgili algoritmaları çalıştırıp, eyleyicilere komut gönderen kontrolcüdür. Uydunun durum ve kiplerine göre ikiye ayrılmaktadır.
    - Hassas yönelim

- Kaba yönelim

- 1553 haberleşme algoritması: MIL-STD 1553 standardı haberleşme veri yolu için oluşturulan görevdir. Bu görev, veri yolunun fiziksel bir kart ile haberleşmesinin ve donanım destek yazılım paketi (Bölüm 4.1.2) kullanılan diğer denetleyicilerde veri trafiğinin kontrolünün sağlandığı kontrolcüdür.

Round Robin (Bölüm 5.3) ile görevlerin çalıştırılması:

RTEMS işletim sistemi ile YBKS ve 1553 görevleri sistemde çalıştırılmıştır.

RTEMS komutu:

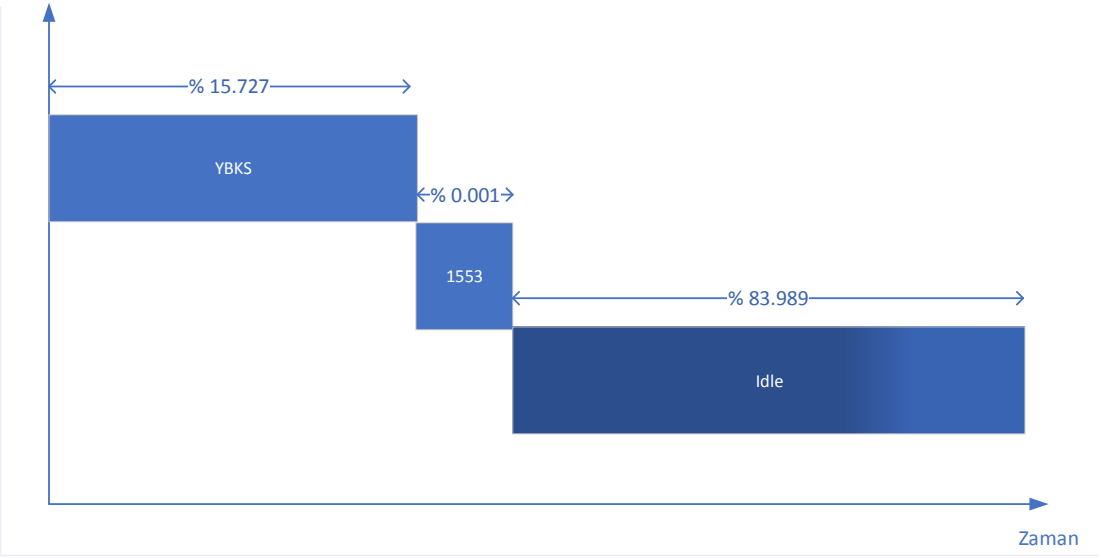
```
#define CONFIGURE_TICKS_PER_TIMESLICE 100 /* 100 milliseconds */
```

Zaman dilimi (time slice) 100 ms olarak tanımlanmıştır. 10 Hertz ile çalışan sistemde her adım 100 ms'lik sürede çalışır. Zaman dilimi ise bu zamanın tümü olarak tanımlanmış olur. Bu sayede görevler arasında anahtarlama (switching) maliyetleri azaltılmış olur.

```
rtems_task_wake_after(10);
```

10 Hertz ile RR çalıştırmak için yukarıdaki RTEMS komutu çağrılmaktadır. 1000 defa çalıştırılma sonrası sonuç çıktıları aşağıda verilmiştir.

```
------------------------------------------------------------------------------
                            CPU USAGE BY THREAD
------------+-----------------------------------+---------------+---------
 ID         | NAME                              | SECONDS       | PERCENT
------------+-----------------------------------+---------------+---------
 0x09010001 | IDLE                              |     92.654899 |  83.989
 0x0A010002 | 1553                              |      0.001481 |   0.001
 0x0A010003 | AOCS                              |     17.350450 |  15.727
------------+-----------------------------------+---------------+---------
 TIME SINCE LAST CPU USAGE RESET IN SECONDS:                     110.317556
------------------------------------------------------------------------------
```



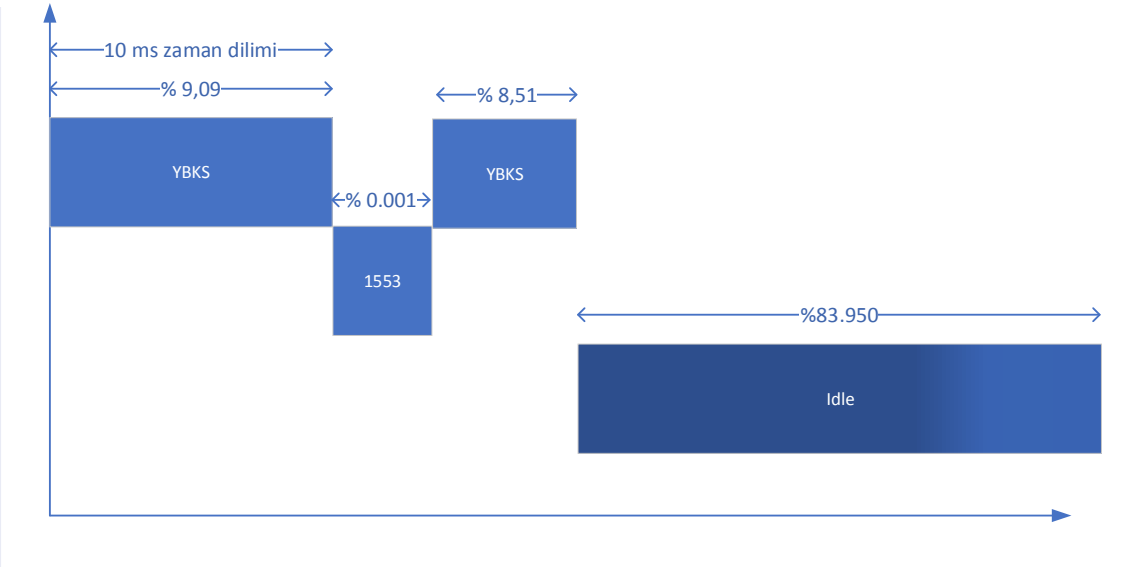
Şekil 6.2 Round Robin ile Görevlerin Koşturulması

RR görev zamanlayıcı ile çalışmadaki zaman dilimi (timeslice) en yüksek aralıkta tanımlanmıştır. Diğer yandan bütün görevlerin öncelikleri aynı ve en yüksek önceliğe sahiptir. Önce YBKS görevi çalışır, ardından 1553 görevi çalışır. Kodu düzenlerken ilk YBKS görevi yaratıldığı için ilk çalışan görev YBKS olmaktadır. Ayrıntılı kodlar EK1'de verilmiştir.

```
#define CONFIGURE_TICKS_PER_TIMESLICE 10 /* 10 milliseconds */
```

Zaman dilimi (time slice) 10 ms tanımlanır ise ~17.61 ms ile çalışan YBKS görevi, zaman dilimi kısıtlaması yüzünden önceliklendirilerek (preemptive) çalıştırılacaktır (Şekil 6.3). Görevler arasında anahtarlama (switching) maliyetleri daha fazla olacaktır.

```
------------------------------------------------------------------------------

                            CPU USAGE BY THREAD

------------+---------------------------------------+---------------+---------
 ID         | NAME                                  | SECONDS       | PERCENT
------------+---------------------------------------+---------------+---------
 0x09010001 | IDLE                                  |     92.652899 |  83.950
 0x0A010002 | 1553                                  |      0.001490 |   0.001
 0x0A010003 | AOCS                                  |     17.612450 |  15.965
------------+---------------------------------------+---------------+---------
 TIME SINCE LAST CPU USAGE RESET IN SECONDS:                        110.26683
------------------------------------------------------------------------------
```



Şekil 6.3 Round Robin ile Görevlerin Koşturulması (Preemptive

Rate Monotonic (Bölüm 5.3) ile görevlerin çalıştırılması:

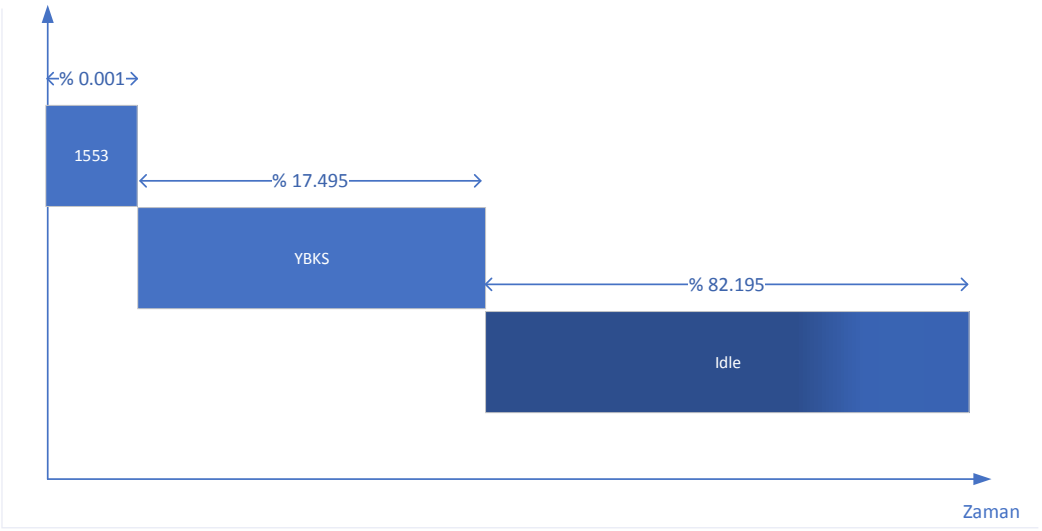
RTEMS işletim sistemi ile YBKS ve 1553 görevleri sistemde çalıştırılmıştır.

RTEMS komutu:

```
if ( rtems_rate_monotonic_period( RM_period ,10) ==RTEMS_TIMEOUT
)
      break;
```

10 Hertz ile RM çalıştırmak için yukarıdaki RTEMS komutu çağrılmaktadır. 1000 defa çalıştırıldıktan sonra sonuç çıktıları aşağıda verilmiştir.

```
-------------------------------------------------------------------------------
                                CPU USAGE BY THREAD
------------+-----------------------------------------+---------------+---------
 ID         | NAME                                    | SECONDS       | PERCENT
------------+-----------------------------------------+---------------+---------
 0x09010001 | IDLE                                    |     82.599354 |  82.195
 0x0A010002 | 1553                                    |      0.001494 |   0.001
 0x0A010003 | AOCS                                    |     17.581507 |  17.495
------------+-----------------------------------------+---------------+---------
 TIME SINCE LAST CPU USAGE RESET IN SECONDS:                          100.490895
-------------------------------------------------------------------------------
```



Şekil 6.4 Rate Monotonic ile Görevlerin Çalıştırılması

RM görev zamanlayıcı ile çalışmadaki bütün görevlerin öncelikleri aynı ve en yüksek önceliğe sahiptir. Önce 1553 görevi çalışır, ardından YBKS görevi çalışır. RM özellikle aynı öncelikte olan görevlerin en kısa olanını önce çalıştırır. RTEMS işletim sisteminde kabukta bulunan RM mekanizmasının periyodu en yüksek olan görevleri çalıştırmaya yönelik bir algoritması vardır. RM görev zamanlayıcısı çok görevli birden fazla öncelik ve hızda çalışan sistemler için daha etkili bir yöntemdir. Ayrıntılı kodlar EK2'de verilmiştir.

Event Driven (Bölüm5.3) ile görevlerin çalıştırılması:

RTEMS işletim sistemi ile YBKS ve 1553 görevleri sistemde çalıştırılmıştır.

RTEMS komutu:

1553 kartından gelen kesme

```
     gr1553bc_slot_irq_prepare(list,    mid,
               tmtc_1553_tc_read_isr,
               (void *)mid );
```

Kesme ile çağrılan fonksiyon ve olay gönderme, burada bir olayı tetiklemiştir.

```
void tmtc_1553_tc_read_isr(union gr1553bc_bd  *bd, void
*data)
{

     isr_count_tc_read_isr++;

     /* poke T_1553_PROCESSOR task */
     rtems_event_send( Task_id[T_1553_PROCESSOR],
EVENT_1553_START );

}
```

Olay alma burada ilk 1553 görevini çalıştırmaktadır.

```
     while(1)
     {
          /* wait for isr to poke */
          rtems_event_receive( EVENT_1553_START,
                    RTEMS_WAIT|RTEMS_EVENT_ALL,
RTEMS_NO_TIMEOUT, &rc_event);
```

While(1) döngüsünden çıkmadan önce kodun sonunda YBKS için olay gönderilmektedir.

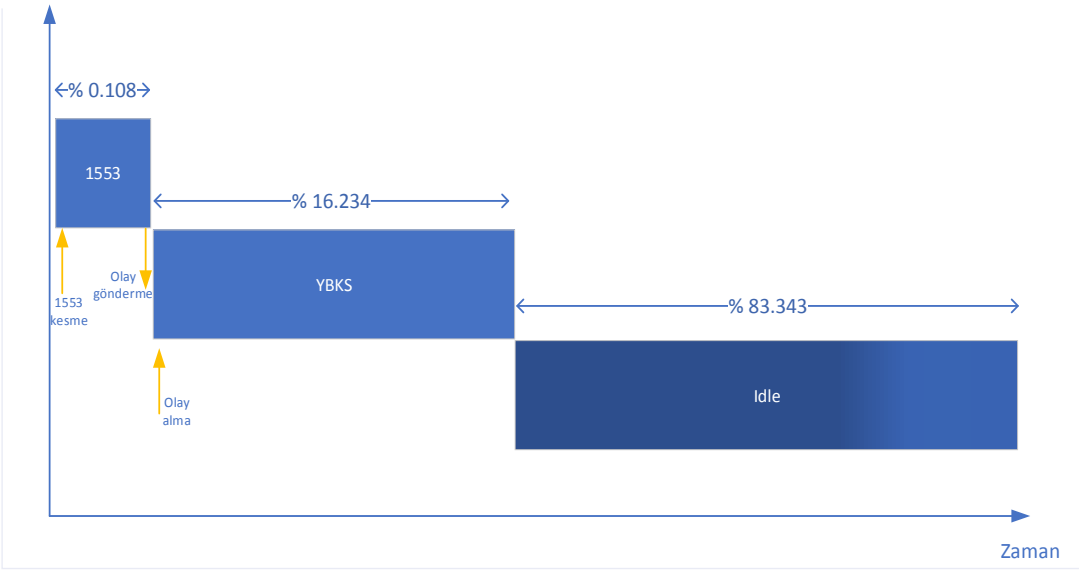
```
rtems_event_send(
Task_id[T_AOCS_NORMAL],EVENT_AODCS_NORMAL );
```

Gönderilen olay mekanizması YBKS görevini tetikler.

```
     while(1)
     {
          rtems_event_receive( EVENT_AODCS_NORMAL,
                    RTEMS_WAIT|RTEMS_EVENT_ALL,
RTEMS_NO_TIMEOUT, &rc_event);
```

10 Hertz ile ED çalıştırmak için yukarıdaki RTEMS komutları çağrılmaktadır. 1000 defa çalıştırıldıktan sonra sonuç çıktıları aşağıda verilmiştir.

```
------------------------------------------------------------------------------
                            CPU USAGE BY THREAD
------------+-----------------------------------------+--------------+---------
 ID         | NAME                                    | SECONDS      | PERCENT
------------+-----------------------------------------+--------------+---------
 0x09010001 | IDLE                                    |     82.776076 |  83.342
 0x0A010002 | 1553                                    |      0.107517 |   0.108
 0x0A010003 | AOCS                                    |     16.124382 |  16.234
------------+-----------------------------------------+--------------+---------
 TIME SINCE LAST CPU USAGE RESET IN SECONDS:                         99.320855
------------------------------------------------------------------------------
```



Şekil 6.5 Event Driven ile Görevlerin Çalıştırılması

ED görev zamanlayıcı, çalışmadaki 1553 kartından gelen kesme ile 1553 adlı görevi çalıştırmış, bu görevin sonunda bir olay gönderme mekanizması kullanılmış ve YBKS kodunun başında bulunan olay alma mekanizması ile YBKS görevi çalıştırılmıştır. ED görev zamanlayıcısı yöntemi ile tasarımcı istenilen sırada görevleri uç uca ekleyerek çalıştırabilir. Bu durum uydu denetleyicisi üzerinde, önem sırasına göre görevleri çalıştırma esnekliği verir. Diğer görev zamanlayıcıları bir görevi sadece bir defa çalıştırabilirken ED görev zamanlayıcısı ile istenilirse aynı görev birkaç defa çalıştırılabilmektedir. Ancak bu yöntem çok değerli olan işlemci zamanını aynı işi yapmak için iki defa kullanmak anlamına gelir ve tercih edilen bir durum değildir. Görev sayısı arttıkça ve kısa görevler çok sayıda oldukça anahtarlama maliyetleri hissedilir bir seviyeye çıkacaktır. Bu yöntemin en büyük dezavantajı anahtarlama maliyetleridir. Bunun yanında RTEMS 32 adet olay gönderme – alma mekanizması ile sınırlıdır ve görev sayısı arttıkça 32 adet olay gönderme – alma mekanizması bir kısıt olarak karşımıza çıkacaktır. Ayrıntılı kodlar EK-3'de verilmiştir.

Çalışmanın sonuçları Tablo 6-1'de özetlenmiştir.

Tablo 6.1 Görev Zamanlayıcıların İşlemci Kullanım Yüzdeleri (RR zaman dilimi 100 ms)

Görevler	Görev zamanlayıcıları		
	RR%	RM%	ED%
Idle	83.989	82.195	83.342
YBKS	15.727	17.495	16.234
1553	0.001	0.001	0.108

Yukarıda verilen tabloda görev zamanlayıcıların görevlere göre yüzdeleri bulunmaktadır. İşlemci kullanımına göre görev zamanlayıcıların performansına bakıldığında sonuçların birbirlerine çok yakın olduğu gözlemlenmiştir (Tablo 6.1). RTEMS işletim sistemi görev zamanlayıcılar politikasında, varsayılan görev zamanlayıcısı RR olarak çalışmaktadır. Bu yüzden tüm görevler için diğer görev zamanlayıcılarına göre daha iyi sonuç vermektedir. RM kısa görevlerde daha etkili olduğu bilinmektedir. Bu sebepten 1553 görevi için RR kadar iyi sonuç vermiştir. ED görev zamanlayıcısı anahtarlama maliyetleri sebebi ile işlemci kullanım yüzdeleri yüksek çıkmıştır.

Tablo 6.2 Görev Zamanlayıcıların İşlemci Kullanım Yüzdeleri (RR zaman dilimi 10 ms)

Görevler	Görev zamanlayıcıları		
	RR%	RM%	ED%
Idle	83.950	82.195	83.342
YBKS	15.965	17.495	16.234
1553	0.001	0.001	0.108

Yukarıda anlatılanlara ek olarak RR görev zamanlayıcısı zaman dilimi 10 ms alınmış ve önceliklendirmeye (preempitive) maruz kalmıştır. RR sütununda işlemci kullanımı yüzdesi YBKS görevi için artığı gözlemlemektedir. Bunun sebebi anahtarlama maliyetlerinin artması olarak değerlendirilmektedir.

Tablo 6.3 Görev Zamanlayıcıların karşılaştırılma matrisi

	Uygulama kolaylığı	**Önceliklendirme (preempitive)**	**Kısa görevlerde etkinliği**	**Karmaşık görevlerde başarımı**	**İşletim sistemi kısıtı**	**Tasarımcıya sağladığı kontrol**
RR	Kolay	Yüksek	Normal	Normal	Yok	Normal
RM	Zor	Mevcut değil	Yüksek	Yüksek	Yok	Zayıf
ED	Zor	Mevcut değil	Normal	Zayıf	Var	Yüksek

7 SONUÇLAR

Bu tezde, uzay araçları hibrit sistem altyapısında gerçekleştirilen çalışmada iki adet görev, üç görev zamanlayıcı üzerinde çalıştırılmıştır. Bu iki görev uydu kontrolünde büyük önem arz etmektedir ve birbirlerini tamamlayan görevlerdir. Yönelimi yapmak için YBKS görevine ihtiyaç vardır ve bu komutları iletmek için 1553 görevine ihtiyaç duyulmaktadır. Bu sebepten iki görev de aynı öncelik ile çalışmaktadırlar.

İşlemci kullanımına göre görev zamanlayıcıların performansına bakıldığında sonuçların birbirlerine çok yakın olduğu gözlemlenmiştir (Tablo 6.1). Bölüm 6.2'de sonuçlar verilmiştir. Söz konusu üç görev zamanlayıcısının koşullara göre avantajları bulunmaktadır.

RR görev zamanlayıcısı; uygulaması kolay, zaman dilimi uygun verilirse yüksek performanslı bir görev zamanlayıcısıdır. Görev sürelerine bakarak en uygun zaman dilim seçimi yapılmalıdır. Bu çalışmada işlemci görevleri çalıştırdıktan sonra boşta zaman kaldığı için zaman dilimi en büyük atanmıştır. RTEMS işletim sistemi görev zamanlayıcılar politikasında, varsayılan görev zamanlayıcısı RR olarak çalışmaktadır. Bu sebepten RR en az yapılandırma ile uygulaması en kolay görev zamanlayıcısıdır.

RM'nin kısa görevler üzerinde etkili olduğu değerlendirilmekte fakat bu çalışmada bu etkiyi sağlayacak görevler bulunmamaktadır. İleride yapılacak çalışmalarda daha kısa görevlerin olacağı ve RM görev zamanlayıcısının daha etkin bir sonuç vereceği düşünülmektedir. Sonuçlardan anlaşıldığı kadarıyla farklı çalışma hızlarında çalışan görevlerin (örneğin bazı görevler 10 Hz. ile çalışırken bazı görevler 30 Hz. çalışıyor) işlemciye gelme zamanları ve periyotları karmaşıklaşacaktır. Çok sayıda görev ve öncelik olduğu düşünülürse görevlerin çalışma takibi daha zor olacaktır. RM görev zamanlayıcısının çok görevli birden fazla öncelik ve hızda çalışan sistemler için etkili olacağı değerlendirilmektedir.

ED, görev zamanlayıcısı tasarımcısına tam kontrol verdiği için tercih edilmektedir. Uydu görevlerinde tasarımcılar her şeyi planlayıp kontrol etmek istemektedirler.

Ancak ED görev zamanlayıcısının uygulaması RR'ye göre daha zordur. Bunun yanında çok görevli sistemlerde anahtarlama maliyeti ve RTEMS işletim sisteminin olay gönderme-alma mekanizmalarının 32 adet olması bir kısıt olarak ortaya konmaktadır.

RTEMS işletim sisteminde görev zamanlayıcıları üzerinde iki görev ile yapılan bu çalışma ileride görev sayısı arttıkça nasıl davranılması gerektiğini anlamak için temel bir çalışma olmuştur. Uydu denetleyicilerinde görevlerin çalıştırılması sırasının öngörülmesi gereken sıralama, çalışma süresi, öncelik ve zaman dilimi gibi karakteristik özelliklerin seçiminde bu çalışma örnek alınacak daha iyi bir görev zamanlayıcısı tasarımı için ışık tutacaktır.

Yapılan çalışmada öne çıkan görev zamanlayıcıları RR ve ED olarak belirlenmiştir. RR görev zamanlayıcısının kolay uygulanması, ED görev zamanlayıcısının tam kontrol sağlaması karşılaştırma sonucunda belirleyici olmuştur.

Gelecek çalışmalarda uydu görev denetleyicilerinde ısıl kontrol görevi ekleyerek hem işlemci performansı hem de hafızada kapladığı yer [26] performansı ile karşılaştırma yapılması hedeflenmektedir.

KAYNAKLAR LİSTESİ

[1] J. Eickhoff, Simulation Tools for System Analysis and Verification in Simulating Spacecraft Systems, Springer, 1.basım, New York, 2009.

[2] C. Liu ve J. Layland, Scheduling algorithms for Multiprogramming in a Hard Real-Time Environment, Journal of the ACM, 20(1), s.46–61, 1973.

[3] F.F. Lindh, T. Otnes, and J. Wennerstrom, Scheduling algorithms for real-time systems, Sch. Comput. Queen's Univ. Tech., 2005.

[4] M. Bashiri, S. Ghassem, Performability Comparison of Schedulability Conditions in Real-Time Embedded Systems, 2010 Third International Conference on Dependability, s.70–75, 2010.

[5] Internet: On board data handling, ESA online, 2014, http://www.esa.int/ Our_Activities/ Space_Engineering_Technology/ Onboard_Computer_and _Data_Handling/Microprocessors.

[6] Internet: On board data handling, ESA online, 2014, http://www.esa.int/ Our_Activities/ Space_Engineering_Technology/ Onboard_Computer_and _Data_Handling/ Mil-STD-1553".

[7] K. Ramamritham, J.A. Stankovic, Scheduling Algorithms and Operating Systems Support for Real-Time Systems, IEEE Xplore, 1994.

[8] Internet: RTEMS C User's Guide, RTEMS documentation, 2017, https://docs.rtems.org/releases/rtems-docs-4.11.2/c-user/index.html.

[9] M. Coutinho, J. Rufino, ve C. Almeida, Control of event handling timeliness in RTEMS. Proceedings of the 17th IASTED International Conference on Parallel and Distributed Computing Systems - PDCS 2005, Phoenix, Arizona, USA, 2005.

[10] J. W S Liu, Real Time Systems, Prentice Hall, New Jersay, s.26, 2000.

[11] M. Molnar, The EDF scheduler implementation in RTEMS Operating System, M.Sc. Thesis Czech Technical University in Prague Faculty of Electrical Engineering, Prague, s.35, 2006.

[12] G. Bloom, J. Sherrill,Scheduling and Thread Management with RTEMS,EWiLi, Toulouse FRANSA, 2013.

[13] Internet: Gaisler Lib, Gaisler downloads, 2018, https://www.gaisler.com/j25/index.php/downloads/leongrlib.

[14] Internet: AIM MIL-STD 1553 ,AIM products documents ,2018, https://www.aim-online.com/products/ape1553-x/.

[15] Internet: SimSat , ESA online, 2014, http://www.esa-tec.eu/space-technologies/from-space/real-time-simulation-infrastructure-simsat/.

[16] Internet: Eurosim , Eurosim sayfasında, 2017, http://www.eurosim.nl/.

[17] Internet: SMP 2, ESA online, 2014, http://www.esa.int/TEC/Modelling_and_simulation/TEC2DCNWTPE_0.html.

[18] William Stallings, Operating Systems Internals And Design Principle, Prentice Hall, 7.basım, New Jersey, s.395, 2012.

[19] G. Buttazzo, M. Bertogna ve G. Yao, Limited Preemptive Scheduling for Real-Time Systems. A Survey, IEEE Transactions On Industrial Informatics Vol.9 No.1,2013.

[20] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien, Controller area network (CAN) schedulability analysis. Refuted, revisited and revised, Real-Time Syst. vol.35, no.3, s.239–272, 2007.

[21] I. Lee , J. Leung ve S. Son ,Handbook of Real-Time and Embedded Systems., Chapman & Hall/CRC Computer & Information Science Series, 2008.

[22] A. Colin, I. Puaut, Worst-case execution time analysis of the RTEMS real-time operating system, Proceedings 13th Euromicro Conference on Real-Time Systems, s.191, 2001.

[23] T. Straumann,Open Source Real Time Operating Systems Overview, eConf C011127 (2001) WEBT001, 2001.

[24] P. Tabuada, Event-Triggered Real-Time Scheduling of Stabilizing Control Tasks, IEEE TRANSACTIONS ON AUTOMATIC CONTROL, VOL. 52, NO. 9, 2007.

[25] D. Cederman, D. Hellstrom, J. Sherrill, G. Bloom,M. Patte, M. Zulianello, RTEMS SMP for LEON3/LEON4 Multi-Processor Devicies, DASIA 2014 - DAta Systems In Aerospace, Proceedings of the conference, 2014.

[26] P. Jinzhan, Improve on Memory Management of RTEMS system and Debug Technical,Beijing University of Aeronautics and Astronautics, 2002.

[27] A. R.Dash, S. K.Sahu, S. K.Samantra, An Optimized Round Robin CPU Scheduling Algorithm with Dynamic Time Quantum, International Journal of Computer Science Engineering and Information Technology (IJCSEIT), vol.5, no.1, 2015.

[28] Y. Jiang , Design and optimization of multiclocked embedded systems using formal techniques, IEEE Trans. Ind. Electron., vol. 62, no. 2, s.1270-1278, 2015.

[29] V. Melissa, O. Luciano, A. M. M. Cesar, R. Carlos ve H. Fabiano, RTOS scheduler implementation in hardware and software for real time applications, in Proc., 17th IEEE Int. Workshop Rapid Syst.Prototyping,s.163–168, 2006.

[30] J. Gaisler, E. Catovic ,Multi-core processor based on leon3-ft ip core (leon3-ft-mp) ,DASIA 2006-Data Systems in Aerospace, 2006.

[31] F. Sturesson, J. Gaisler, R. Ginosar, T. Liran, Radiation characterization of a dual core LEON3-FT processor, 2011 12th European Conference on Radiation and Its Effects on Components and Systems,2011.

[32] S. Habinc, J. Gaisler, C. Monteleone, C.Taylor, ESA's Spacecraft Avionics Development Platform - RASTA, Proceedings of DASIA 2008 Data Systems In Aerospace, 2008.

[33] G. Furano, F. Guettache,G. Magistrati, G. Tiotto, C. Ortega, A. Valverde, CanOpen on RASTA: The Integration of the CanOpen IP Core in the Avionics Testbed , Proceedings of DASIA 2013 DAta Systems In Aerospace, 2013.

[34] J. Windsor, K. Eckstein, P. Mendham, T. Pareaud, Time and space partitioning security components for spacecraft flight software, 2011 IEEE/AIAA 30th Digital Avionics Systems Conference ,2011.

[35] A. Sakthivel, J. Ekergarn, D. Hellstrom, S. Habinc, M. Suess, Spacewire time distribution protocol implementation and results, 2014 International SpaceWire Conference (SpaceWire) , 2014.

[36] L. Bremer, V. F. Garcia, M. Neefs, A.A. Diaz, The Euclid verification facilities ,Workshop on Simulation for European Space Programmes (SESP 2017), 2017.

EKLER LİSTESİ

EK-1 Round Robin

Kod;

```
/*
 * AOCS_proc.c with RR
/*
 * thermal_control_system_proc.c
 *
 *  Created on: 09 Feb 2015
 *      Author:MEG
 */

#include <rtems.h>
/* configuration information */

#include "rtems_obc_demo.h"
#include "utils.h"
#include "obc_config.h"
#include "AOCS_Normal.h"

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <gr1553bc.h>

/*********************************TCA_RTW*********************************/
#include "./AOCS/AOCS_Const.h"
/*********************************REALTIME*********************************/
/********************************** Buffer for RealTime ******************/
extern uint16_t SRX_RT2BC_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t SRX_RT2BC_BUFF2[32] __attribute__ ((aligned(8)));
extern uint16_t SRX_RT2BC_BUFF3[32] __attribute__ ((aligned(8)));
extern uint16_t SRX_RT2BC_BUFF4[32] __attribute__ ((aligned(8)));
extern uint16_t SRX_RT2BC_BUFF5[32] __attribute__ ((aligned(8)));
extern uint16_t SRX_BC2RT_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t PAB_RT2BC_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t PAB_BC2RT_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t RW_RT2BC_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t RW_BC2RT_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t MTQ_RT2BC_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t MTQ_BC2RT_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t PROP_RT2BC_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t PROP_BC2RT_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t MTM_RT2BC_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t MTM_BC2RT_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t SS_RT2BC_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t SS_RT2BC_BUFF2[32] __attribute__ ((aligned(8)));
extern uint16_t SS_BC2RT_BUFF[32] __attribute__ ((aligned(8)));

extern uint16_t PCDU_RT2BC_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t PCDU_BC2RT_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t THERM_RT2BC_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t THERM_BC2RT_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t GPS_RT2BC_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t GPS_BC2RT_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t GYRO_RT2BC_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t GYRO_BC2RT_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t STR_RT2BC_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t STR_BC2RT_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t STX_RT2BC_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t STX_BC2RT_BUFF[32] __attribute__ ((aligned(8)));
/*****************************************************************************************/

int AOCS_count=0;
double Qref[4]; //last Q reference of satellite
```

```
double Wref[3]; //last W reference of satellite


struct TcMtq MTQsend;
struct TcRW RWSend;
struct TcGyro  GyroSend;
struct TCSatMode SATModeSend;
struct TmSTR StrRecv;
struct TmGPS GpsRecv;
struct TmSS SSRecv;
struct TmSSonlyY SSyRecv;
struct TmMTM MtmRecv;
struct TmGyro GyroWRecv;
struct TmRW RWTachoRecv;
struct TmGSLVLH GSLVLHRecv;
struct TmGSEci GSEciRecv;
struct TmGSPolyID GSPolyIDRecv;
struct TmGSPolyDiff GSPolyDiffRecv;
struct TmGSMode SatModeRecv;



rtems_task AOCS_normal(rtems_task_argument arg)
{

        OBC_LOG("AOCS Algorithm task started..");
        rtems_status_code status;

        rtems_id RM_period;


        while(1)
        {
                AO_LVLH2ECI_initialize();
                AO_Rest2RestControl_initialize();


                rtems_task_wake_after(10);


                /*TM*/
                memcpy ( (void *) &StrRecv, (void *) STR_RT2BC_BUFF , sizeof(StrRecv)) ;
                memcpy ( (void *) &GpsRecv, (void *) GPS_RT2BC_BUFF , sizeof(GpsRecv)) ;
                memcpy ( (void *) &SSRecv, (void *) SS_RT2BC_BUFF , sizeof(SSRecv)) ;
                memcpy ( (void *) &SSyRecv, (void *) SS_RT2BC_BUFF2 , sizeof(SSyRecv)) ;
                memcpy ( (void *) &MtmRecv, (void *) MTM_RT2BC_BUFF , sizeof(MtmRecv)) ;
                memcpy ( (void *) &GyroWRecv, (void *) GYRO_RT2BC_BUFF , sizeof(GyroWRecv)) ;
                memcpy ( (void *) &RWTachoRecv, (void *) RW_RT2BC_BUFF , sizeof(RWTachoRecv)) ;
                memcpy ( (void *) &GSLVLHRecv, (void *) SRX_RT2BC_BUFF2 , sizeof(GSLVLHRecv)) ;
                memcpy ( (void *) &GSEciRecv, (void *) SRX_RT2BC_BUFF3 , sizeof(GSEciRecv)) ;
                memcpy ( (void *) &GSPolyIDRecv, (void *) SRX_RT2BC_BUFF4 ,
sizeof(GSPolyIDRecv)) ;
                memcpy ( (void *) &GSPolyDiffRecv, (void *) SRX_RT2BC_BUFF5 ,
sizeof(GSPolyDiffRecv)) ;
                memcpy ( (void *) &SatModeRecv, (void *) SRX_RT2BC_BUFF , sizeof(SatModeRecv))
;


                AO_Rest2RestControl_P.enableMomDump_Value=enableMomDump;
                AO_Rest2RestControl_P.r2rData_Value.Kd=Kd;
                AO_Rest2RestControl_P.r2rData_Value.Kp=Kp;
                AO_Rest2RestControl_P.r2rData_Value.KpSpeedDump=KpSpeedDump;
                AO_Rest2RestControl_P.r2rData_Value.KpSpeedMan=KpSpeedMan;
                memcpy(AO_Rest2RestControl_P.r2rData_Value.Mb2w,Mb2w,sizeof(Mb2w));
                memcpy(AO_Rest2RestControl_P.r2rData_Value.nomSpeed,nomSpeed,sizeof(nomSpeed));
                AO_Rest2RestControl_P.r2rData_Value.tMax=tMax;


        memcpy(AO_Rest2RestControl_P.q_Value,StrRecv.StrQuatOut,sizeof(StrRecv.StrQuatOut));
                memcpy(AO_Rest2RestControl_P.w_Value,GyroWRecv.gyroW,sizeof(GyroWRecv.gyroW));
                memcpy(AO_Rest2RestControl_P.speed_Value,RWTachoRecv.TachoSpeed,
```

```
sizeof(RWTachoRecv.TachoSpeed));
                /*Step*/
                memcpy(AO_LVLH2ECI_P.eulLVLH_Value,GSLVLHRecv.LvlhEuler,
sizeof(GSLVLHRecv.LvlhEuler));
                memcpy(AO_LVLH2ECI_P.wLVLH_Value,GSLVLHRecv.LvlhW,sizeof(GSLVLHRecv.LvlhW));
                memcpy(AO_LVLH2ECI_P.r_Value,GpsRecv.GpsR,sizeof(GpsRecv.GpsR));
                memcpy(AO_LVLH2ECI_P.v_Value,GpsRecv.GpsV,sizeof(GpsRecv.GpsV));
                AO_LVLH2ECI_P.Deg2Rad_Gain=0.017453292519943;
                /*Step transform LVLH2ECI*/
                AO_LVLH2ECI_step();
                memcpy(Qref,AO_LVLH2ECI_Y.qEci,sizeof(AO_LVLH2ECI_Y.qEci)); //Qref transform of
LVLH2ECI
                memcpy(Wref,AO_LVLH2ECI_Y.wBody,sizeof(AO_LVLH2ECI_Y.wBody)); //Wref transform
of LVLH2ECI
                memcpy(AO_Rest2RestControl_P.qRef_Value,Qref,sizeof(Qref));
                memcpy(AO_Rest2RestControl_P.wRef_Value,Wref,sizeof(Wref));
                AO_Rest2RestControl_step();

                memcpy(RWSend.RwTrqOut,AO_Rest2RestControl_Y.tCtrl,
sizeof(AO_Rest2RestControl_Y.tCtrl)) ; //send to OBC

                /*TC*/
                memcpy ( (void*)MTQ_BC2RT_BUFF,(void*) &MTQsend, sizeof(MTQsend)) ; // RT2 SA
29
                memcpy ( (void*) RW_BC2RT_BUFF,(void*) &RWSend, sizeof(RWSend)) ;  //RT 2 SA 30
                memcpy ( (void*) GYRO_BC2RT_BUFF,(void*) &GyroSend, sizeof(GyroSend)) ;  //RT
16 SA 30
                memcpy ( (void*) STX_BC2RT_BUFF,(void*) &SATModeSend, sizeof(SATModeSend)) ;
//RT 25 SA 30

                printf("AODCS Count is %d\n",AOCS_count);
                AOCS_count=AOCS_count+1;
                if( 0 == (AOCS_count % 1000) ) {
                        rtems_cpu_usage_report();
                        printf("AODCS------ Count is %d\n",AOCS_count);
                }
```

Sonuç;

```
------------------------------------------------OUTPUT-----------------------------------------
--

Initializing manager


--- GR-RASTA-IO[0] ---


 PCI BUS: 0, SLOT: 15, FUNCTION: 0


 PCI VENDOR: 0x1ac8, DEVICE: 0x0010


 PCI BAR[0]: 0xe1000000, BAR[1]: 0xd0000000, IRQ: 4


--- GR-RASTA-TMTC[0] ---


 PCI BUS: 0, SLOT: 19, FUNCTION: 0


 PCI VENDOR: 0x1ac8, DEVICE: 0x0011
```

```
 PCI BAR[0]: 0xe0000000, BAR[1]: 0xc0000000, IRQ: 4

AODCS Count is 0

AODCS Count is 1

AODCS Count is 2

AODCS Count is 3

AODCS Count is 4

AODCS Count is 5
.
.
..
.
.
.
.

AODCS Count is 995

AODCS Count is 996

AODCS Count is 997

AODCS Count is 998

AODCS Count is 999

------------------------------------------------------------------------------
                              CPU USAGE BY THREAD
------------+----------------------------------------+---------------+---------
 ID         | NAME                                   | SECONDS       | PERCENT
------------+----------------------------------------+---------------+---------
 0x09010001 | IDLE                                   |     92.654899 |  83.989
 0x0A010002 | 1553                                   |      0.001481 |   0.001
 0x0A010003 | AOCS                                   |     17.350450 |  15.727
------------+----------------------------------------+---------------+---------
 TIME SINCE LAST CPU USAGE RESET IN SECONDS:                       110.317556
------------------------------------------------------------------------------
AODCS------ Count is 1000
```

EK-2 Rate Monotonic

Kod

```
/*
 * AOCS_proc.c with RM
 *
 *  Created on: 09 Feb 2015
 *      Author:MEG
 */

#include <rtems.h>
/* configuration information */
#include "rtems_obc_demo.h"
#include "utils.h"
#include "obc_config.h"
#include "AOCS_Normal.h"

#include <stdio.h>
#include <stdlib.h>
//#include <grspw.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <gr1553bc.h
/*********************************TCA_RTW*********************************/
#include "./AOCS/AOCS_Const.h"
/*********************************REALTIME*********************************/
/********************************* Buffer for RealTime ***********************/
extern uint16_t SRX_RT2BC_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t SRX_RT2BC_BUFF2[32] __attribute__ ((aligned(8)));
extern uint16_t SRX_RT2BC_BUFF3[32] __attribute__ ((aligned(8)));
extern uint16_t SRX_RT2BC_BUFF4[32] __attribute__ ((aligned(8)));
extern uint16_t SRX_RT2BC_BUFF5[32] __attribute__ ((aligned(8)));
extern uint16_t SRX_BC2RT_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t PAB_RT2BC_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t PAB_BC2RT_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t RW_RT2BC_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t RW_BC2RT_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t MTQ_RT2BC_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t MTQ_BC2RT_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t PROP_RT2BC_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t PROP_BC2RT_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t MTM_RT2BC_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t MTM_BC2RT_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t SS_RT2BC_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t SS_RT2BC_BUFF2[32] __attribute__ ((aligned(8)));
extern uint16_t SS_BC2RT_BUFF[32] __attribute__ ((aligned(8)));

extern uint16_t PCDU_RT2BC_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t PCDU_BC2RT_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t THERM_RT2BC_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t THERM_BC2RT_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t GPS_RT2BC_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t GPS_BC2RT_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t GYRO_RT2BC_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t GYRO_BC2RT_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t STR_RT2BC_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t STR_BC2RT_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t STX_RT2BC_BUFF[32] __attribute__ ((aligned(8)));
extern uint16_t STX_BC2RT_BUFF[32] __attribute__ ((aligned(8)));
/*****************************************************************************************/

int AOCS_count=0;
double Qref[4]; //last Q reference of satellite
double Wref[3]; //last W reference of satellite
```

```
struct TcMtq MTQsend;
struct TcRW RWSend;
struct TcGyro  GyroSend;
struct TCSatMode SATModeSend;
struct TmSTR StrRecv;
struct TmGPS GpsRecv;
struct TmSS SSRecv;
struct TmSSonlyY SSyRecv;
struct TmMTM MtmRecv;
struct TmGyro GyroWRecv;
struct TmRW RWTachoRecv;
struct TmGSLVLH GSLVLHRecv;
struct TmGSEci GSEciRecv;
struct TmGSPolyID GSPolyIDRecv;
struct TmGSPolyDiff GSPolyDiffRecv;
struct TmGSMode SatModeRecv;

rtems_task AOCS_normal(rtems_task_argument arg)
{

        OBC_LOG("AOCS Algorithm task started..");
        rtems_status_code status;

        rtems_id RM_period;

        status = rtems_rate_monotonic_create( Task_name[T_AOCS_NORMAL], &RM_period );
        if( RTEMS_SUCCESSFUL != status ) {
                printf("RM failed with status: %d\n", status);
                exit(1);
          }

        while(1)
        {
                AO_LVLH2ECI_initialize();
                AO_Rest2RestControl_initialize();



                    if ( rtems_rate_monotonic_period( RM_period , 10 ) == RTEMS_TIMEOUT )
                      break;

                    /* Perform some periodic actions */

                /*TM*/
                memcpy ( (void *) &StrRecv, (void *) STR_RT2BC_BUFF , sizeof(StrRecv)) ;
                memcpy ( (void *) &GpsRecv, (void *) GPS_RT2BC_BUFF , sizeof(GpsRecv)) ;
                memcpy ( (void *) &SSRecv, (void *) SS_RT2BC_BUFF , sizeof(SSRecv)) ;
                memcpy ( (void *) &SSyRecv, (void *) SS_RT2BC_BUFF2 , sizeof(SSyRecv)) ;
                memcpy ( (void *) &MtmRecv, (void *) MTM_RT2BC_BUFF , sizeof(MtmRecv)) ;
                memcpy ( (void *) &GyroWRecv, (void *) GYRO_RT2BC_BUFF , sizeof(GyroWRecv)) ;
                memcpy ( (void *) &RWTachoRecv, (void *) RW_RT2BC_BUFF , sizeof(RWTachoRecv)) ;
                memcpy ( (void *) &GSLVLHRecv, (void *) SRX_RT2BC_BUFF2 , sizeof(GSLVLHRecv)) ;
                memcpy ( (void *) &GSEciRecv, (void *) SRX_RT2BC_BUFF3 , sizeof(GSEciRecv)) ;
                memcpy ( (void *) &GSPolyIDRecv, (void *) SRX_RT2BC_BUFF4 ,
sizeof(GSPolyIDRecv)) ;
                memcpy ( (void *) &GSPolyDiffRecv, (void *) SRX_RT2BC_BUFF5 ,
 sizeof(GSPolyDiffRecv)) ;
                memcpy ( (void *) &SatModeRecv, (void *) SRX_RT2BC_BUFF , sizeof(SatModeRecv))
;

                AO_Rest2RestControl_P.enableMomDump_Value=enableMomDump;
                AO_Rest2RestControl_P.r2rData_Value.Kd=Kd;
                AO_Rest2RestControl_P.r2rData_Value.Kp=Kp;
                AO_Rest2RestControl_P.r2rData_Value.KpSpeedDump=KpSpeedDump;
                AO_Rest2RestControl_P.r2rData_Value.KpSpeedMan=KpSpeedMan;
                memcpy(AO_Rest2RestControl_P.r2rData_Value.Mb2w,Mb2w,sizeof(Mb2w));
                memcpy(AO_Rest2RestControl_P.r2rData_Value.nomSpeed,nomSpeed,sizeof(nomSpeed));
                AO_Rest2RestControl_P.r2rData_Value.tMax=tMax;
```

```
        memcpy(AO_Rest2RestControl_P.q_Value,StrRecv.StrQuatOut,sizeof(StrRecv.StrQuatOut));

        //memcpy(AO_Rest2RestControl_P.wRef_Value,GSEciRecv.EciW,sizeof(GSEciRecv.EciW));
                memcpy(AO_Rest2RestControl_P.w_Value,GyroWRecv.gyroW,sizeof(GyroWRecv.gyroW));
                memcpy(AO_Rest2RestControl_P.speed_Value,RWTachoRecv.TachoSpeed,
sizeof(RWTachoRecv.TachoSpeed));
                /*Step*/

        memcpy(AO_LVLH2ECI_P.eulLVLH_Value,GSLVLHRecv.LvlhEuler,sizeof(GSLVLHRecv.LvlhEuler));
                memcpy(AO_LVLH2ECI_P.wLVLH_Value,GSLVLHRecv.LvlhW,sizeof(GSLVLHRecv.LvlhW));
                memcpy(AO_LVLH2ECI_P.r_Value,GpsRecv.GpsR,sizeof(GpsRecv.GpsR));
                memcpy(AO_LVLH2ECI_P.v_Value,GpsRecv.GpsV,sizeof(GpsRecv.GpsV));
                AO_LVLH2ECI_P.Deg2Rad_Gain=0.017453292519943;
                /*Step transform LVLH2ECI*/
                AO_LVLH2ECI_step();
                memcpy(Qref,AO_LVLH2ECI_Y.qEci,sizeof(AO_LVLH2ECI_Y.qEci)); //Qref transform of
LVLH2ECI
                memcpy(Wref,AO_LVLH2ECI_Y.wBody,sizeof(AO_LVLH2ECI_Y.wBody)); //Wref transform
of LVLH2ECI
                memcpy(AO_Rest2RestControl_P.qRef_Value,Qref,sizeof(Qref));
                memcpy(AO_Rest2RestControl_P.wRef_Value,Wref,sizeof(Wref));
                AO_Rest2RestControl_step();

                memcpy(RWSend.RwTrqOut,AO_Rest2RestControl_Y.tCtrl,
sizeof(AO_Rest2RestControl_Y.tCtrl)) ; //send to OBC header

                /*TC*/
                memcpy ( (void*)MTQ_BC2RT_BUFF,(void*) &MTQsend, sizeof(MTQsend)) ; // RT2 SA
29
                memcpy ( (void*) RW_BC2RT_BUFF,(void*) &RWSend, sizeof(RWSend)) ;  //RT 2 SA 30
                memcpy ( (void*) GYRO_BC2RT_BUFF,(void*) &GyroSend, sizeof(GyroSend)) ;  //RT
16 SA 30
                memcpy ( (void*) STX_BC2RT_BUFF,(void*) &SATModeSend, sizeof(SATModeSend)) ;
//RT 25 SA 30

                printf("AODCS Count is %d\n",AOCS_count);
                AOCS_count=AOCS_count+1;
                if( 0 == (AOCS_count % 1000) ) {
                        rtems_rate_monotonic_report_statistics();
                        rtems_cpu_usage_report();
                        printf("AODCS------ Count is %d\n",AOCS_count);
                }
        }
        status = rtems_rate_monotonic_delete( RM_period  );
          if ( status != RTEMS_SUCCESSFUL ) {
            printf( "rate_monotonic_delete failed with status of %d.\n", status );
            exit( 1 );
          }
          status = rtems_task_delete(RTEMS_SELF);
```

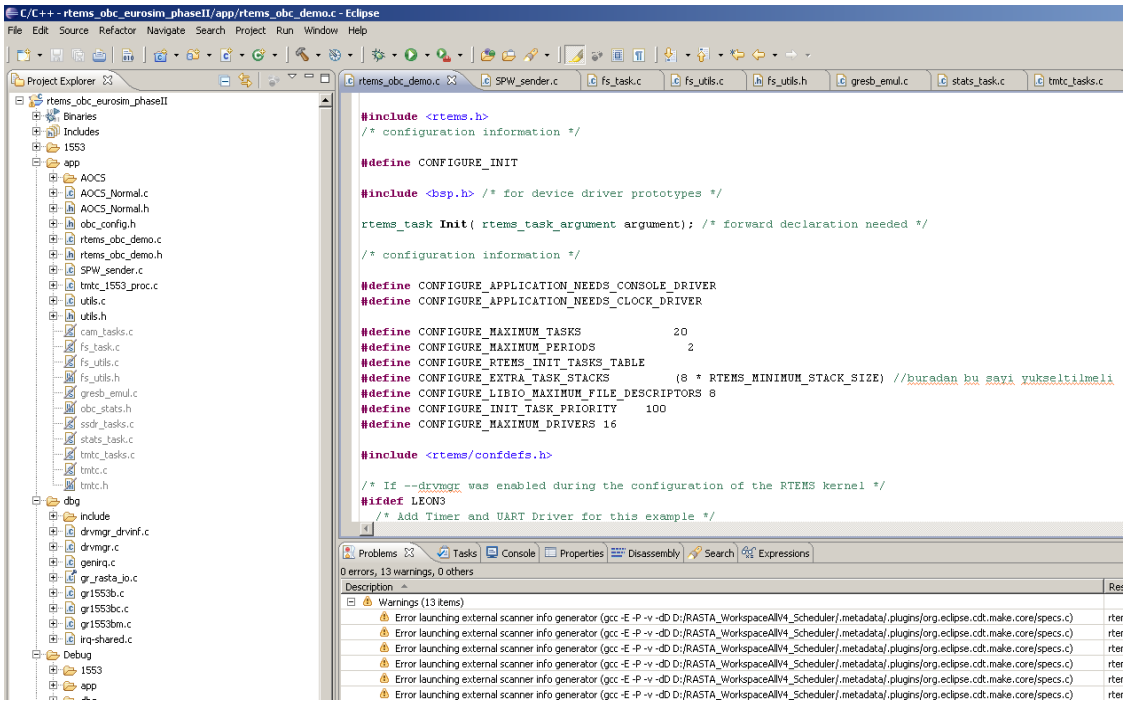
Sonuç;

```
-----------------------------------------------OUTPUT-------------------------------------

Initializing manager


--- GR-RASTA-IO[0] ---


 PCI BUS: 0, SLOT: 15, FUNCTION: 0


 PCI VENDOR: 0x1ac8, DEVICE: 0x0010
```

```
 PCI BAR[0]: 0xe1000000, BAR[1]: 0xd0000000, IRQ: 4

--- GR-RASTA-TMTC[0] ---

 PCI BUS: 0, SLOT: 19, FUNCTION: 0

 PCI VENDOR: 0x1ac8, DEVICE: 0x0011

 PCI BAR[0]: 0xe0000000, BAR[1]: 0xc0000000, IRQ: 4

AODCS Count is 0

AODCS Count is 1

AODCS Count is 2

AODCS Count is 3

.
.
.
.
.
.
AODCS Count is 996

AODCS Count is 997

AODCS Count is 998

AODCS Count is 999

------------------------------------------------------------------------------

                            CPU USAGE BY THREAD

------------+----------------------------------------+---------------+---------
 ID         | NAME                                   | SECONDS       | PERCENT
------------+----------------------------------------+---------------+---------
 0x09010001 | IDLE                                   |     82.599354 |  82.195
 0x0A010002 | 1553                                   |      0.001494 |   0.001
 0x0A010003 | AOCS                                   |     17.581507 |  17.495
------------+----------------------------------------+---------------+---------
 TIME SINCE LAST CPU USAGE RESET IN SECONDS:                      100.490895
------------------------------------------------------------------------------

AODCS------ Count is 1000
```

EK-3 Event Driven

IDE görüntüsü



Kod
Bulunduğu yer: tmtc_1553_proc.c
Major frame

```
/* Major Frame */
struct {
        int minor_cnt;
        struct gr1553bc_minor_cfg minor_cfgs[4];
} initial cfg =
{
        .minor_cnt = 4,

                        {
                                {.slot_cnt = 31, .timeslot = 97000}, //10000 */
                                {.slot_cnt = 1, .timeslot = 1000}, /* dummy delay */
                                {.slot_cnt = 1, .timeslot = 1000}, /* dummy delay */
                                {.slot_cnt = 1, .timeslot = 1000}, /* dummy delay */
                        },
};
```

Dört parça “minor frame”den ikinci olanı kesme için kullanılmıştır.
“{.slot_cnt = 1, .timeslot = 1000}, /* dummy delay */””

Bulunduğu yer: tmtc_1553_proc.c
Kesme slotu

```
        /* TC slot 9: interrupt */
        mid = GR1553BC_ID(0, 1, 0);
        gr1553bc_slot_alloc(list, &mid, 1000, NULL);
        gr1553bc_slot_irq_prepare(list,      mid,
                        tmtc 1553 tc read isr,
                        (void *)mid );
        gr1553bc_slot_irq_enable(list, mid);
```

MIL-STD 1553 veri yolu bu “slot” sayesinde kesmeleri bir olay ile gösterimini sağlarız. “tmtc_1553_tc_read_isr”

Bulunduğu yer: tmtc_1553_proc.c
(Interrupt service routine)

```
void tmtc_1553_tc_read_isr(union gr1553bc_bd  *bd, void *data)
{

        isr_count_tc_read_isr++;

        /* poke T 1553 PROCESSOR task */
        rtems_event_send( Task_id[T_1553_PROCESSOR], EVENT_1553_START );

}
```

“tmtc_1553_tc_read_isr” fonksiyonu içerisinde çağrılan. “**rtems_event_send**(Task_id[*T_1553_PROCESSOR*],EVENT_1553_START) ;” bir olay (event) oluşturur.

Bulunduğu yer: tmtc_1553_proc.c
“Olay ” alınması

```
rtems_task tmtc_1553_processor(
                rtems task argument arg
)
{
        rtems clock time value first,last,deltaTime;
        rtems_clock_get(RTEMS_CLOCK_GET_TIME_VALUE, &first);
        rtems event set rc event;

        OBC_LOG("TMTC 1553 processor task started..");

        if( tmtc_1553_init_bc() )
        {
                OBC ERR("Error initializing 1553 BC device");
                rtems_task_delete( RTEMS_SELF );
        }

        if( tmtc_1553_set_bc_transfers() )
        {
                OBC ERR("Error setting transfers");
                rtems_task_delete( RTEMS_SELF );
        }

        if( tmtc_1553_start() )
        {
                OBC ERR("Failed to start the BC");
                rtems_task_delete( RTEMS_SELF );
        }
        //char APP LOG[PATH MAX+1]={'A','P','P',' ','L','O','G',};
        while(1)
        {
                /* wait for isr to poke */
                rtems_event_receive( EVENT_1553_START,
                                RTEMS WAIT|RTEMS EVENT ALL, RTEMS NO TIMEOUT, &rc event);
//              rtems task wake after(10);
                rtems_clock_get(RTEMS_CLOCK_GET_TIME_VALUE, &first);

                //run_attitude_control_algorithm();
                rtems_clock_get(RTEMS_CLOCK_GET_TIME_VALUE, &last);

                deltaTime.microseconds=last.microseconds-first.microseconds;
                printf("Delta Time %d ms\n",deltaTime.microseconds);
                rtems_event_send( Task_id[T_AOCS_NORMAL], EVENT_AODCS_NORMAL );

        }
```

"rtems_event_receive(
EVENT_1553_START,RTEMS_WAIT|RTEMS_EVENT_ALL,
RTEMS_NO_TIMEOUT, &rc_event);"

Olay alma ile alınması sağlanır her 1553 veri yolundan kesme geldiğinde tetiklenir.
Kod parçasının sonunda

"**rtems_event_send**(Task_id[*T_AOCS_NORMAL*],
EVENT_AODCS_NORMAL);"

Başka bir olay oluşturulur bu olay YBKS görevini tetikler.

```
rtems_task AOCS_normal(rtems_task_argument arg)
{
        OBC LOG("AOCS Algorithm task started..");
        rtems event set rc event;
        while(1)
{
        rtems_event_receive( EVENT_AODCS_NORMAL,
                        RTEMS WAIT|RTEMS EVENT ALL, RTEMS NO TIMEOUT,&rc event);
        AO LVLH2ECI initialize();
        AO Rest2RestControl initialize();
                    /* Perform some periodic actions */
        memcpy ((void *)&StrRecv, (void *) STR_RT2BC_BUFF,sizeof(StrRecv)) ;
        memcpy ((void *)&GpsRecv, (void *) GPS_RT2BC_BUFF,sizeof(GpsRecv)) ;
       memcpy ((void *)&SSRecv, (void *) SS_RT2BC_BUFF,sizeof(SSRecv)) ;
        memcpy ((void *)&SSyRecv, (void *) SS_RT2BC_BUFF2,sizeof(SSyRecv)) ;
        memcpy ((void *)&MtmRecv, (void *) MTM_RT2BC_BUFF,sizeof(MtmRecv)) ;
        memcpy ((void *)&GyroWRecv, (void *) GYRO_RT2BC_BUFF,sizeof(GyroWRecv)) ;
        memcpy ((void *)&RWTachoRecv, (void *) RW_RT2BC_BUFF,sizeof(RWTachoRecv)) ;
        memcpy ((void *)&GSLVLHRecv, (void *) SRX_RT2BC_BUFF2,sizeof(GSLVLHRecv)) ;
        memcpy ((void *)&GSEciRecv, (void *) SRX_RT2BC_BUFF3,sizeof(GSEciRecv)) ;
        memcpy ((void *)&GSPolyIDRecv, (void *)SRX_RT2BC_BUFF4,sizeof(GSPolyIDRecv));
        memcpy ((void *)&GSPolyDiffRecv,(void*)SRX_RT2BC_BUFF5,sizeof(GSPolyDiffRecv));
        memcpy ((void *)&SatModeRecv, (void *) SRX_RT2BC_BUFF,sizeof(SatModeRecv));
               AO_Rest2RestControl_P.enableMomDump_Value=enableMomDump;
               AO Rest2RestControl P.r2rData Value.Kd=Kd;
               AO Rest2RestControl P.r2rData Value.Kp=Kp;
               AO Rest2RestControl P.r2rData Value.KpSpeedDump=KpSpeedDump;
               AO_Rest2RestControl_P.r2rData_Value.KpSpeedMan=KpSpeedMan;
        memcpy(AO_Rest2RestControl_P.r2rData_Value.Mb2w,Mb2w,sizeof(Mb2w));
        memcpy(AO_Rest2RestControl_P.r2rData_Value.nomSpeed,
nomSpeed,sizeof(nomSpeed));
        AO Rest2RestControl P.r2rData Value.tMax=tMax;
        memcpy(AO_Rest2RestControl_P.q_Value,StrRecv.StrQuatOut,
sizeof(StrRecv.StrQuatOut));
        memcpy(AO_Rest2RestControl_P.w_Value,GyroWRecv.gyroW,
sizeof(GyroWRecv.gyroW));
        memcpy(AO_Rest2RestControl_P.speed_Value,RWTachoRecv.TachoSpeed,
sizeof(RWTachoRecv.TachoSpeed));
               /*Step*/
        memcpy(AO_LVLH2ECI_P.eulLVLH_Value,GSLVLHRecv.LvlhEuler,
sizeof(GSLVLHRecv.LvlhEuler));
        memcpy(AO_LVLH2ECI_P.wLVLH_Value,GSLVLHRecv.LvlhW,sizeof(GSLVLHRecv.LvlhW));
        memcpy(AO_LVLH2ECI_P.r_Value,GpsRecv.GpsR,sizeof(GpsRecv.GpsR));
        memcpy(AO LVLH2ECI P.v Value,GpsRecv.GpsV,sizeof(GpsRecv.GpsV));
        AO_LVLH2ECI_P.Deg2Rad_Gain=0.017453292519943;
        /*Step transform LVLH2ECI*/
        AO LVLH2ECI step();
        memcpy(Qref,AO_LVLH2ECI_Y.qEci,sizeof(AO_LVLH2ECI_Y.qEci));
        memcpy(Wref,AO_LVLH2ECI_Y.wBody,sizeof(AO_LVLH2ECI_Y.wBody))
        memcpy(AORest2RestControl P.qRefValue,Qref,sizeof(Qref));
        memcpy(AO_Rest2RestControl_P.wRef_Value,Wref,sizeof(Wref));
        AO Rest2RestControl step();
        printf("AODCS Count is %d\n",AOCS_count);
        AOCS count=AOCS count+1;
        if( 0 == (AOCS_count % 1000) ) {
               rtems cpu usage report();
               printf("AODCS------ Count is %d\n",AOCS_count);
}
```

YBKS görevinin periyodik çağrılması

"`rtems_event_receive( EVENT_AODCS_NORMAL,RTEMS_WAIT|RTEMS_EVENT_ALL, RTEMS_NO_TIMEOUT, &rc_event);`"

"`EVENT_AODCS_NORMAL`" olayı ile sağlanır.

Çıktılar:
Bulunduğu yer: Konsol

```
Initializing manager
--- GR-RASTA-IO[0] ---

 PCI BUS: 0, SLOT: 15, FUNCTION: 0

 PCI VENDOR: 0x1ac8, DEVICE: 0x0010

 PCI BAR[0]: 0xe1000000, BAR[1]: 0xd0000000, IRQ: 4

--- GR-RASTA-TMTC[0] ---

 PCI BUS: 0, SLOT: 19, FUNCTION: 0

 PCI VENDOR: 0x1ac8, DEVICE: 0x0011

 PCI BAR[0]: 0xe0000000, BAR[1]: 0xc0000000, IRQ: 4

AODCS Count is 0


AODCS Count is 1

.
.


AODCS Count is 998


AODCS Count is 999


--------------------------------------------------------------------------------

                              CPU USAGE BY THREAD

------------+---------------------------------------+---------------+---------

 ID         | NAME                                  | SECONDS       | PERCENT

------------+---------------------------------------+---------------+---------

 0x09010001 | IDLE                                  |     82.776076 |  83.342

 0x0A010002 | 1553                                  |      0.107517 |   0.108

 0x0A010003 | AOCS                                  |     16.124382 |  16.234

------------+---------------------------------------+---------------+---------

 TIME SINCE LAST CPU USAGE RESET IN SECONDS:                        99.320855

--------------------------------------------------------------------------------

AODCS------ Count is 1000
```