Uydu Bilgisayarı için Yeni Bir Önyükleyici Yaklaşımının Tasarımı ve Uygulanması

Ezgi Kutlu Murat

YÜKSEK LİSANS

Elektrik-Elektronik Mühendisliği Anabilim Dalı

Ocak 2024

Design and Implementation of a New Bootloader Approach for a Satellite On-Board Computer

Ezgi Kutlu Murat

MASTER OF SCIENCE THESIS

Department of Electrical and Electronics Engineering

January 2024

Uydu Bilgisayarı için Yeni Bir Önyükleyici Yaklaşımının Tasarımı ve Uygulanması

Ezgi Kutlu Murat

Eskişehir Osmangazi Üniversitesi

Fen Bilimleri Enstitüsü

Lisansüstü Eğitim ve Öğretim Yönetmeliği Uyarınca

Elektrik-Elektronik Mühendisliği Anabilim Dalı

Telekomünikasyon-Sinyal İşleme Bilim Dalında

YÜKSEK LİSANS TEZİ

olarak hazırlanmıştır.

Danışman: Prof. Dr. Hakan Çevikalp

Ocak 2024

ONAY

Elektrik-Elektronik Mühendisliği Anabilim Dalı Telekomünikasyon ve Sinyal İşleme Bilim Dalı Yüksek Lisans **Ezgi Kutlu Murat**'ın YÜKSEK LİSANS tezi olarak hazırladığı "Uydu Bilgisayarı için Yeni Bir Önyükleyici Yaklaşımının Tasarımı ve Uygulanması" başlıklı bu çalışma, jürimizce lisansüstü yönetmeliğin ilgili maddeleri uyarınca değerlendirilerek oybirliği ile kabul edilmiştir.

Danışman : Prof. Dr. Hakan Çevikalp

İkinci Danışman :

Yüksek Lisans /Doktora Tez Savunma Jürisi:

Üye: Prof. Dr. Hakan Çevikalp

Üye: Doç. Dr. Cihan Topal

Üye: Doç. Dr. Erol Seke

Üye:

Üye:

Fen Bilimleri Enstitüsü Yönetim Kurulu'nun ……..…….. tarih ve ……..…… sayılı kararıyla onaylanmıştır.

Prof. Dr. ....

Enstitü Müdürü

ETİK BEYAN

Eskişehir Osmangazi Üniversitesi Fen Bilimleri Enstitüsü tez yazım kılavuzuna göre, Prof. Dr. Hakan Çevikalp danışmanlığında hazırlamış olduğum "Uydu Bilgisayarı için Yeni Bir Önyükleyici Yaklaşımının Tasarımı ve Uygulanması" başlıklı Yüksek Lisans tezimin özgün bir çalışma olduğunu; tez çalışmamın tüm aşamalarında bilimsel etik ilke ve kurallarına uygun davrandığımı; tezimde verdiğim bilgileri, verileri akademik ve bilimsel ilke ve kurallara uygun olarak elde ettiğimi; tez çalışmamda yararlandığım eserlerin tümüne atıf yaptığımı ve kaynak gösterdiğimi ve bilgi, belge ve sonuçları bilimsel etik ilke ve kurallara göre sunduğumu beyan ederim. 15/01/2024

Ezgi Kutlu Murat

İmza

ÖZET

Uydu bilgisayarlarında temel olarak kullanılan iki yazılım, uçuş yazılımı ve önyükleyici yazılımdır. Uçuş yazılımı, bir uydunun ömrü boyunca çeşitli görevleri yerine getirmesini sağlayan ve kritik işlevleri kontrol eden temel yazılım olarak öne çıkar. Öte yandan, önyükleyici yazılım, bilgisayarın her başlatıldığında devreye girip kontrolü uçuş yazılımına devredecek şekilde tasarlanmıştır.

Bu çalışma, geleneksel önyükleme yöntemlerinden farklı olarak, uydu uçuş sistemleri için yeni bir önyükleyici yazılım yaklaşımını detaylı bir şekilde ele almaktadır. Bu yaklaşım, mevcut standartlardan ayrılarak, uydu sistemlerinin gereksinimlerini daha etkin bir şekilde karşılamak için tasarlanmış yazılım stratejilerini içermektedir.

Önyükleyici yazılım, uydu bilgisayarında ilk olarak devreye giren ve uçuş yazılımını başlatmadan önce potansiyel hataları tespit etme ve düzeltme sorumluluğunu üstlenen kritik bir yazılımdır. Aynı zamanda, uzay ortamındaki çeşitli zorlayıcı etkenleri göz önünde bulundurarak güvenilirlik ve hata toleransı yüksek bir önyükleyici yazılım tasarımı amaçlanmaktadır. Özellikle, uydunun yörüngede bulunduğu süreçte ortaya çıkabilecek yazılım hatalarını düzeltmek amacıyla önyükleyici yazılım aracılığıyla uçuş yazılımının güncellenmesi özelliği, bu çalışmanın odak noktalarından biridir. Bu yaklaşım, mevcut yazılım güncelleme yöntemlerine alternatif bir çözüm sunmayı hedeflemektedir.

Bu çalışma, uydu bilgisayarları için yeni bir önyükleyici yazılım yaklaşımının adım adım geliştirilme sürecini açıklamaktadır. Bu detaylı süreç, uydu sistemlerinin daha güvenilir, verimli ve etkili bir şekilde çalışmasını sağlamaktadır. Ayrıca, bu araştırmanın metodolojisi ve bulguları, benzer projelerde rehberlik sağlayabilir. Bu kapsamlı yaklaşımın, uydu sistemleri alanında ilerlemeye ve akademik literatüre önemli bir katkı sağlama potansiyeli olduğu değerlendirilmektedir.

Anahtar Kelimeler: Önyükleyici Yazılım, Uydu Bilgisayarı, Uçuş Yazılımı, Yazılım Güncelleme, Uzay Ortamı, Hata Tespiti ve Düzeltme, Uydu Sistemleri, Yazılım Geliştirme Süreci

SUMMARY

Two primary software components utilized in satellite computers are the flight software and the bootloader. The flight software stands out as the core software responsible for executing various tasks throughout a satellite's lifespan and overseeing critical functionalities. Conversely, the bootloader is designed to engage every time the computer starts up, facilitating the transition of control to the flight software.

This study intricately explores a novel approach to bootloader systems in satellite flight systems, distinct from conventional booting methods. This approach is designed to diverge from current standards and encompasses software strategies tailored to effectively meet the requirements of satellite systems.

The bootloader, the initial program activated within the satellite's computer, assumes a critical role in identifying and rectifying potential errors before initiating the flight software. Additionally, the goal is to craft a bootloader design with high reliability and error tolerance, considering various demanding factors in the space environment. Notably, a focal point of this study is the capability of updating the flight software through the bootloader, specifically to rectify software errors that may arise during the satellite's orbital phase. This approach aims to present an alternative solution to current software update methods.

This study describes the step-by-step development process of a new bootloader software approach for satellite computers. This detailed process ensures the more reliable, efficient, and effective operation of satellite systems. Furthermore, the methodology and findings of this research can provide guidance for similar projects. The comprehensive nature of this approach is deemed to have significant potential in advancing satellite systems and making a noteworthy contribution to academic literature.

Keywords: Bootloader, Satellite On-board Computer, Flight Software, Software Update, Space Environment, Error Detection and Correction, Satellite Systems, Software Development Process

TEŞEKKÜR

Bu çalışmanın yürütülmesi sırasında desteğini esirgemeyen danışmanım Prof. Dr. Hakan Çevikalp'e, engin bilgi ve tecrübelerinden yararlandığım ve iş hayatıma paha biçilmez katkıları için Akın Yılmaz'a, yazılım yaması konusundaki yönlendirmeleri ve yardımları için Camal Gençtürk'e, çalışmalarım sırasında beni motive ettikleri ve bana hep destek oldukları için iş arkadaşlarıma ve özellikle bana sürekli yardımcı olan değerli arkadaşım Renda Yiğit'e, tüm hayatım boyunca benden maddi ve manevi desteklerini esirgemeyen her zaman yanımda olan sevgili aileme ve çalışma süresince tüm zorlukları benimle göğüsleyen ve hayatımın her evresinde bana destek olan değerli eşim Yunus Murat'a sonsuz teşekkürlerimi sunarım.

Ezgi Kutlu Murat

İÇİNDEKİLER

Sayfa

İÇİNDEKİLER

Sayfa

ŞEKİLLER DİZİNİ

Şekil **Sayfa**

ÇİZELGELER DİZİNİ

Çizelge **Sayfa**

SİMGELER VE KISALTMALAR DİZİNİ

Kısaltmalar	Açıklama
ASIC	Application Specific Integrated Circuit (Uygulamaya Özel Tümleşik Devre)
CPU	Central Processing Unit (Merkezi İşlem Birimi)
CRC	Cyclic Redundancy Check (Döngüsel Artıklık Kontrolü)
ECSS	European Cooperation for Space Standardization (Uzay Standardizasyonu için Avrupa İş Birliği)
EDAC	Error Detection and Correction (Hata Tespiti ve Düzeltme)
EEPROM	Electronically Erasable Programmable Read-Only Memory (Elektronik Olarak Silinebilir Programlanabilir Salt Okunur Bellek)
ELF	Executable and Linkable Format (Yürütülebilir ve Bağlanabilir Biçim)
ESA	European Space (Agency Avrupa Uzay Ajansı)
FDIR	Fault Detection, Isolation and Recovery (Arıza Tespiti, İzolasyon ve Kurtarma)
FPGA	Field Programmable Gate Array (Alanda Programlanabilir Kapı Dizileri)
FPU	Floating Point Unit (Kayan Nokta Birimi)
FT	Fault-tolerant (Hata Toleransı)
GCC	GNU Compiler Collection (GNU Derleyici Koleksiyonu)
GNU	GNU's Not Unix (GNU Unix değildir)
LEO	Low Earth Orbit (Alçak Dünya Yörüngesi)

SİMGELER VE KISALTMALAR DİZİNİ

Kısaltmalar	Açıklama
NASA	National Aeronautics and Space Administration (Ulusal Havacılık ve Uzay Dairesi)
ÖY	Önyükleyici Yazılımı
ÖYF-1	Önyükleyici Yazılımı Faz-1
ÖYF-2	Önyükleyici Yazılımı Faz-2
PUS	Packet Utilisation Standard (Paket Kullanım Standardı)
RAM	Random Access Memory (Rastgele Erişim Belleği)
ROM	Read Only Memory (Salt Okunur Bellek)
SAR	Synthetşc Aperture Radar (Sentetik Açıklı Radar)
SAVOIR	Space Avionics Open Interface Architecture (Uzay Aviyonik Açık Arayüz Mimarisi)
SCOC	Space Computer on a Chip (Çip Üzerindeki Uzay Bilgisayarı)
SDRAM	Synchronous Dynamic Random-Access Memory (Eşzamanlı Dinamik Rastgele Erişimli Bellek)
SEL	Single Event Latch-Up (Tekil Olay Kalıcı Hata)
SEU	Single Event Upset (Tekil Olay Geçici Hata)
SPARC	Scalable Processor Architecture (Ölçeklenebilir İşlemci Mimarisi)
TMR	Triple Modular Redundancy (Üçlü Modüler Yedeklilik)
UART	Universal Asynchronous Receiver Transmitter (Evrensel Asenkron Alıcı / Verici)
UY	Uçuş Yazılımı

1. GİRİŞ VE AMAÇ

Önyükleyici yazılım, bilişim sistemlerinde önemli bir konuma sahip olan ve bir sistemin başlatılmasında temel bir rol oynayan ilk yazılım türüdür. Bu yazılım, bilgisayar sistemlerinde işletim sistemini yüklemek ve çalıştırmak için gereken temel adımları gerçekleştirir. Mikrodenetleyicilerde ise programlanabilir cihazların donanıma yüklenmesinde kritik bir fonksiyonu bulunmaktadır. Önyükleyici yazılım, belirli bir hafıza bölgesine yerleştirilir ve donanım özelliklerine uygun olarak USB, UART, CAN, SPI, Wi-Fi, Ethernet gibi veri alışverişi yapabilen çevre birimlerine erişebilir. Ayrıca, hafıza okuma/yazma işlemleri gerçekleştirebilir. Bu yetenekler sayesinde, çeşitli iletişim çevresel birimlerinden ana yazılımın dış kaynaklardan aktarılarak yüklenmesi ve işletilmesi mümkün olmaktadır.

Uydu sistemlerinde, uçuş yazılımı ve önyükleyici yazılım temel yazılım türleri olarak kabul edilmekte ve bir sistemin kritik işlevlerini ve başlatılma sürecini kontrol etmektedir. Uydu uçuş yazılımı, uydunun yaşam döngüsü boyunca önceden tanımlanmış görevleri yerine getirmesini sağlayan ve uydu ile ilgili hayati fonksiyonları kontrol eden yazılımdır (Kutlu ve Gençtürk, 2021). Uydu önyükleyici yazılım ise, bilgisayara güç verildiğinde veya bilgisayar yeniden başlatıldığında, uçuş yazılımına geçiş yapılıncaya kadar çalışan temel bir yazılımdır. Bu aşamada donanımın başlatılması, uçuş yazılımının yüklenmesi öncesi çeşitli kontrol ve işlemleri gerçekleştirir. Bu nedenle, önyükleyici yazılımın güvenilirliği ve işlevselliği, uzaydaki bir görevin başarılı bir şekilde gerçekleşmesi için hayati önem taşımaktadır.

Bu çalışma, geleneksel önyükleme yöntemlerinden farklı olarak, uydu uçuş sistemleri için yeni bir önyükleyici yazılım yaklaşımını detaylı bir şekilde ele almaktadır. Bu yaklaşım, mevcut standartlardan ayrılarak, uydu sistemlerinin ihtiyaçlarını daha etkin bir şekilde karşılamak üzere tasarlanmış yazılım stratejilerini içermektedir. Önyükleyici yazılım, uçuş yazılımını başlatmadan önce potansiyel hataları tespit etme ve düzeltme sorumluluğunu üstlenen kritik bir yazılım olarak öne çıkmaktadır. Aynı zamanda, uzay ortamındaki çeşitli zorlayıcı etkenleri göz önünde bulundurarak güvenilirlik ve hata toleransı yüksek bir önyükleyici yazılım tasarımı amaçlanmaktadır. Özellikle, uydunun yörüngede

bulunduğu süreçte ortaya çıkabilecek yazılım hatalarını düzeltmek amacıyla önyükleyici yazılım aracılığıyla uçuş yazılımının güncellenmesi özelliği, bu çalışmanın odak noktalarından biridir. Bu yaklaşım, mevcut yazılım güncelleme yöntemlerine alternatif bir çözüm sunmayı hedeflemektedir.

Bu çalışma ayrıca, uzay ortamında çalışan önyükleyici yazılımın tasarımı ve uygulanmasıyla ilgili literatür incelemelerinin yapılmasını içermekte ve bu incelemelerin sonuçlarına dayanarak geliştirilmekte olan uydu bilgisayarı önyükleyici yazılımın tasarım çözümlerinin değerlendirilmesini sağlamaktadır.

Bu araştırma, uydu bilgisayarları için yeni bir önyükleyici yazılım yaklaşımının geliştirilme sürecini, gereksinim analizinden başlayarak tasarım, uygulama, doğrulama ve geçerleme aşamalarını ayrıntılı bir şekilde ele almaktadır. Bu detaylı yaklaşımın sağladığı temel fayda, uydu sistemlerinin karmaşık gereksinimlerini daha kapsamlı bir şekilde anlama ve ele alma imkanıdır. Gereksinim analizi, uydu sistemlerinin işlevsel ve performans gereksinimlerinin yanı sıra güvenilirlik, güvenlik ve sistem bütünlüğü gibi kritik faktörlerini belirleyerek, tasarım ve uygulama aşamalarında doğru bir yönlendirme sağlar. Tasarım aşamasında, belirlenen gereksinimler doğrultusunda uydu yazılımının mimarisi oluşturulur ve uydu sistemi için uygun olan algoritmalar ve veri yapıları seçilir. Uygulama aşaması, tasarlanan yazılımın gerçek ortama entegrasyonunu içerir ve yazılımın uydu donanımı üzerinde başarılı bir şekilde çalışması sağlanır. Doğrulama ve geçerleme aşamaları ise yazılımın gereksinimlere uygunluğunun ve performansının test edilmesini içerir; bu da yazılımın güvenilirliğini ve etkinliğini sağlamak için önemlidir. Bu süreç, uydu sistemlerinin daha güvenilir, verimli ve etkili bir şekilde çalışmasını sağlamaktadır. Bu kapsamlı yaklaşım, literatüre katkı sağlama ve uydu sistemleri alanında yeni bir bakış açısı sunma amacını taşımaktadır.

2. LİTERATÜR ARAŞTIRMASI

Günümüzde uzay ortamında faaliyet gösteren sistemler, yüksek radyasyon seviyeleri ve diğer çevresel zorluklarla karşı karşıya kalmaktadır. Bu sistemlerin, bir hatayla karşılaşmaları durumunda uzaktan müdahale edilemeyecekleri ve doğrudan çözülemeyecekleri bir ortamda çalışması beklenmektedir. Bu durum, özellikle uzay misyonları için kritik olan sistemlerin, güvenilirlik ve hata toleransı sağlayan mekanizmalarla donatılmasını gerektirmektedir. Uzay görevlerinde kullanılan önyükleyici yazılımların literatürdeki incelenmesi, bu alanda mevcut kaynakların sınırlı olduğunu göstermektedir. Bu kaynaklar, önyükleyici yazılımın tasarımı ve uygulanmasında güvenilirlik kavramının vurgulanması bakımından önemlidir.

Öncelikle, gömülü sistemler için güvenilir önyükleme yöntemlerine ilişkin bir inceleme yapılmıştır. Bu inceleme, LEON3 açık kaynak yazılım çekirdekli (SPARC V8 komut seti) iki işlemcili bir tasarımın güvenilir önyükleme özelliklerini ele almıştır. Bu tasarımda, bir işlemci uygulama işlevini yerine getirirken, diğer işlemci güvenlik odaklı bir yardımcı işlevini üstlenmektedir. Uygulama işlemcisi, ölçüm amacıyla statik bir güven kökü olarak bir önyükleme ROM'u ile birleştirilirken, güvenli yardımcı işlemci güvenilir platform modülü ürün yazılımını yürüterek, uygulama işlemcisine ait yazılım kimliklerini ve özel anahtarları mühürlerken farklı yazılımları önyükleyip çalıştırmasını sağlar. Geleneksel gömülü sistemler "güvenli önyükleme" için önyükleme ROM'undaki bir imza doğrulaması kullanırken, bu mimari "güvenilir önyükleme" ile sahte imzalara ya da sertifikalara karşı ek bir güvenlik katmanı sunar (Khalid vd., 2013).

Bu sunulan uygulamanın temel amacı, yazılım saldırılarına karşı koruma sağlamaktır. Ancak, bu mimari, %113'lük bir ek donanım maliyeti ve %25'lik bir ek önyükleme gecikmesi ile birlikte "güvenilir önyükleme" işlevselliğini yerine getirir. Khalid vd.ne (2013) göre gelecekteki çalışmalarda, performans iyileştirmeleri için kod boyutunun önemli ölçüde azaltılması mümkün olabilir. Önyükleme süresi, doğrudan bellek erişimi denetleyicileri kullanılarak geliştirilebilir. Ayrıca, karma hesaplamalar için daha küçük bir işlemci ve donanım hızlandırıcı kullanılarak daha iyi bir önyükleme süresi elde edilebilir.

İkinci olarak, anti-radyasyon özellikli TSC695F işlemcisine dayalı bir havacılık ve uzay yükü denetleyicisinin önyükleyici yazılım tasarımına ilişkin bir inceleme yapılmıştır. Bu işlemci genellikle uzay ortamında yüksek güvenilirlik gerektiren faydalı yük kontrolcüsü olarak kullanılmaktadır ve SPARC V7 mimarisine sahiptir. Aynı zamanda, parite kontrolü, hata tespiti ve düzeltme mekanizmaları gibi unsurlara sahiptir, bu özellikler sayesinde yüksek güvenilirliğe sahip önyükleyici yazılım tasarımı mümkün kılınmıştır. Önyükleyici yazılım, işletim sistemi ve uygulama kodlarının doğru bir şekilde yüklenmesini, kararlı ve verimli bir şekilde çalışmasını sağlamak amacıyla geliştirilmiştir (Xu ve Piao, 2010).

Makalede, TSC695F işlemcisi için önyükleyici yazılımının faydalı yük kontrolcüsü kapsamında yapılan yazmaç ve yığın bellek ilklendirmeleri, sistem donanımı ilklendirmeleri ve yazılımsal/donanımsal kesme kurma yöntemi adımları detaylı bir şekilde sunulmuştur. Assembly seviyesinde gerçekleştirilen işlemlerden sonra C diline geçilmiş ve burada bekleme, yükleme ve enjeksiyon modlarıyla bu modlar arasında geçişler gibi daha karmaşık işlemler ele alınmıştır. Ayrıca, çalışmada kullanılan hedef kodlar ve derleme adımları gibi detaylar da yer almaktadır. Bekleme modu, başlangıçta varsayılan olarak gelir ve eğer çevresel birimlerden haberleşme hattı üzerinden 10 saniye boyunca bir talimat alınmazsa yükleme moduna geçilir. Yükleme modunda, uygulama yazılımı kalıcı hafızadan geçici hafızaya yüklenir ve işletilir, böylece önyükleme aşaması tamamlanır. Enjeksiyon modunda ise çevresel birimlerden alınan veriler kalıcı hafızaya kaydedilir ve ardından bekleme moduna geri dönülür, böylece kalıcı hafızadaki değiştirilmiş uygulama yazılımı yükleme modunda çalıştırılmış olur (Xu ve Piao, 2010).

TSC695F için geliştirilen önyükleyici yazılımın analizini ve tasarımını içeren bu makale, SPARC mimarisi tabanlı diğer sistemler için de genişletilebilir niteliktedir. Bu sayede, uzay alanında sıklıkla tercih edilen anti-radyasyon özellikli bu sistemler için mühendislik çözümleri referans alınarak uygulanabilir. Az sayıda kaynak bulunması sebebiyle, bir önyükleyici yazılımın tasarımı ve uygulanması ile ilgili bu çalışma önemli bir katkı sunmaktadır.

Üçüncü olarak, uzay görevlerinde kritik bir rol üstlenen uçuş bilgisayarlarının başlatma sekansından sorumlu olan uydu önyükleyici yazılımının güvenilirlik odaklı tasarımına ilişkin inceleme yapılmıştır. Uzay ortamında faaliyet gösteren işlemci modülleri,

radyasyona karşı yüksek tolerans gerektiren yapıya sahip olmalıdır. Uzay sistemleri, görev gereksinimlerini karşılamak için farklı mekanizmalar ve standartlar gerektirmektedir. Avrupa Uzay Ajansı (ESA) tarafından benimsenen Avrupa Uzay Standartları İçin Avrupa İşbirliği (ECSS), uzay görevlerinde hata algılama, izolasyon ve iyileştirme (FDIR) sistemleri olarak adlandırılan mekanizmaları kullanarak hataya dayanıklılık sağlamaktadır. Sorunlar ortaya çıktığında, doğrudan operatör müdahalesi veya hasarlı ekipmanın değiştirilmesi gibi çözümler mümkün olmayabilir. Bu nedenle, sistem yazılımının donanım hatalarını tespit edebilme yeteneği, yazılım tabanlı hata toleransı tekniklerinin uygulanabilmesi için önemlidir. Uydu önyükleyici yazılımının tasarımında sistem güvenirliğini sağlamak için hata algılama ve düzeltme yetenekleri entegre edilmelidir (Polo vd., 2021).

Makalede, Solar Orbiter görevinde kullanılan Energetic Particle Detector cihazının kontrol ünitesi için önyükleyici yazılım geliştirme çalışması incelenmektedir. Bu görevde, yüksek enerjili parçacıkların özellikle SDRAM ve EEPROM gibi diğer bileşenlere göre daha düşük radyasyon toleransına sahip olan elektronik cihazlarda tekil olay etkilerine neden olabileceği belirtilmiştir. Öncelikle, önyükleyici yazılımın davranışı üzerindeki tekil olay etkilerinin sıklığı üzerine bir çalışma yürütülmüştür. Bu analiz için bir hata enjeksiyon ortamı kullanılmıştır. Sonuçlar, önyükleyici yazılımın bu tür kalıcı etkilere karşı savunmasız olduğunu göstermektedir. Önyükleyici yazılımın temel amacı, uygulama yazılımını yüklemek ve kontrolünü devretmektir. Ancak, önyükleyici yazılımın bellekteki kalıcı hatalar nedeniyle kendisi çalışamaz durumdaysa, bu işlevsellik asla yerine getirilemez. Hatalar sistem belleğinde herhangi bir zamanda ve herhangi bir yerde meydana gelebilir, bu nedenle önyükleyici yazılımın güvenilirlik odaklı bir tasarımı geliştirilmiştir. Bu tasarımda, FDIR spesifikasyonları doğrultusunda bir dizi tasarım kararı alınmıştır (Polo vd., 2021).

- Birinci set, SDRAM'deki Single Event Latch-Ups (SEL) hatalarına tolerans sağlamayı hedefler. Önyükleyici yazılımın gerektirdiği SDRAM alanının dinamik tahsisine odaklanır ve bu alanlarda hafıza testleri gerçekleştirilir.

- İkinci set, EEPROM'daki Single Event Upsets (SEU) hatalarına ve SEL hatalarına tolerans sağlamayı amaçlar. Kritik değişkenlerde Triple Modular Redundancy (TMR) kullanımına odaklanır, uygulama yazılımı imajlarının segment yapılarına ve çeşitli kontroller sonucunda imajın yüklenmesine ve çalıştırılmasına dikkat eder. İmaj

segmentlerinin yapısal kontrolleri ve bütünlük Cyclic Redundancy Check (CRC) kontrolleri gerçekleştirilir.

- Ayrıca, önyükleyici yazılımın tasarımı, SDRAM'de SEU oluşumu için hata toleransı sağlamayı amaçlar. SEU'lar, SEL'lerden daha fazla görülme eğilimindedir. Tekil hataları düzeltmek için bellek temizleme (Scrubbing) ve Error Detection and Correction (EDAC) çift bit hata algılama gibi standart yazılım mekanizmaları kullanılır.

Yazılım tasarım kararları hayata geçirildikten sonra kalifikasyon faaliyetleri gerçekleştirilmiş ve Şubat 2020'de devreye alınmıştır. Benzer hatalarla karşılaşan uzay sistemlerinde, bu çalışmadaki tasarım kararları doğrudan referans alınabilir. Gerçek bir uygulama üzerinde çalışması nedeniyle, bu çalışma büyük önem taşımaktadır.

Son olarak, Solar Orbiter'in Energetic Particle Detector cihazının kontrol ünitesi için önyükleyici yazılımın çevik dağıtım ve kod kapsamı test ölçümlerine ilişkin bir inceleme yapılmıştır. Bu çalışmada, günümüzdeki başarılı gömülü kritik yazılımların çevik ve sürekli gelişim ve test prosedürlerine olan ihtiyacını ele almıştır. Önyükleyici yazılımını doğrulamak için gerçekleştirilen birim testler ile elde edilen kod kapsama ölçümleri çalışmada sunulmuştur (Parra vd., 2018).

Önyükleyici yazılım, proje açısından hayati bir öneme sahiptir ve doğrulanması gereken erken geliştirme aşamasında titizlikle ele alınmalıdır. Herhangi bir test durumunun gözden kaçırılması, yazılımın kalitesini olumsuz etkileyebilir. ESA tarafından benimsenen ECSS standartlarına göre, bu tür kritik yazılımların test edilmesi, kaynak kod ifadelerinin (%100 satır kapsama oranı) ve dallarının (%100 dal kapsama oranı) tamamını kapsamalıdır (Parra vd., 2018). Bu sayede, uzay ortamının neden olabileceği her olası bellek bozulması veya iletişim hatasını ele almak üzere hata toleransı ve kurtarma mekanizmalarının tam anlamıyla test edilmesi sağlanır.

Makalede, test ortamının model güdümlü bir mühendislik çerçevesine ve hata enjeksiyon mekanizmasına dayanan sanal bir platformda gerçekleştirildiği belirtilmiştir. Bu yöntemle, kaynak kodun %100 kapsanabilirliği mümkün hale gelmiştir. Hata enjeksiyonu için Leon2Vip Sanal platformu kullanılırken, kaynak kod kapsama analizi için GCC'nin

GCOV aracı tercih edilmiştir (Parra vd., 2018). Sürekli geliştirme ve test prensiplerine uyularak, erken hata tespiti, gerileme sorunlarından kaçınma, otomatik rapor oluşturma ve gereksinimlere uygun olmayan kodun tespiti gibi birçok avantaj elde edilmiştir.

Gelecekteki çalışmalarda, gömülü yazılım alanında kod kapsama odaklı bir yaklaşım kullanılarak otomatik test senaryolarının oluşturulabileceği belirtilmiştir. Parra vd.ne (2018) göre otomatik test kodunun üretilmesi, büyük projelerde test kodunu geliştirmek için gereken çabadan daha etkili olabilir. Test ortamının oluşturulması, testlerin yürütülmesi, sonuçların gözlemlenmesi ve her testin tekrarlanması gibi tekrar eden görevlerin otomatikleştirilmesi, kapsama verilerinin en kısa sürede elde edilmesini sağlayacaktır. Ancak, sistem ne kadar otomatikleştirilirse de test sürecinin yürütülmesi için insan dokunuşları gerekliliğini korumaktadır.

2.1. Literatür Araştırmasının Analizi ve Tezin Sağladığı Katkılar

Gömülü sistemlerde önyükleyici yazılım tasarımıyla ilgili literatürde yapılan çalışmalar genellikle güvenilirlik ve hataya dayanıklılık odaklıdır. Bu çalışmalar, önyükleyici yazılımın sistemi başlatma sürecinde karşılaşabileceği potansiyel hataları önlemek veya bunlarla başa çıkmak için çeşitli güvenlik önlemlerini ve mekanizmalarını ele almaktadır. Bunlar arasında EDAC, CRC, TMR gibi mekanizmalar bulunmaktadır. Ayrıca, hafıza testleri gibi teknikler de kullanılarak önyükleyici yazılımın hata toleransı ve güvenilirliği artırılmaya çalışılmaktadır. Bu tür güvenlik önlemleri ve mekanizmaları, özellikle havacılık, uzay endüstrisi ve diğer kritik gömülü sistemlerde, sistemin güvenli ve hataya karşı dirençli bir şekilde başlatılmasında hayati öneme sahiptir.

Literatürdeki araştırmalar, önyükleyici yazılımın tasarımı ve geliştirilmesi sürecinde çeşitli güvenilirlik odaklı tasarımların hata enjekte edilerek test edilmesine odaklanmaktadır. Bu tasarımlar genellikle sistemi radyasyon etkilerinden koruma, tekil hata etkilerine karşı dayanıklılık sağlama gibi yönleri içerirken, aynı zamanda testler ve kod kapsama analizleri ile analiz edilmiştir. Bu tür testler, önyükleyici yazılımın güvenilirliğini artırmak ve hata durumlarında doğru davranışı sağlamak için önemlidir. Hata enjeksiyonu, yazılımın beklenmedik durumlarla başa çıkma yeteneğini değerlendirmek ve geliştirmek için etkili bir

yöntemdir. Bu sayede, önyükleyici yazılımın gerçek dünya koşullarında nasıl performans göstereceği daha iyi anlaşılabilir ve güvenilirlik artırılabilir.

Bu tez çalışması, literatürdeki önyükleyici yazılım tasarımına yönelik araştırmalarla karşılaştırıldığında aşağıdaki gibi belirgin farklılıklar sunmaktadır:

- Bütünsel Perspektif Sunması: Bu tez, gereksinimlerden başlayarak tasarım, kodlama, doğrulama ve geçerleme aşamalarını içeren bütünsel bir perspektif sunar. Diğer çalışmalar genellikle belirli alanlara odaklanırken, bu tez tüm süreci bir araya getirerek daha geniş bir bakış açısı sağlar. Bu bakış açısı sayesinde, tasarımın her adımında ortaya çıkabilecek potansiyel problemler daha erken tespit edilir ve çözümlenir. Ayrıca, ekip üyeleri arasındaki iletişimi ve iş birliğini güçlendirerek, daha tutarlı ve bütüncül bir tasarım süreci sağlanır. Bu da tasarım sürecinin daha verimli ve tutarlı olmasına zemin hazırlar, zaman ve kaynakların daha etkin kullanılmasına olanak tanır.

- Ek Fonksiyonelliklerin Sunulması: Hataya dayanıklılık ve güvenilirlik açısından yapılan değerlendirmeler, literatürdeki diğer çalışmalarla benzer hedefleri paylaşsa da bu tez öne çıkan ek fonksiyonelliklere dikkat çeker. Özellikle, önyükleyici yazılımın iki aşamada gerçekleştirilmesi, farklı uçuş yazılımı imajlarını seçme olanağı sunması ve yazılım bakım faaliyetlerinde hata giderme özelliğini içermesi gibi ek özellikler, bu tezi diğerlerinden ayırır. Bu ek fonksiyonellikler, önyükleyici yazılımın daha esnek ve adaptif olmasını sağlar. İki aşamada gerçekleştirilmesi ve farklı yazılım imajlarını seçme olanağı sunması, sistemin değişen gereksinimlere ve koşullara daha iyi uyum sağlamasını sağlar. Yazılım bakım faaliyetlerinde hata giderme özelliği ise sistemdeki hataların daha hızlı ve etkili bir şekilde düzeltilmesine olanak tanır. Daha güvenilir ve esnek bir önyükleyici yazılım tasarımı, sistemdeki potansiyel hataların erken tespit edilmesini ve düzeltilmesini sağlar. Bu da uzun vadede bakım maliyetlerini azaltır ve sistem sahiplerine maliyet tasarrufu sağlar.

- Güncelleme ve Esneklik Yeteneği: Özellikle, uydunun yörüngede bulunduğu süreçte ortaya çıkabilecek yazılım hatalarını düzeltmek amacıyla önyükleyici yazılım aracılığıyla uçuş yazılımının yamalanarak güncellenmesi özelliği, bu çalışmanın odak noktalarından biridir. Bu özellik, mevcut yazılım güncelleme yöntemlerine alternatif bir

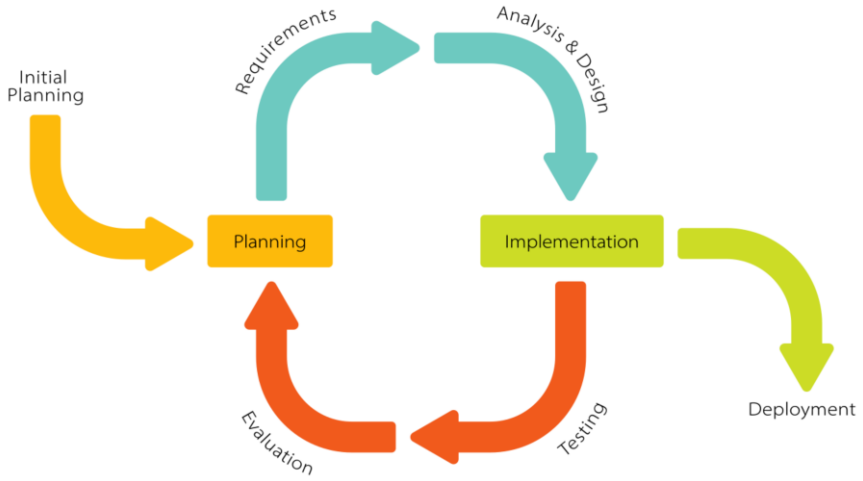
çözüm sunmayı hedeflerken aynı zamanda uydunun işleyişinde esneklik sağlamayı amaçlar. Önyükleyici yazılımın güncelleme özelliği, sistemdeki hataların ve güvenlik açıklarının daha hızlı bir şekilde giderilmesine imkân verir. Bu da sistemin daha güvenilir ve güvenli olmasını sağlar. Ayrıca, güncelleme yeteneği, sistemin işlevselliğinin zamanla artırılmasına da olanak tanır ve uzun vadeli performansını artırır. Bu da sistemdeki riskleri minimize eder.

Sonuç olarak, bu tez çalışması önyükleyici yazılım tasarımında yeni bir perspektif sunarak sistemlerin daha güvenilir, esnek ve güvenli hale gelmesine katkı sağlamaktadır. Bu kapsamlı yaklaşım, uydu sistemleri ve gömülü sistemler alanında önemli ilerlemelere ve akademik literatüre değerli bir katkı sunabilir. Çalışmanın sunduğu fikirler ve yöntemler gelecekteki araştırmacılar için ilham kaynağı olabilir ve endüstriyel uygulamalarda da kullanılabilir. Bu tezin sonuçları, önyükleyici yazılım tasarımı ve geliştirmesinde yapılan çalışmalara yeni bir bakış açısı getirmekte ve sektördeki güncel ihtiyaçlara yanıt verme potansiyeline sahiptir.

3. ÖNYÜKLEYİCİ YAZILIMIN GELİŞTİRİLME SÜRECİ

Önyükleyici yazılımın geliştirilmesi sürecinde, yinelemeli ve artırımlı yazılım geliştirme yöntemleri kullanılmıştır. Bu metodoloji, yazılım geliştiricilerin önceki iterasyonlarda elde edilen deneyimlerden faydalanarak, sistemin sürekli olarak daha küçük parçalar halinde (artırımlı) ve tekrar eden döngüler (yinelemeli) şeklinde geliştirilmesine olanak tanır. Öğrenme hem geliştirme sürecinden hem de sistemin kullanımından kaynaklanır. Bu süreç, yazılım gereksinimlerinin bir alt kümesinin basit bir şekilde uygulanmasıyla başlar ve tam sistem uygulanana kadar tekrarlanan döngülerle geliştirilen sürümlerle devam eder. Her bir yinelemede tasarım değişiklikleri yapılır ve yeni işlevsel yetenekler eklenir (Farcic, 2014).

Yazılım sürecinde temel rol oynayan planlama, gereksinimlerin belirlenmesi, analiz ve tasarım, uygulama, test, değerlendirme ve dağıtım adımları Şekil 3.1'de gösterilmiştir (Wikimedia Foundation, 2023). Adımlar, tekrar eden döngülerle ve her döngüde daha küçük parçalar halinde artırımlı bir şekilde geliştirilir.



Şekil 3.1. Yinelemeli ve Artırımlı Yazılım Geliştirme Süreci

Yinelemeli ve artırımlı geliştirme terimi yazılım endüstrisinde ortaya çıkmış olmakla birlikte, birçok donanım ve gömülü yazılım geliştirme çabası bu teknikleri sıkça kullanmaktadır. Son zamanlarda, bu yöntem, uzay fırlatma endüstrisinde hızlı ve kapsamlı teknolojik yeniliklerin önünü açan önemli bir düşünce değişikliğine yol açmıştır.

4. ÖNYÜKLEYİCİ YAZILIMIN GEREKSİNİMLERİNİN BELİRLENMESİ

Önyükleyici yazılımın gereksinimlerinin belirlenmesi sürecinde, hedeflenen uydu platformu ve üzerinde faaliyet göstereceği donanım ve hafıza alanları detaylı bir şekilde incelenmiştir. Bu inceleme, ihtiyaçların belirlenmesi ve daha iyi anlaşılması için önemlidir. Bu bağlamda, öncelikle alçak yörüngede konumlanan ve yer gözlemi amacıyla kullanılan uydu sistemleri ve yazılımın işleyeceği uydu bilgisayarı özelliklerine değinmek gerekmektedir.

4.1. Yer Gözlem Uydu Sistemleri

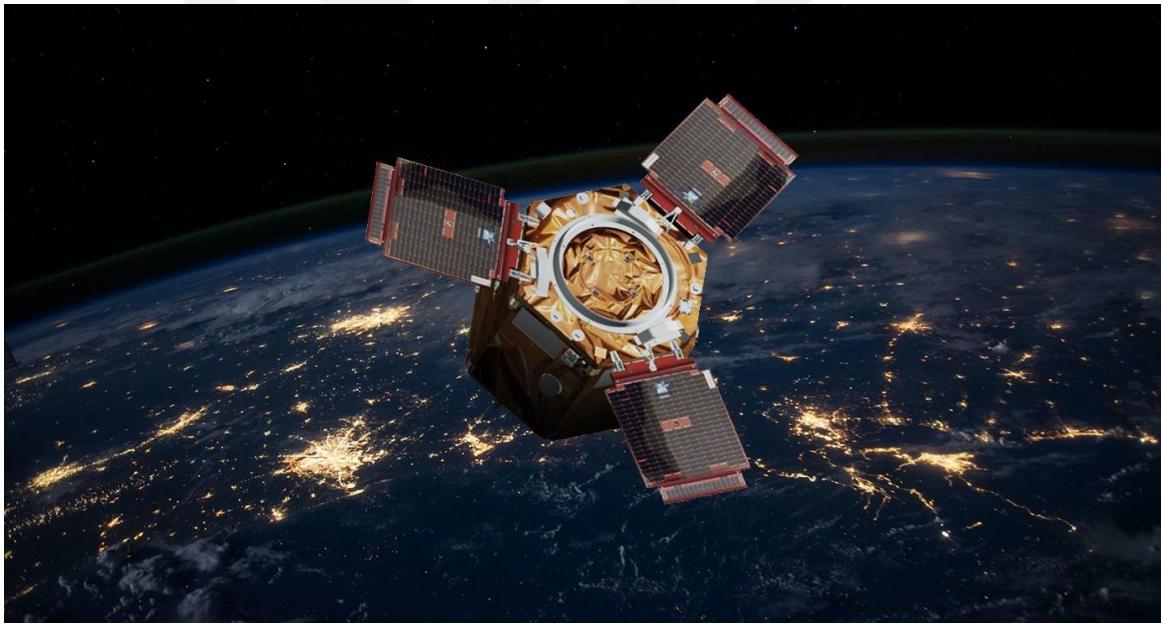
Yörünge yüksekliği 2000 kilometreye kadar olan uydu sistemleri, genellikle Alçak Dünya Yörüngesi (LEO) olarak bilinir. 2011 itibariyle, LEO yörüngede 8000'den fazla nesne bulunmaktadır. Bu nesnelerin büyük bir kısmı uyduları oluştururken, eski roket parçaları, hasar görmüş uydu parçaları ve diğer küçük parçalar da bu alanda yer almaktadır (Williams, 2017).

Alçak yörünge uyduları genellikle yer gözlem amaçları için kullanılır. Bu amaçla, dünya üzerindeki olayları uzaydan gözlemlemek, afet sonrası durumları incelemek, belirli bölgeleri görüntülemek ve zaman içinde değişimleri analiz etmek gibi farklı alanlarda kullanılmaktadır. Ayrıca, stereo görüntüleme kabiliyetine sahip olan uydular, üç boyutlu uydu görüntüleri elde etmek ve bu görüntülerden yer kabuğunun topografik haritalarını oluşturmak için de kullanılır. Tarım, orman yönetimi, madencilik, imar planları gibi birçok alanda kullanılan bu teknoloji, son 15 yılda yaygınlaşmıştır (Çoban, 2016; Bayır, 2013).

Son yıllarda, yeni teknolojiler ve internet uygulamalarının yaygınlaşması ile bu hizmetler geniş kitlelere ulaşılabilir hale gelmiştir. Örneğin, 15 yıl öncesine kadar belirli bir bölgede yüksek çözünürlüklü hava fotoğrafı çekmek neredeyse imkansızken, bugün 41 cm çözünürlükle dünyayı turlayan optik uydular sayesinde herhangi bir yer kolaylıkla görüntülenebilir hale gelmiştir (Bayır, 2013).

Türkiye'de de optik gözlem uyduları alanında son 10 yılda önemli gelişmeler kaydedilmiştir. Örneğin, TÜBİTAK Uzay tarafından Surrey Satellite Technology şirketi ile yapılan iş birliğiyle Bilsat uydusu LEO yörüngeye yerleştirilmiştir. Daha sonra TÜBİTAK tarafından Rasat projesi hayata geçirilmiş ve uydu 2011 yılında uzaya gönderilmiştir. Bilsat ve Rasat görev ömrünü tamamlamıştır (TÜBİTAK UZAY, 2023).

2007 yılında Göktürk-2 Yüksek Çözünürlüklü Yer Gözlem Uydusu projesine başlanmış ve 2,5 metre çözünürlüğe sahip uydu Türk Havacılık ve Uzay Sanayii A. Ş. (TUSAŞ) ve TÜBİTAK Uzay Teknolojileri Araştırma Enstitüsü (TÜBİTAK UZAY) ortaklığında milli olarak geliştirilmiştir. 2012 yılı sonunda uzaya gönderilen uydu görüntü çekim işlemlerine başlamıştır. Şekil 4.1'de Göktürk-2 uydusunu görseli bulunmaktadır (Marangoz, 2020; TUSAŞ, 2023).



Şekil 4.1. Göktürk-2 Uydusu

Göktürk-2'nin başarısı, Türkiye'deki optik gözlem uyduları alanında yeni bir dönemin başlangıcını işaret etmiş ve bu projenin kazanımları diğer gözlem uydularının yapım kararlarını hızlandırmıştır.

2009 yılında başlayan Göktürk-1 Yer Gözlem ve Keşif Uydusu ana yükleniciliği Telespazio-İtalya ve Thales Alenia Space-Fransa firmaları sorumluluğundadır ve 50cm

yersel çözünürlüğe sahiptir. Aralık 2016'da Fransız Guyanası'ndan uzaya başarıyla fırlatılmıştır (TUSAŞ, 2023).

2017 yılında başlayan İMECE Uydu projesi ile metre altı çözünürlüklü kendi elektro-optik kamerası ile yüksek çözünürlüklü görüntüleme hedeflenmiştir. Ayrıca geliştirilen Yıldız İzler, Güneş Algılayıcı vb. birçok ekipmanın uyguda kullanılması ile tarihçe kazandırılması hedeflenmektedir. 15 Nisan 2023'te fırlatma işlemi başarı ile gerçekleştirilmiştir (TÜBİTAK UZAY, 2023).

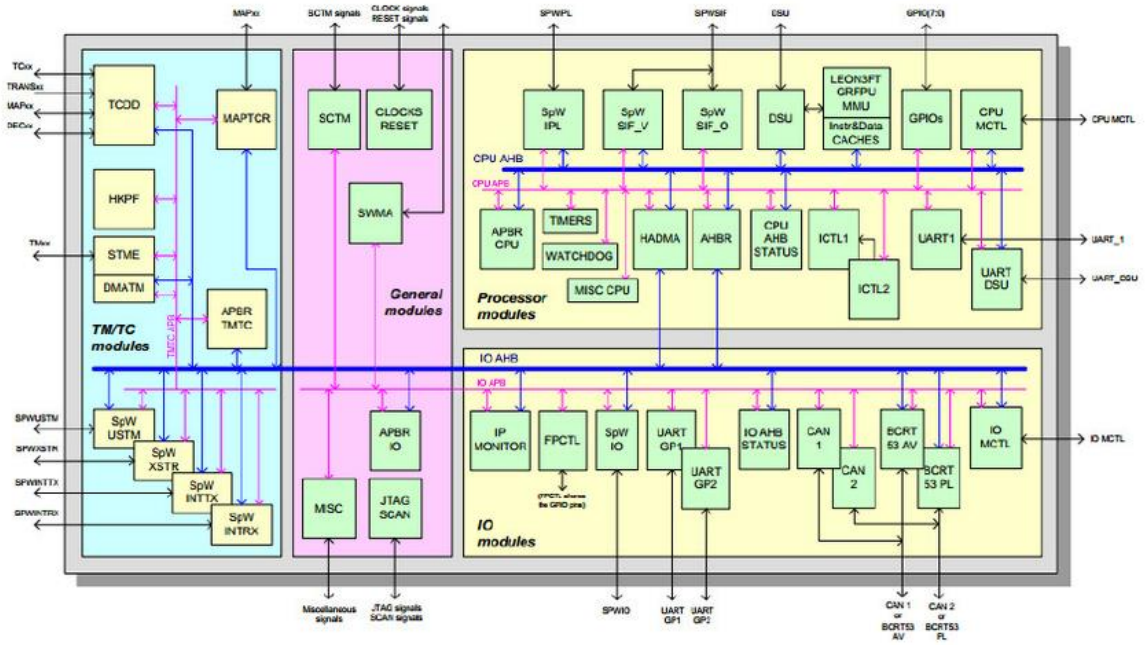
2013 yılında başlayan Göktürk-3 Sentetik Açıklı Radar (SAR) Yer Gözlem ve Keşif Uydu Sistemi ile Dünya üzerinden herhangi bir bölgenin metre-altı çözünürlüklü olarak gece ve gündüz şartlarında görüntülenmesi kabiliyetinin kazanılması hedeflenmiştir. Ön Tasarım Aşamasına kadar tüm faaliyetler TUSAŞ ana yükleniciliğinde, Aselsan ve TÜBİTAK UZAY desteğiyle yürütülmüş ve Mayıs 2016'da başarı ile tamamlanmıştır. Fırlatmanın 2025 yılında gerçekleştirilmesi planlanmaktadır (TUSAŞ, 2023).

2021 yılında başlayan Göktürk Yenileme Keşif Gözetleme Uydu Sistemi Geliştirme (Göktürk-Y) Projesi; Göktürk-1 Uydusunun görev ömrü sona erdiğinde de devam ettirilmesine yönelik olarak başlatılmıştır. Proje TUSAŞ ana yükleniciliğinde gerçekleştirilmektedir (TUSAŞ, 2023).

4.2. Uydu Uçuş Bilgisayarı

Uydu sistemlerinde, uydu bilgisayarı genellikle uydunun komuta ve kontrolünden sorumlu olan birimdir. Yedekli bir yapıda tasarlanan uydu bilgisayarı, yedeklilik geçişleri, mod geçişleri, girişlerin ve çıkışların kontrolü gibi işlevleri genellikle Field Programmable Gate (FPGA) tarafından gerçekleştirilir.

Uydu bilgisayarının donanım tasarımı içerisinde SPARC V8 mimarisine sahip LEON3-FT işlemcisi bulunan SCOC3 ASIC kullanılmaktadır (GRLIB, 2012). Şekil 4.2'de SCOC3 ASIC blok diyagramı gösterilmektedir (Furano vd., 2013).



Şekil 4.2. SCOC3 ASIC Blok Diyagramı

Uydu bilgisayarında çalışan önyükleyici yazılım, çeşitli denetimleri sağladıktan sonra temel işlevi olan uçuş yazılımını yüklemekten sorumludur. Uçuş yazılımı, uydunun operasyonları için gerekli olan komut verme, veri alımı, depolama ve otonomi yeteneklerini sağlar. Bu sayede, uydudaki güç, termal kontrol, yörünge yönlendirme, haberleşme, faydalı yük gibi tüm alt sistemlerin kontrolünü sağlar.

4.3. Uydu Spesifik İhtiyaçlar

Uydu Spesifik İhtiyaçlar, önyükleyici yazılımın uydu platformuna yönelik gereksinimlerini içerir ve aşağıdaki gibi özetlenir:

Kalıcı Bellek Kullanımı:

- Önyükleyici yazılım, doğrudan uydu bilgisayarı kalıcı belleğinden çalışabilmelidir.
- Tekrar başlatma durumunda, önyükleyici yazılım uydu bilgisayarı kalıcı belleğinden çalışabilir olmalıdır.

Geçici Bellek Kullanımı:

- Yazılım, uydu bilgisayarı kalıcı belleğindeki içeriği geçici bellek alanına kopyalayarak uçuş yazılımını bu geçici bellek üzerinden çalıştırmalıdır.

- Uydu bilgisayarı üzerinde çalışacak olan uçuş yazılımını yüklemek için yamalama işlemi yapılmalı ve bu yolla yazılımın geçici bellek üzerinden çalışması sağlanmalıdır.

Başlatma Süresi Kısıtları:

- Uydu bilgisayarı başlatma süresi, uydu ve proje spesifik olarak belirlenen maksimum sınırlar içinde olmalıdır.

Bu ihtiyaçlar, önyükleyici yazılımın uydu sistemine entegrasyonu ve uydu bilgisayarının işlevselliği için gerekli olan temel özellikleri ve sınırlamaları belirtir.

4.4. Donanım Bağımlı İhtiyaçlar

Önyükleyici yazılımın donanım bağımlılıkları aşağıdaki gibidir:

SCOC3 ASIC ve LEON3FT İşlemcisi Bağımlılığı:

- Önyükleyici yazılım, SCOC3 ASIC ve LEON3FT işlemcilerine bağımlıdır. Bu donanım bileşenleri, önyükleme sürecinin ve yazılımın uygun bir şekilde çalışabilmesi için temel donanım gereksinimleridir.

Bu bağımlılıklar, önyükleyici yazılımın belirli donanım bileşenlerine ihtiyaç duyduğunu ve bu donanım yapılarına bağlı olarak geliştirilmesi gerektiğini ifade etmektedir. Bu gereksinimler, yazılımın uyumlu bir şekilde çalışabilmesi için donanım bileşenlerinin önyükleyici yazılım tarafından desteklenmesi gerektiğini vurgular.

4.5. Kısıtlar

Bu bölümde, önyükleyici yazılımın karşılaştığı kısıtlar aşağıdaki gibidir:

ÖYF-1, ÖYF-2 ve UY İmajlarının Boyut Kısıtları:

- ÖYF-1, ÖYF-2 ve UY imajlarının boyutu, kullanılan bellek boyutuyla kısıtlanmaktadır.

Yama Sayısı ve Alanı Kısıtları:

- Yama sayısı ve yama alanı için kullanılan bellek boyutu, yazılım güncellemeleri ve yama işlemleri için kısıt oluşturmaktadır.

Kalıcı Bellekteki Konfigürasyon Bilgileri Kısıtları:

- Kalıcı bellekteki konfigürasyon bilgileri için kullanılan bellek boyutu, kısıtlar içinde yer almaktadır.

Bu kısıtlar, önyükleyici yazılımın bellek boyutu ve kapasitesiyle ilgili olarak önemli sınırlamaları ifade etmektedir.

5. ÖNYÜKLEYİCİ YAZILIMIN TASARIMI

Önyükleyici yazılımlar, bir bilgisayar veya bir sistem çalıştırılmaya başlamadan önce yüklenen yazılım parçalarıdır. Uydu sistemleri de bu kapsamda önyükleyici yazılımlara sahiptir ve çeşitli tasarım yöntemleri kullanılmaktadır. Bu yöntemler genellikle tek aşamalı ve çok aşamalı önyükleme olarak adlandırılır.

Tek aşamalı önyükleme, tüm işlemleri sıralı olarak gerçekleştiren bir yaklaşımdır. Yani, önyükleme sürecinde gerekli tüm görevler tek seferde tamamlanır ve uçuş yazılımı çalıştırılıncaya kadar işlem devam eder. Bu tasarım, basitlik açısından avantaj sağlasa da performans açısından bazı sınırlamalar getirebilir. Özellikle kalıcı bellekte çalışması gerektiği durumlarda performans sorunları ortaya çıkabilir.

Uydu sistemlerinde kullanılan önyükleyicilerde tercih, genellikle sistemin gereksinimlerine ve performans beklentilerine dayanmaktadır. Özellikle, performansın kritik olduğu durumlarda çok aşamalı önyükleme tercih edilebilir. Bu yöntem, işlemleri daha verimli bir şekilde gerçekleştirerek performans sınırlamalarını azaltabilir. Ancak, tasarım karmaşıklığını artırabilir ve buna bağlı olarak geliştirme sürecini uzatabilir.

Bu bağlamda, tek aşamalı önyükleme basitliğiyle tercih edilirken, çok aşamalı önyükleme performans ve esneklik açısından tercih edilebilir. Seçim, özellikle uydunun ihtiyaçlarına ve işlevselliğine bağlı olarak yapılır.

Bu çalışmada, uydu sistemlerinde kullanılan önyükleyici yazılımların gereklilikleri doğrultusunda iki aşamalı bir yapıda gerçekleştirilmesine karar verilmiştir. İki aşamalı önyükleme yöntemi, sistemin performansını ve esnekliğini artırmak amacıyla benimsenmiştir. Bu yaklaşım, özellikle performansın kritik olduğu durumlarda işlemleri daha verimli hale getirerek performans kısıtlamalarını azaltmaya yönelik bir adım olarak değerlendirilmiştir. Bu şekilde, önyükleme sürecindeki performans ihtiyacı ve gereksinimler, iki aşamalı bir tasarımın seçilmesinde belirleyici olmuştur.

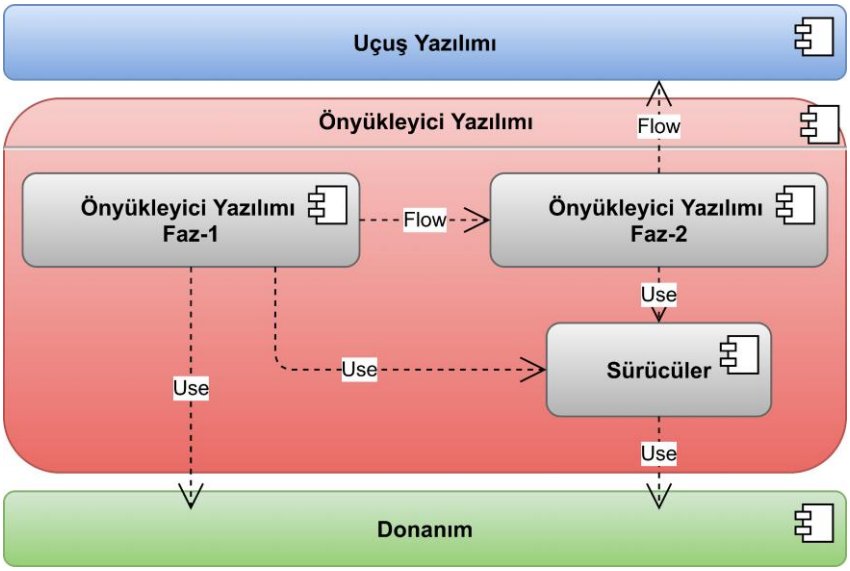
5.1. Mimari Tasarım

Önyükleyici yazılımların iki aşamalı yapısının, mimari tasarımdaki donanım ve uygulama katmanları ile olan ilişkisi incelenmiştir.

Yazılımlar, donanım katmanına sürücüler aracılığıyla erişim sağlamaktadır. Bu sürücüler, yazılımın çalıştığı donanım özelliklerine bağlı olarak belirlenir ve işlev görür. İlk aşama önyükleyici yazılım, en temel işlevlerin yerine getirilebilmesi için yazılımın çalışacağı ortamı oluşturur. Bu aşamada, C çalışma zamanındaki sürücüler kullanılamadığından dolayı doğrudan yazmaçların başlangıç değerlerini belirlemek amacıyla ilklendirmeler gerçekleştirilir.

Önyükleyici yazılımlar, farklı uçuş yazılımı imajlarından ve işletim sistemlerinden bağımsız olarak çalışacak şekilde tasarlanmıştır. Bu yaklaşım, çeşitli uçuş yazılımı imajlarını, içeriklerinden bağımsız olarak sistemin çalışmasından önce seçebilme ve uygulayabilme esnekliği sunar.

Mimari tasarımda yazılım akışı sırası ile ÖYF-1, ÖYF-2 ve UY sıralamasında Şekil 5.1'de gösterilmektedir.



Şekil 5.1. Önyükleyici Yazılımı Mimari Tasarımı

5.2. Detaylı Tasarım

Mimari tasarım aşamasında ortaya konulan ÖYF-1 ve ÖYF-2 içerisinde geliştirilen modüller, sınıflar, metotlar, akışlar ve bunlar arasındaki ilişkiler, detaylı tasarım sürecinde detaylı bir şekilde ele alınmaktadır. Bu adım, mimari tasarım ve kodlama arasında köprü oluşturarak, geniş bir çerçevede detaylı tasarım ve kodlama sürecini birleştirmektedir. Detaylı tasarım, kodlama sürecinin çerçevesini belirleyerek bu sürecin ilerlemesine yol açmaktadır.

5.2.1. Uydu bilgisayarı önyükleyici yazılım faz-1

ÖYF-1, genellikle donanımın başlangıç sürecini yönetir ve temel çalışma ortamını sağlamaktan sorumludur. Bu aşama, donanımın öncelikle başlangıç aşamalarında gereken tüm temel işlemleri gerçekleştirir. Asgari çalışma ortamının temin edilmesi için bellek zamanlamalarının düzenlenmesi hem yazılımsal hem de donanımsal kesmelerin ayarlanması, işlemci yazmaçlarının ilk değerlerle doldurulması, önbellek ve yığın belleklerin hazırlanması gibi işlemleri içerir.

Kalıcı belleğin farklı yonga seti boyutlarına uyum sağlayabilmesi, yazılımın derlendiği yonga seti ile uyumlu olup olmadığının kontrol edilmesini gerektirir. Bu, bellek boyutlarındaki değişikliklere göre hafıza adres aralıklarının nasıl değiştiğini değerlendirir.

ÖYF-2 yazılımının yüklenmesi işlemi, ÖYF-2 imajının kalıcı bellekteki boyutunu ve bütünlüğünü kontrol etmek için yapılan testlerin ardından gerçekleşir. Eğer imaj boyutu ve bütünlüğü doğrulanırsa, ÖYF-2 imajı geçici bellekte çalışmaya hazır hale getirilir.

ÖYF-1 aşamasındaki adımları daha detaylı bir şekilde açıklamak gerekirse:

- ÖYF-1 İlklendirme: Bu aşama, donanımın başlangıç sürecini yönetir. İşlemci, bellek ve diğer temel donanım bileşenleri bu aşamada ilklendirilir. Bellek zamanlamaları düzenlenir, donanım kesmeleri (interrupts) ve yazılımsal kesmeler (software interrupts) ayarlanır. İşlemci yazmaçları, ilk değerlerle doldurulur. Önemli bellek bölgeleri olan önbellek ve yığın bellekler de bu aşamada başlatılır.

- ÖYF-1 İmajı Bütünlük Kontrolü: Kalıcı bellekte depolanan ÖYF-1 imajının bütünlüğü ve doğruluğu burada kontrol edilir. İmajın doğru biçimde depolandığı, uygun formatta olduğu ve beklenen verilere sahip olduğu doğrulanır. Böylece, imajın eksiksiz ve hatasız olduğu teyit edilir.

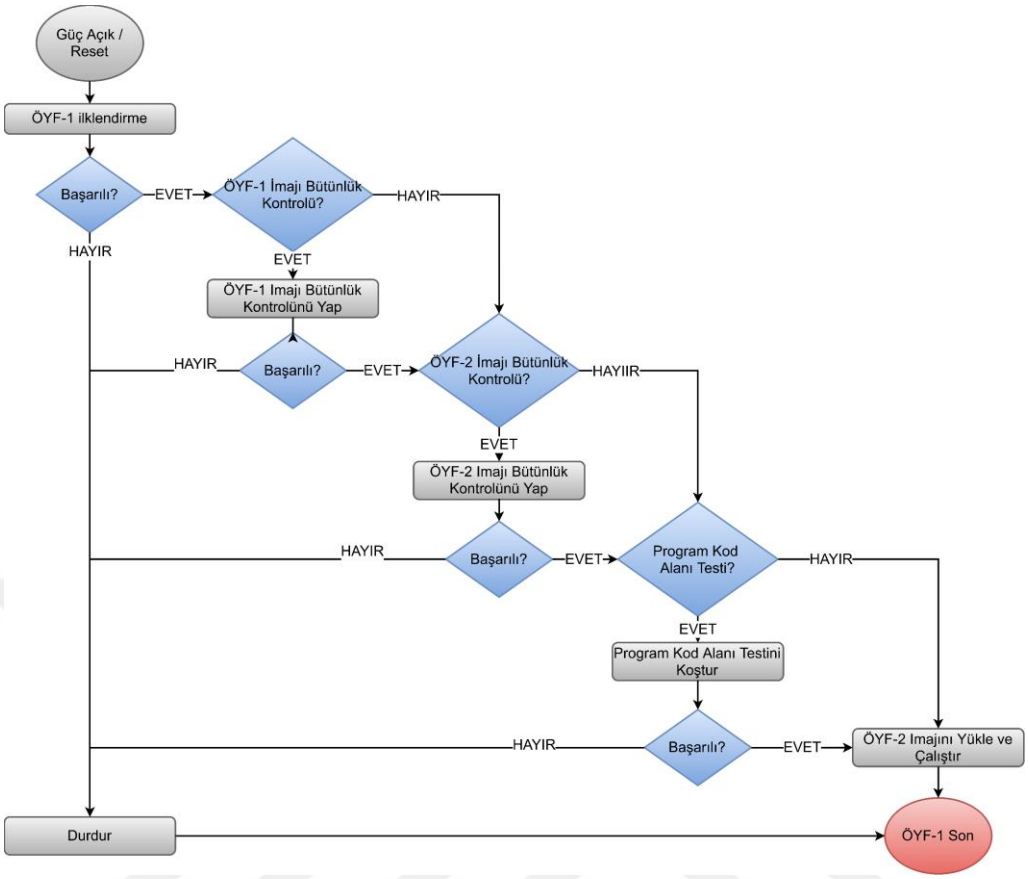
- ÖYF-2 İmajı Bütünlük Kontrolü: ÖYF-2 imajının kalıcı bellekteki bütünlüğü ve uygunluğu burada incelenir. ÖYF-2 imajının da ÖYF-1 gibi doğru biçimde depolandığı ve gereken özelliklere sahip olduğu teyit edilir.

- Program Kod Alanı Testi: ÖYF-1 ve ÖYF-2 imajlarının yüklendiği program kod alanlarının geçerliliği ve işlevselliği test edilir. Bu aşamada, yüklenen kodun doğru biçimde çalışacağından emin olunur.

- ÖYF-2 İmajı Yükle ve Çalıştır: ÖYF-2 imajı, ÖYF-1 aşamasındaki bütünlük ve uygunluk testlerini geçtikten sonra geçici belleğe yüklenir ve çalıştırmaya hazır hale getirilir. Eğer yama ayarlanmışsa, bu aşamada ÖYF-2 imajına yama uygulanır ve işletilmeye başlanır.

Bu adımlar, sistemin başlatılması ve önemli bileşenlerinin başlatılmasını içeren temel aşamalardır. Bu süreçlerin eksiksiz ve hatasız gerçekleştirilmesi, uydu sisteminin istikrarlı ve güvenilir çalışmasını sağlamak için kritik öneme sahiptir.

ÖYF-1 aşamasında alınan kararlar ve gerçekleştirilen işlemler, Şekil 5.2'deki akış diyagramında detaylı olarak gösterilmektedir.



Şekil 5.2. ÖYF-1 Akış Diyagramı

5.2.2. Uydu bilgisayarı önyükleyici yazılım faz-2

ÖYF-2, genellikle cihaz içi testlerin gerçekleştirilmesi, yazılım yamalarının uygulanması ve uçuş yazılımının yüklenmesinden sorumludur. Uçuş yazılımının yüklenme işlemi sırasında, kalıcı bellekte yer alan UY imajının boyutu, bütünlüğü ve yama kontrolleri öncelikle yapılır. Eğer yama belirlenmemişse, UY imajı geçici bellekte çalışmaya hazır hale getirilir. Yama belirlenmişse, UY imajı geçici belleğe kopyalanır ve belirlenen yama bilgileri çekilir. Ardından yama işlemi gerçekleştirilir ve yama uygulanmış UY imajı geçici bellekte çalışmaya hazır hale getirilir. Bu şekilde, yama uygulanmış ve uygulanmamış durumlarına göre farklı süreçler izlenir.

ÖYF-2 aşamasındaki adımları daha detaylı bir şekilde açıklamak gerekirse:

- ÖYF-2 İlklendirme: Bu adım, ÖYF-1 aşamasından devralınan kontrolleri sürdürür ve cihazın daha karmaşık sistemlerini başlatır. Önceki aşamalarda ilklendirilen donanımın

devamı niteliğindedir. Bu adım, daha detaylı ve özelleşmiş donanım işlevlerinin başlatılmasını içerir.

- UY İmaj Seçimi: Bu aşamada, uyduya yüklenecek olan UY imajının seçimi yapılır. Birden fazla UY imajı mevcut olabilir ve bu aşamada hangi imajın kullanılacağı belirlenir. Bu, sistem ihtiyaçları, performans, güvenilirlik veya diğer gereksinimler doğrultusunda yapılan bir seçim olabilir.

- Cihaz İçi Testler (Basit-Detaylı Test): Bu adım, cihazın içindeki donanım ve yazılım bileşenlerini test etme sürecini içerir. Cihaz içi testler aşamasında, test süreci cihazın gereksinimlerine ve test amacına bağlı olarak iki seviyede gerçekleştirilebilir: Basit seviye ve Detaylı seviye testler.

    - Basit Seviye Testler: Bu testler, cihazın ana işlevlerini genel olarak kontrol eder. Donanım ve yazılım bileşenlerinin temel işlevselliğini doğrular. Bu seviyedeki testler, genellikle temel işlevlerin sağlıklı bir şekilde çalışıp çalışmadığını görmek için yapılır.

    - Detaylı Seviye Testler: Bu testler, daha kapsamlı bir inceleme sunar ve cihazın daha karmaşık ve derin işlevlerini test eder. Bu aşamada, daha karmaşık algoritmalar, veri iletim protokolleri, özel donanım modülleri ve daha ayrıntılı sistem bileşenleri üzerinde detaylı testler gerçekleştirilir.

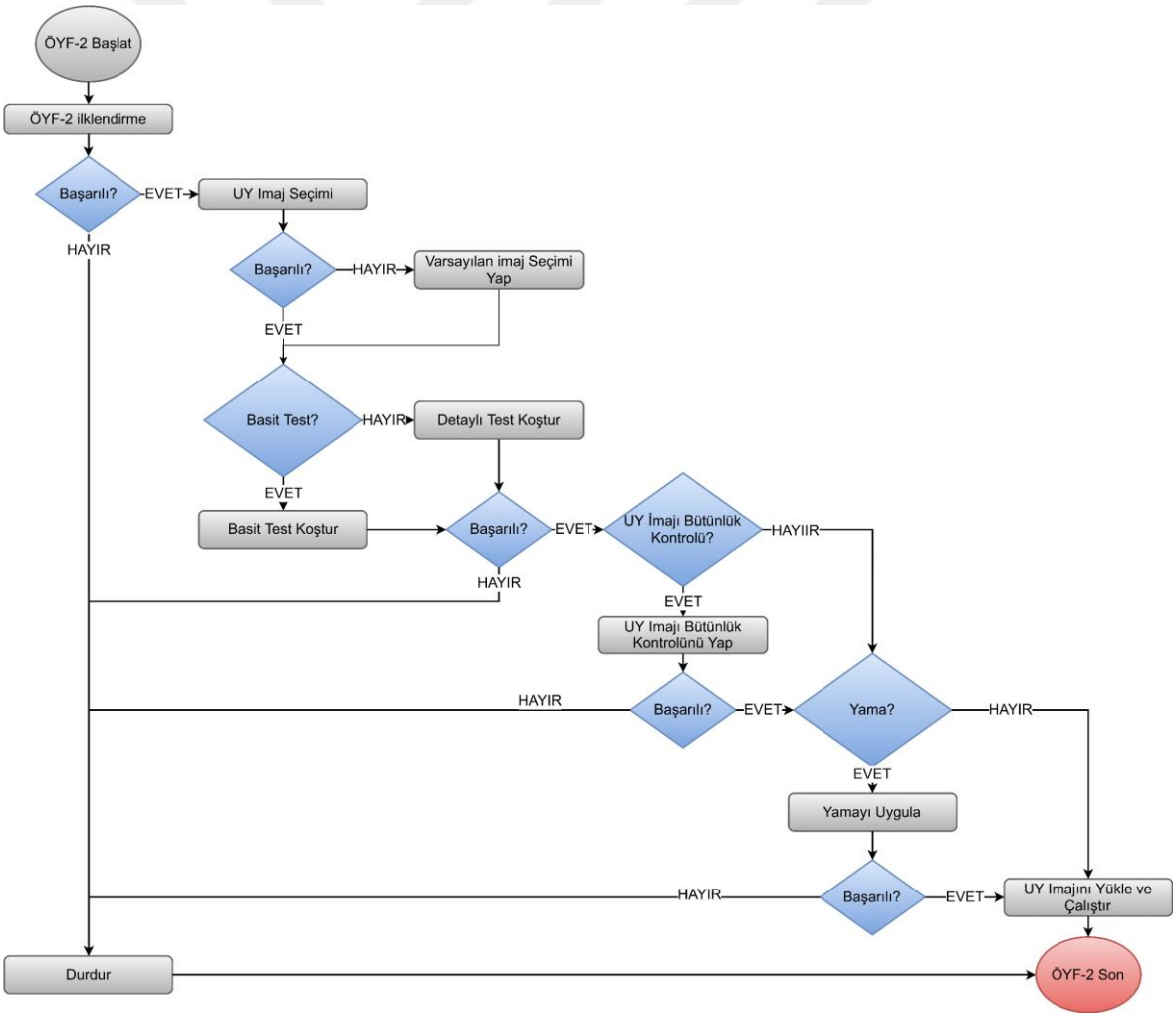
- UY İmajı Bütünlük Kontrolü: UY imajının kalıcı bellekteki bütünlüğü ve uygunluğu bu aşamada kontrol edilir. Önceki aşamalarda olduğu gibi, imajın doğru biçimde depolandığı, gereken özelliklere sahip olduğu ve beklenen verilere sahip olduğu teyit edilir.

- Yama: Bu adımda, eğer bir güncelleme (yama) varsa, bu yama UY imajına uygulanır. Güvenlik açıklarının kapatılması, performans iyileştirmeleri veya diğer yazılım düzeltmeleri gibi nedenlerle yamalar uygulanabilir.

- UY İmajını Yükle ve Çalıştır: UY imajı, tüm önceki adımları geçtikten ve bütünlüğü doğrulandıktan sonra geçici belleğe yüklenir ve çalıştırmaya hazır hale getirilir. Eğer yama uygulanmışsa, bu aşamada güncellenmiş UY imajı geçici belleğe kopyalanır ve işletilmeye başlanır.

Bu adımlar, ÖYF-2 aşamasının önemli işlevlerini ve sistemin çalışabilir duruma gelmesi için gerçekleştirilen adımları temsil eder. Bu süreçler, uydu sisteminin güvenilirliği ve doğruluğu için kritik öneme sahiptir.

ÖYF-2 aşamasında alınan kararlar ve gerçekleştirilen işlemler, Şekil 5.3'teki akış diyagramında detaylı olarak gösterilmiştir.



Şekil 5.3. ÖYF-2 Akış Diyagramı

6. ÖNYÜKLEYİCİ YAZILIMIN UYGULANMASI: YÖNTEMLER VE KODLAMA STRATEJİLERİ

Önyükleyici yazılımın uygulanması, önceden belirlenmiş olan gereksinimlerin ve detaylı tasarım sürecinin tamamlanmasının ardından gerçekleşmiştir. Gereksinimlerin belirlenmesi aşamasında, sistemin işlevsel ve performansla ilgili ihtiyaçları titizlikle gözden geçirilmiş ve uygun şekilde belgelenmiştir. Bu aşama, sistemin donanım ve yazılım bileşenlerini içeren kapsamlı bir analiz sürecini de kapsamıştır.

Tasarım süreci, önyükleyici yazılımın mimarisini, işlevlerini ve bileşenlerini belirlemek için ayrıntılı bir yaklaşımla yürütülmüştür. Önyükleyici sürecinde kullanılacak algoritmalar, veri yapıları ve iletişim protokolleri gibi temel özellikler titizlikle incelenmiş ve bu doğrultuda tasarlanmıştır. Bu aşamada ayrıca, güvenilirlik, güvenlik ve performans gibi önemli unsurlar göz önünde bulundurulmuş ve tasarım kararları bu kriterlere göre alınmıştır.

Gereksinim ve tasarım aşamalarının tamamlanmasının ardından, belirlenen hedeflere ulaşmak üzere kodlama sürecine geçilmiştir. Bu süreçte, önyükleyici yazılımın temel işlevlerini gerçekleştirmek üzere uygun programlama dilleri ve teknikler kullanılarak önyükleme kodları geliştirilmiştir. Bu aşamada, öncelikle başvurulabilecek önyükleme yazılımı kodları detaylı bir şekilde incelenmiş ve bu referans kodları üzerinden önyükleyici yazılımının geliştirilmesine başlanmıştır. Aynı zamanda, önyükleyici yazılımı için test edilebilirlik, bakım kolaylığı ve genel sistem entegrasyonu gibi önemli faktörler de göz önünde bulundurulmuş ve bu yönde programlama süreci yürütülmüştür.

6.1. Referans Yazılımlar

Önyükleyici yazılımın geliştirilmesi sürecinde, LEON3-FT işlemci üreticisi olan Cobham Gaisler firmasının önyükleyici yazılım ürünleri detaylı bir şekilde incelenmiştir. Cobham Gaisler firmasının sunmuş olduğu önyükleyici yazılım ürünleri, tasarım açısından örnek bir nitelik taşımaktadır. Özellikle önyükleyici yazılım ürünleri, LEON3-FT işlemci

mimarisine uygun olarak optimize edilmiş ve donanımın gereksinimlerini karşılamak adına çeşitli avantajlar sunmaktadır.

Cobham Gaisler firmasının önyükleyici yazılım ürünleri arasında aşağıdaki seçenekler yer almaktadır:

- MKPROM2
- GR712RC Boot SW
- GRBOOT- Gaisler Flight Software Boot Loader

Bu ürünler, önyükleyici yazılım geliştirme sürecinde referans alınarak, mevcut mimari ve gereksinimlerin karşılanmasına yönelik önemli bir kaynak teşkil etmektedir.

6.1.1. MKPROM2

MKPROM2, açık kaynak kodlu bir yazılım olup, sistem ilklendirmesi ve uygulama yükleyici (application loader) işlevlerinin birleşiminden oluşan bir yapıya sahiptir (Aberg, 2023).

MKPROM2'nin sunduğu temel adımlar şunlardır:

- Yazılım, Floating Point Unit (FPU) yazmaçlarını başlangıç değerleriyle ilklendirir.

- Hafıza kontrolcüsü, Universal Asynchronous Receiver Transmitter (UART) ve Timer birimlerini, belirli opsiyonlara göre yapılandırarak başlatır.

- Hafızayı EDAC koruma modunda ilklendirir, böylelikle hafıza hatalarına karşı koruma sağlar.

- Geçici hafızaya yüklenecek imajı kalıcı bellekte saklar. Bu, önyükleme işlemi sırasında gereken verilerin muhafaza edilmesini sağlar.

- Uygulama geçici belleğe yüklendiğinde, yığın bellek göstericisini (stack pointer) RAM'ın en son adresine yönlendirir. Bu adım, geçici bellekte yüklü olan uygulamanın düzgün bir şekilde çalışabilmesi için önemlidir.

- MKPROM2'nin kaynak kodları ve kullanıcı kılavuzu, ilgili web sitesinden doğrudan indirilebilir ve incelenebilir.

6.1.2. GR712RC Boot SW

Flight Computer Initialisation Sequence, TEC-SWS/10-373-ESA Gereksinim Dokümanı'nın gerçeklenmesi sonucunda ortaya çıkan bir yazılımdır (Aberg, 2023). Bu yazılım, ECSS-E-ST-40C ve ECSS-Q-ST-80C yazılım standartlarına uygun olarak, kritiklik seviyesi B olarak belirlenmiş şekilde geliştirilmiştir. Bu yazılım, lisanslı bir üründür ve üç ana bölümden oluşmaktadır:

- "Boot": Bu kısım, işlemcinin başlatılmasını (CPU ilklendirmesi) ve cihaz içi testlerin gerçekleştirilmesini içerir. Bu aşama, sistem başlangıcında temel donanımın başlatılmasını ve cihazın başlangıç durumunun sağlanmasını hedefler.

- "Standby": Bu kısmında, Packet Utilization Standard (PUS) standartlarında tanımlı olan yazılım bakım fonksiyonelliğini sağlayacak işlevler bulunmaktadır. Bu kısımda, yazılım bakımı ve yönetimi için gerekli olan fonksiyonlar yer almaktadır.

- "Application Loader": Bu bölümünde ise, geçici belleğe seçilen bir uygulama yazılımı yüklenir ve kontrol bu uygulama yazılımına devredilir. Bu adım, sistemde çalışacak olan uygulama yazılımının seçilmesini ve yüklenmesini sağlar.

6.1.3. GRBOOT- Gaisler Flight Software Boot Loader

GRBOOT, ECSS-E-ST-40C ve ECSS-Q-ST-80C yazılım standartlarına uygun olarak kritiklik seviyesi B düzeyinde geliştirilmiş olup, ESA "SAVOIR Flight Computer Initialisation Sequence" (SAVOIR-GS-002) dokümanındaki tasarım bilgilerine dayanılarak

meydana getirilmiştir (Aberg, 2023). Bu yazılım, lisanslı bir üründür ve dört temel bölümden oluşmaktadır:

- "Initialisation": Bu aşama, sistemin başlatılması sürecinde donanım ve yazılım bileşenlerinin temel ayarlarının yapılmasını içerir. Önyükleme sırasında, işlemcinin başlatılması, bellek ayarlarının yapılması, temel donanım birimlerinin (örneğin, UART, zamanlayıcılar) başlatılması gibi önemli adımlar bu aşamada gerçekleştirilir. Bu adım, sistemin çalışma ortamının oluşturulmasını sağlar.

- "Self-Tests": Bu aşama, donanım ve yazılım bileşenlerinin kendi kendini test etmesini içerir. Bu testler, sistemde olası hata noktalarını tespit etmek ve cihazın sağlıklı bir şekilde çalışıp çalışmadığını doğrulamak için gerçekleştirilir. Bu aşama, sistemdeki potansiyel sorunları belirleyerek daha güvenilir bir işletim ortamı oluşturmayı amaçlar.

- "Standby Extension Point": Bu kısım, sistemdeki belirli durumlarda geçiş yapılabilen bir bekleme modunu ifade eder. Sistem, normal çalışma modundan beklemeye alınabilir ve bu noktada beklemeye geçişler için belirli noktalar belirlenir. Bekleme modu, güç tüketimini azaltabilir veya belirli durumlarda sistem kaynaklarını serbest bırakabilir.

- Application Loader: Bu adım, önyükleme işlemi sırasında geçici belleğe yüklenecek uygulama yazılımının yüklenmesini sağlar. Bu kısımda, seçilen uygulama yazılımı geçici belleğe taşınır ve bu yazılımın çalışması için gerekli olan başlangıç yapılandırmaları yapılır. Uygulama yükleyici aşaması, ana uygulamanın yüklendiği ve çalıştırıldığı aşamadır.

6.1.4. Referans yazılımların değerlendirilmesi

Önyükleyici yazılımların geliştirilme sürecinde MKPROM2 adlı açık kaynak kodlu önyükleyici yazılım, incelenerek önemli bir referans kaynağı olmuştur. MKPROM2'nin kaynak kodları üzerinde yapılan incelemeler, önyükleme sürecindeki temel adımların anlaşılması ve ilgili işlevlerin gerçekleştirilmesi konusunda rehberlik sağlamıştır. Bu yazılım, özellikle FPU yazmaçlarının ilklendirilmesi, hafıza kontrolcüsü, UART ve zamanlayıcı birimlerinin belirli opsiyonlara göre ayarlanması gibi temel işlemleri

içermektedir. MKPROM2'nin açık kaynak kodlu olması, geliştirme sürecinde kodların anlaşılması, özelleştirilmesi ve ihtiyaç duyulan yerlerde değişiklik yapılması açısından faydalı bir kaynak olmuştur.

Diğer yazılımların kaynak kodları erişilebilir olmasa da ilgili konseptler üzerinde yapılan çalışmalar referans alınarak kullanılmıştır. İlklendirme, cihaz içi testler, uçuş yazılımının yüklenmesi için imaj işlemleri ve yazılım bakım uygulaması gibi adımlar, bu yazılımların yaptığı adımlardan esinlenilerek geliştirme sürecinde uygulanmıştır. Bu durum, özellikle önyükleme yazılımının kritik adımlarının belirlenmesi ve bu adımların doğru ve güvenilir bir şekilde gerçekleştirilmesi açısından önemli bir rol oynamıştır. Her ne kadar kaynak kodlarına erişilememiş olsa da bu konseptler ve işlevsellikler üzerindeki çalışmalar, yazılım geliştirme sürecinde yol gösterici olmuştur.

6.2. Kullanılan Programlama Dilleri ve Araçlar

Önyükleyici yazılımların kodlama süreci, yazılım geliştirme yaşam döngüsünün temel adımlarını içerir. Bu adımlar; gereksinim analizi, tasarım aşamaları ve MKPROM2 gibi açık kaynak kodlardan alınan referanslar doğrultusunda gerçekleştirilmiştir. Kodlama sürecinde, birden fazla programlama dilinden faydalanılmış olup, genellikle C ve Assembly dilleri tercih edilmiştir.

Ek olarak, yazılımın önyükleme sürecinde kullanılacak olan ELF formatındaki ÖYF-1, ÖYF-2 ve UY imajlarının kalıcı belleğe yazılmasını sağlayacak çeşitli yardımcı yazılımlar geliştirilmiştir. Bu yazılımlar, genellikle bu imajları belleğe yerleştirmek ve yönetmek için optimize edilmiştir.

"Eclipse" entegre geliştirme ortamı ile geliştirilmiş, "Scons" derleme aracı ve "Python" betikleri kullanılarak "gcc" derleyicisi derlenmiş bir yazılım ortaya çıkmıştır.

6.3. Yazılımın Bileşenleri: Yöntemler ve Kodlama Stratejileri

Yazılım geliştirme süreci, gereksinimlerin ve tasarım aşamalarının belirlenmesinden sonra, yazılımların kodlanmasıyla devam etmiştir. Bu aşamada yazılımlar, ana birimlere

ayrılmıştır. Kodlama faaliyetleri birim seviyesinde artırımlı ve yinelemeli olarak gerçekleştirilmiştir.

Kodlama faaliyetleri, temelde dört ana kategori altında organize edilmiştir:

- İlklendirme: Bu kategori, donanımın başlangıç sürecini yöneten ve çalışma ortamını hazırlayan işlevlerin kodlanmasını içermektedir. Bu aşama, sistem başlatılması ve gereksinimlerin karşılanması için temel fonksiyonları içerir.

- Cihaz İçi Testler: Donanım ve yazılım bileşenlerinin test edilmesini kapsayan bu aşama, başlangıçta basit seviyede başlar ve zamanla daha kapsamlı ve detaylı testlere doğru genişler. Böylece, sistemin bütünüyle uyumlu ve sağlıklı bir şekilde çalıştığı doğrulanır.

- İmaj İşlemleri: Bu kategori, ÖYF-1, ÖYF-2 ve UY imajlarının kalıcı belleğe yazılması ve bu imajların etkin bir şekilde yönetilmesini içerir. İmaj işlemleri, yazılımın istenen özelliklere uygun olarak kalıcı bellekte yönetilmesini sağlar.

- Yazılım Yaması: Yazılımın bakımı ve güncellemeleri için gereken adımların kodlanması bu kategoriyi oluşturur. Bu aşama, yazılımın kullanım sırasında ortaya çıkabilecek sorunların giderilmesi, hataların düzeltilmesi ve yeni özelliklerin eklenmesini içerir.

Bu dört ana kategori, yazılım geliştirme sürecinin kapsamlı bir şekilde yönetilmesini sağlayarak, yazılımın güvenilir, etkili ve sürekli olarak güncel kalmasını temin eder.

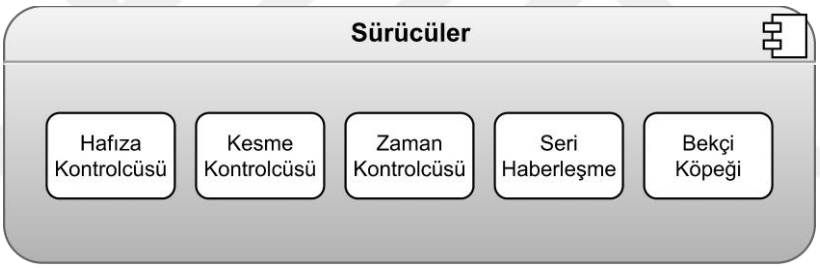
6.3.1. İlklendirme

İlklendirme adımı hem assembly hem de C düzeyinde gerçekleştirilmektedir. Assembly seviyesinde yapılan ilklendirmeler, yazılımsal ve donanımsal kesmeleri almak, yönlendirmek ve C seviyesine geçirebilmek amacıyla gerçekleştirilmektedir. Bu aşamada, bellek zamanlama ayarlamaları, işlemci yazmaçlarının başlangıç değerleri ve önbellek ile

yığın belleklerin ilklendirilmesi gibi işlemler gerçekleştirilir. Böylece, C çalışma zamanından önce temel bir çalışma ortamı hazırlanmış olur.

Assembly seviyesindeki ilklendirmeler, yazılımsal ve donanımsal kesmeler dışında kalıcı olarak gerçekleştirilir. Bu nedenle, ÖYF-2 aşamasında assembly seviyesi ilklendirme, sadece yazılımsal ve donanımsal kesmeleri almak, yönlendirmek ve C seviyesine geçirmek amacıyla gereklidir.

Asgari çalışma ortamı sağlandıktan sonra C düzeyinde gerçekleştirilen ilklendirmeler, ÖYF-1 ve ÖYF-2 yazılımlarının çalışırken kullanacağı sürücülerin başlangıç yapılandırmalarını içerir. Özellikle SCOC3 için özelleştirilmiş olan sürücüler içerisinden, temel işlevleri gerçekleştiren sürücüler tercih edilir. Önyükleme aşamasında kullanılan sürücüler, Şekil 6.1'de gösterilmiştir.



Şekil 6.1. Önyükleyici Yazılımı Sürücüleri

Önyükleyici yazılımın çalışmasını sağlamak ve sistem bileşenlerini düzenlemek için kullanılan kontrol birimleri aşağıda yer almaktadır:

- Hafıza Kontrolcüsü: Hafıza kontrol birimi, yazma, okuma ve zamanlama ayarlamalarını yönetir. Bu birim, bellek işlemlerini kontrol ederek veri akışını düzenler.

- Kesme Kontrolcüsü: Donanımsal ve yazılımsal kesmeler, kesme kontrol birimi aracılığıyla gerçekleştirilir. Bu birim, kesmeleri tanımlar, yönlendirir ve işler.

- Zaman Kontrolcüsü: SCOC3 platformunda üç farklı zamanlayıcı bulunmaktadır. Bu zamanlayıcılar, çeşitli çözünürlüklerde ve periyotlarda bir kere veya düzenli aralıklarla

başlatılabilir. Zaman kontrol birimi, bu zamanlayıcıları yönetir ve ayarlamalarını gerçekleştirir.

- Seri Haberleşme (UART): SCOC3 platformunda üç adet seri haberleşme birimi mevcuttur. Bu birimlerin haberleşme hızları, veri gönderme ve alma işlemleri UART birimi aracılığıyla ayarlanır. Ayrıca, önyükleme sırasında meydana gelen hatalar, çevresel birimlere UART ile raporlanır.

- Bekçi Köpeği (Watchdog): Önyükleme aşamasında kullanılan harici bir birimdir. Periyodik olarak veya giriş pininden tetiklenerek çalışan bir sinyal gönderir. Bu, çevresel birimlere yazılımın aktif olduğunu gösteren bir sinyaldir ve genellikle geniş yelpazeli amaçlar için kullanılır.

6.3.2. Cihaz içi test

Cihaz içi testler, yazılımın uçuş sırasında kullanılacak olan bileşenleri ve sistemleri önceden test etmek için tasarlanmıştır. Bu testler, uçuş yazılımının işlevselliği ve güvenilirliği üzerinde doğrudan etkili olabilecek önemli unsurları kapsar. Bu kapsamda, bütünlük testleri yazılımın bütünlüğünü ve doğruluğunu kontrol etmeyi, hafıza testleri bellek yapılarını ve işlevselliğini incelemeyi, kesme mekanizmasının testi yazılımsal ve donanımsal kesme işlevselliğini doğrulamayı, EDAC mekanizmasının testi ise hataları tespit etmeyi ve düzeltme işlemlerini içermektedir. Bu testler, uçuş sırasında ortaya çıkabilecek potansiyel hataları minimize etmek ve yazılımın güvenilirliğini sağlamak adına kritik öneme sahiptir.

Cihaz içi testler, iki ana gruba ayrılır. İlk olarak, basit testler, yazılımın temel işlevlerini ve genellikle genel sistem performansını değerlendirir. Bu testler, belirli alanlarda hafıza testlerini içerir ve genel işlevselliği kontrol eder. Diğer yandan, detaylı testler daha derin bir analiz sunar ve özel özellikleri veya bileşenleri ayrıntılı olarak inceler. Bu detaylı testler, hafıza testlerine ek olarak kesme mekanizmaları ve hata tespiti ve düzeltme mekanizmalarının testini içerir. Her iki test türü de yazılımın işlevselliği, hatasız çalışması ve güvenilirliği üzerinde büyük etkiye sahiptir.

6.3.2.1. Bütünlük testi

Önyükleme aşamasında, yazılım imajlarının kalıcı belleğe yazılması ve bu imajların daha sonradan okunması sırasında ortaya çıkabilecek bozulmalar, birden fazla kaynaktan meydana gelebilir. Bu durum, ilgili hafıza biriminin doğasından kaynaklanabileceği gibi aynı zamanda yazılım imajlarının yazılma sürecinde veri kaybı veya bozulmasından da kaynaklanabilir. Özellikle uzay ortamındaki radyasyon kaynaklı etkiler nedeniyle hafıza birimlerinde geçici veya kalıcı veri bozulmaları meydana gelebilir.

Bu tür durumlarla karşılaşıldığında, bütünlük testleri, yazılım imajlarının güvenirliğini sağlamak ve hata olasılığını en aza indirmek için kritik önem taşır. İmajlar kalıcı belleğe yazılırken, veri bütünlüğünü korumak amacıyla imajları temsil eden verilere koruma verileri eklenir. Bu koruma verileri, CRC adı verilen bir yöntemle imajın sonuna eklenir ve bu sayede imajın bütünlüğü açılmadan önce kontrol edilir.

Bütünlük testlerinin yapılandırılması, kalıcı bellekteki konfigürasyon adresleri ile gerçekleştirilebilir. Bu testler genellikle hafıza birimlerindeki verilerin bütünlüğünü kontrol eder ve önyükleme aşamasında yazılım imajlarının doğru bir şekilde yüklendiğini doğrulamak için kullanılır. ÖYF-1, ÖYF-2 ve UY imajları için bütünlük testleri yapılabilmektedir. ÖYF-1 yazılımı, kendi imajının yanı sıra ÖYF-2 imajının, ÖYF-2 yazılımı ise seçilen UY imajının bütünlük testini gerçekleştirebilme yeteneğine sahiptir. Yer istasyonundan gönderilen uzkomutlarla hangi imajların bütünlük testinden geçirileceği belirlenebilmektedir.

Bütünlük testlerinin sonuçları, genellikle kalıcı bellekte saklanır ve çevresel birimlere iletilir. Bu sonuçlar aynı zamanda yer istasyonuna uzölçüm paketleri olarak indirilebilir. Bu süreç, önyükleme aşamasında oluşabilecek hataların kaynağını belirlemek ve hatasız imajların bozulmadan yüklendiğini doğrulamak için kullanılır.

6.3.2.2. Hafıza testi

Yazılımlar, hafıza birimlerini kullanarak veri depolama ve işleme faaliyetlerini yürütür. Bu nedenle, yazılımların sağlıklı bir şekilde çalışabilmesi için hafıza birimlerinin

sağlık durumu oldukça kritiktir. Özellikle uzay gibi radyasyona maruz kalan ve uzun süreli kullanım ömrü gerektiren sistemlerde, hafıza birimlerinde oluşabilecek hataların tespiti ve önlenmesi son derece önemlidir. Bu sebeple, önyükleme aşamasında hafıza testleri yapılmaktadır.

Hafıza testleri yazma-okuma-karşılaştırma işlemleri yoluyla gerçekleştirilir. Bu testler, hafıza birimlerindeki tüm alanları kapsayacak şekilde yapılandırılır ve hangi hafıza alanlarının test edileceği genellikle hafızadaki konfigürasyon adresleri üzerinden belirlenir. Yer istasyonundan gönderilen uzkomutlar aracılığıyla test edilecek hafıza alanlarını seçebilir.

ÖYF-1 aşamasında, ÖYF-2 imajının geçici bellekte açılacağı alan önceden test edilir. Benzer şekilde, ÖYF-2 aşamasında da UY imajının geçici bellekte açılacağı alan önceden test edilir. Bu testler program kod alanı testi olarak isimlendirilmiştir.

Önyükleme aşamasının zaman açısından etkin bir şekilde tamamlanabilmesi için, tüm hafıza alanlarının detaylı bir testine zaman ayırmak yerine kritik alanların öncelikli olarak test edilmesi daha uygun olabilir. Bu durumda, programın çalışmasını en çok etkileyecek veya kritik öneme sahip olan program kod alanlarına odaklanmak önerilir. Genel olarak tüm hafıza alanlarının test edilmesi önemlidir, ancak bu süreç uzun zaman alabilir ve önyükleme süresini önemli ölçüde uzatabilir. Bu nedenle, kritik alanların test edilmesi, programın temel işlevselliği için gereklidir ve zaman açısından daha verimli bir yaklaşım olabilir.

ÖYF-2 aşamasında, program kodu dışındaki hafıza testleri gerçekleştirilir. Bu yaklaşımın sebebi, performans hedefleridir. ÖYF-1 yazılımı, kalıcı bellekte çalıştığından dolayı okuma ve yazma hızları geçici bellekte çalışan bir yazılıma göre daha yavaş olabilir. Bu nedenle, performans etkileşimi göz önüne alınarak testlerin ÖYF-2 aşamasında gerçekleştirilmesi tercih edilir.

Hafıza testlerinin sonuçları, genellikle bütünlük testi sonuçları gibi yönetilir. Bu süreç, önyükleme aşamasında ortaya çıkabilecek hataların kaynağını belirlemek ve hatasız olduğu durumlarda hafıza birimlerinin sağlıklı olduğunu doğrulamak için kullanılır.

6.3.2.3.Kesme testi

Kesme testi, SCOC3 için özel olarak tasarlanmış sürücü modüllerinin donanımsal ve yazılımsal kesme ve kotarma fonksiyonlarının önyükleme aşamasında test edilmesini içerir. Bu özellikler, UY içinde kullanılmakta olup önyükleme sırasında test edilir.

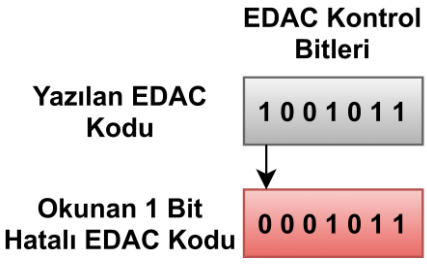
Test, zaman birimi üzerinden kesme oluşturulması ve bu kesmenin kesme kotarıcı fonksiyona yönlendirilmesiyle gerçekleştirilir. Bu süreç, zaman birimi ve kesme kontrol biriminin testini kapsar. Ayrıca, yazılımda yer alan kesme kotarma mekanizmasının doğru çalışıp çalışmadığı da bu test kapsamında incelenir.

Kesme testi yapılandırması, kalıcı bellekteki konfigürasyon adresleri kullanılarak gerçekleştirilebilir. Yer istasyonundan gönderilen uzkomutlar aracılığıyla testin tercihlerini belirleyerek yürütülmesini sağlayabilir.

Kesme testinin sonuçları, bütünlük ve hafıza testleri gibi yönetilir. Bu yöntem, önyükleme aşamasında olası hataların kaynağını tespit edebilirken, kesme ve kotarma mekanizmasının, zaman biriminin ve kesme kontrol biriminin sağlıklı bir şekilde çalışıp çalışmadığını da göstermektedir. Bu süreç, testin bütünlüğünü ve sistemdeki işlevselliği doğrulamak adına kritik bir rol oynamaktadır.

6.3.2.4.EDAC testi

Mevcut sistemde kullanılan SCOC3 ve kalıcı hafıza için özel olarak tasarlanmış olan Hafıza Kontrol Birimi, EDAC mekanizmasını sağlar. Bu mekanizma, 32-bitlik veriyi 8-bitlik sembollerle ifade edebilme özelliğine sahiptir. Bu 8-bitlik semboller EDAC kontrol bitleri olarak adlandırılır ve her yazma işlemi sırasında güncellenir (Bentoutou, 2010). Okuma işlemlerinde ise hafızadaki ve veri yolundaki veriler karşılaştırılarak herhangi bir hata olup olmadığı tespit edilir. EDAC mekanizması, 1 bitlik hataları düzeltebilirken, 1 bit üzerindeki hataları düzeltemez ve bu durumda ilgili hata aldığı adresi yazmaçlar aracılığıyla bildirir (Garg vd., 2016). EDAC bitleri ve bozulma durumu Şekil 6.2'de gösterilmektedir.



Şekil 6.2. EDAC Mekanizması

EDAC testi, bu mekanizmanın hafıza alanında 32 bitlik verideki 1 bit bozulmayı düzeltebilme yeteneğini test eder. Test sürecinde hafıza kontrol biriminin hata ayıklama yazmaçlarından faydalanılır.

EDAC testinin yapılandırılması, kalıcı bellekteki konfigürasyon adresleri kullanılarak gerçekleştirilebilir. Yer istasyonundan gönderilen uzkomutlar aracılığıyla tercihlerini belirleyerek testin yürütülmesini sağlayabilir.

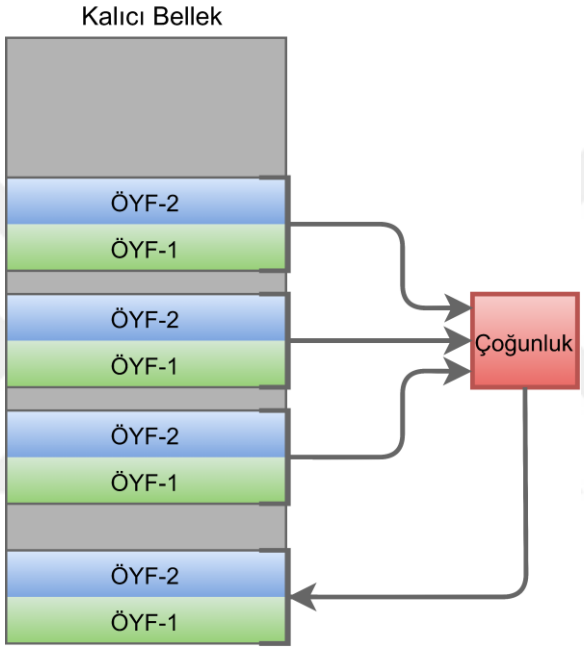
EDAC testinin sonuçları, diğer cihaz içi testlerin sonuçlarına benzer bir yöntemle ele alınır. Bu süreç, önyükleme aşamasındaki hataların kaynağını tespit etme amacıyla kullanılırken, EDAC mekanizmasının kalıcı hafıza üzerinde sorunsuz bir şekilde çalışıp çalışmadığını doğrular. Bu süreç, sistemin güvenilirliğini arttırmada önemli bir rol oynar.

6.3.3. İmaj işlemleri

Önyükleyici yazılımlarının ana işlevi, bir imajı hafızaya açmak ve program akışını yönlendirmektir. Bu süreç için öncelikle önyükleyici yazılımının ve uçuş yazılımının kalıcı hafızaya yazılması gerekmektedir ve bu işlemler yardımcı yazılımlar kullanılarak gerçekleştirilir.

Önyükleyici yazılımlarının kalıcı hafızaya yazılması aşamasında imajlar üzerinde olası bozulmaların tespit edilmesi ve hatta düzeltilmesi için bir yöntem kullanılmaktadır. Bu yöntemde imajlar, üç farklı alana yazılır, bu veriler karşılaştırılır ve gerekli ise düzeltilen veriler başka bir alana yazılarak yazılım bu alandan çalıştırılır. Triple Modular Redundancy yöntemi SCOC3'ün sunduğu şekilde donanımsal olarak gerçekleştirilmekte ve seçilebilmektedir. Şekil 6.3'te önyükleyici yazılım imajlarının SCOC3 TMR mekanizması

kullanılarak kalıcı belleğe yazılması gösterilmiştir. Fakat donanımsal olarak sunulmadığı durumda ise yazılımsal olarak yapılması mümkündür. TMR, modüler yedekliliğin hataya dayanıklı bir biçimidir; burada üç sistem bir işlem gerçekleştirir ve bu sonuç çoğunluk oylama sistemi tarafından işlenir. Sonuçta tek bir çıktı üretilir. Üç sistemden herhangi biri arızalanırsa, diğer iki sistem hatayı düzeltebilir ve maskeleyebilir. TMR konsepti genellikle hataya dayanıklı bilgisayar sistemlerinde bulunur ve bu nedenle uzay sistemlerinde sıkça tercih edilir (Wirthlin vd., 2016).



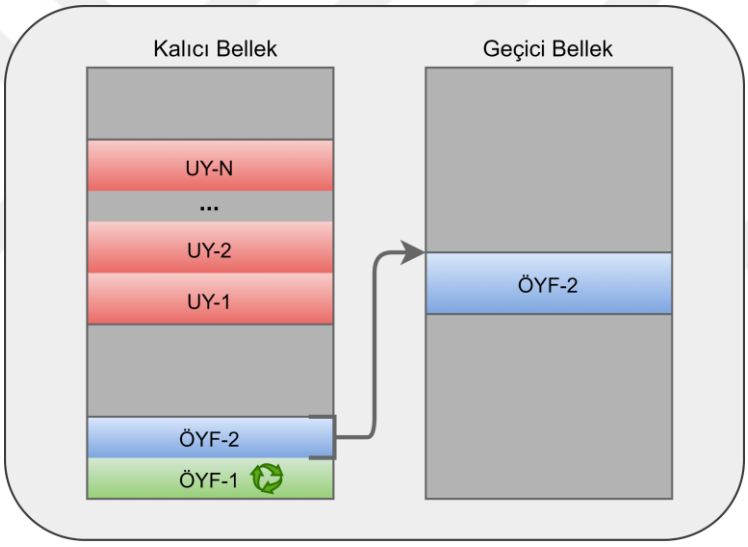
Şekil 6.3. TMR Mekanizması

Önyükleyici yazılım imajlarının bulunduğu alanlarda donanımsal olarak SCOC3'te bulunan TMR kullanılmaktadır. Buna ek olarak konfigürasyon bilgilerinin tutulduğu kalıcı hafıza alanında ise yazılımsal olarak TMR mekanizması kullanılmaktadır. Konfigürasyon alanında önyükleme aşamasında seçim için kullanılan bilgiler, test raporları, UY ilklendirme tabloları ve çevresel birimler ile kullanılan ortak arayüz bilgileri gibi güvenilirlik seviyesi yüksek bilgiler bulunmaktadır. Konfigürasyon bilgilerinin yanlışlıkla değiştirilmesi veya donanımsal kaynaklı hatalardan etkilenmemesi için bilgiler üç alanda tutulur. Bilgiler yazılırken üç alana birden yazılması gerekmektedir ve okuma işlemlerinde de üç alandan okunup karşılaştırılıp düzeltilerek kullanılır. Okuma ve yazma maliyeti artsa da konfigürasyon bilgilerinin doğruluğu kritik önem taşır ve bu nedenle bu yöntem tercih edilir.

İmaj işlemleri kapsamında imaj yükleme aşamaları, imaj seçimi yapılması, Executable and Linkable Format (ELF) imaj yapısı ve son olarak ELF imajının hafıza açılması adımları yer almaktadır.

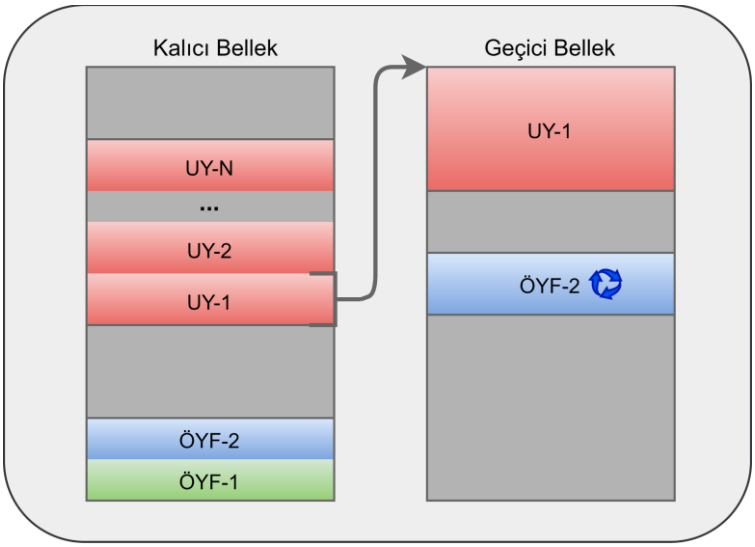
6.3.3.1. İmaj yükleme

İmaj yükleme aşaması, ÖYF-1 yazılımının kalıcı bellekte başlatılmasıyla başlar. Bu aşamada, ÖYF-1 yazılımı, kalıcı bellekte depolanan ÖYF-2 imajını geçici belleğe aktararak işlemi devam ettirir. Geçici belleğe aktarılan ÖYF-2 imajı, ÖYF-1 yazılımı tarafından işletilir. Bu durum, ÖYF-1 yazılımının çalışma sürecinin bir parçasıdır ve bellek durumlarının akışı Şekil 6.4'te görsel olarak temsil edilmiştir.



Şekil 6.4. ÖYF-2 İmajının Yüklenmesi

ÖYF-2 yazılımı ise geçici bellekte çalışmaya başlar. Bu aşamada, ÖYF-2 yazılımı, kalıcı bellekte bulunan ve kullanıcı tarafından seçilen UY imajını geçici belleğe aktararak işlemine devam eder. Daha sonra, geçici bellekteki bu UY imajı, ÖYF-2 yazılımı tarafından işletilir. Bellek durumlarının akışı Şekil 6.5'te görsel olarak temsil edilmiştir.



Şekil 6.5. UY İmajının Seçilmesi ve Yüklenmesi

6.3.3.2. İmaj boyut kontrolü

İmaj boyut kontrolü, önyükleyici yazılımın, hafızaya aktarılacak olan yazılım imajlarının boyutunu değerlendirdiği bir aşamayı ifade etmektedir. Bu kontrol işlemi, ÖYF-2 ve UY imajlarının kalıcı belleğe yazılması sırasında imajların boyut bilgilerinin ayrı bir alana kaydedilmesiyle gerçekleştirilmektedir. Böylece ÖYF-1 çalışırken ÖYF-2 imajı maksimum boyutu, ÖYF-1 çalışırken de UY imajının maksimum boyutu kontrol edilmektedir.

Yazılım, bu boyut kontrol sürecinde imajların boyutunu aşarsa veya beklenen değerlerin dışına çıkarsa hata durumunu algılar. Bu durumda, yazılım ilgili hata durumunu çevre birimlere ileterek ve gerekli bilgilendirmeleri yaparak işlemeyi durdurur. Bu şekilde, imajların boyutlarındaki tutarsızlıklar veya aşma durumları önlenir ve yazılımın güvenilirliği sağlanmış olur.

6.3.3.3. İmaj seçimi

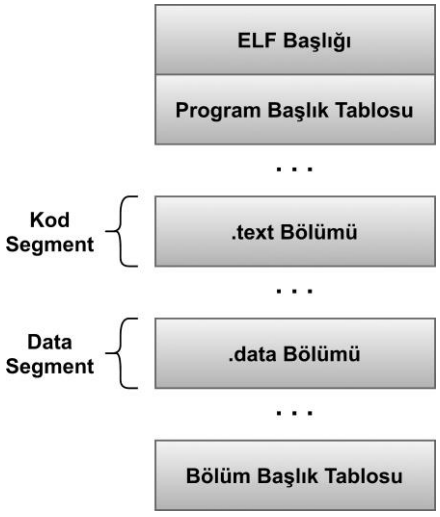
İmaj seçimi, önyükleyici yazılımların, çalıştırılacak olan uçuş yazılımını ve işletim sistemini bağımsız bir şekilde seçebilme yeteneğini içermektedir. Bu durum, önyükleyici yazılımların farklı uçuş yazılımı imajlarını, sistem çalıştırılmadan önce seçebilme özelliğine sahip olmasını sağlar. Uçuş yazılımları önceden belirlenmiş olan hafıza alanlarına

kaydedilmektedir ve hangi uçuş yazılımının çalıştırılacağı bilgisi, hafızadaki konfigürasyon adresleri üzerinden belirlenebilmektedir.

ÖYF-2 çalışırken, belirlenen UY bilgisi doğrultusunda ilgili UY imajı, kalıcı bellekte bulunan alanlardan geçici belleğe açılır. (Bkz. Şekil 6.5) Bu işlem, önyükleyici yazılımın sistemde çalışacak olan belirli bir uçuş yazılımını seçme ve bu yazılımı geçici belleğe aktarma sürecini temsil etmektedir. Bu esneklik, farklı uçuş yazılımlarının kullanılmasına izin verirken, aynı zamanda sistemin daha fazla işlevselliğini ve uyumluluğunu sağlar.

6.3.3.4. ELF imaj yapısı

ELF, komutların ve verilerin bir çalıştırılabilir dosya içerisinde nasıl tutulduğunu tanımlayan format olarak özetlenebilir. ELF dosyaları, ELF başlığı ve veri bölümü olmak üzere iki bölümden oluşmaktadır. ELF başlığı bölümü; ELF imajı hakkında bilgi verirken, veri bölümünün içerisinde; segmentleri tanımlayan bir adet program başlığı tablosu, bölümleri tanımlayan bir adet bölüm başlığı tablosu ve bu bilgilerin devamında da bu iki tablo tarafından atıfta bulunulan segment ve bölümler ile ilgili girdiler yer almaktadır (Clowes, 2019; Tool Interface Standard Committee, 2004; Tool Interface Standard Committee, 1993). Şekil 6.6'da ELF dosya yapısı gösterilmektedir.



Şekil 6.6. ELF Dosya Yapısı

ELF dosya yapısı içerisinde bulunan ana alanlar aşağıda yer almaktadır:

- ELF başlığı: Bu başlık, genellikle ELF dosyasının başında yer alan ve ELF dosyası hakkında bilgiler içeren başlıktır. Yürütülebilir bir programın nasıl yüklenmesi gerektiği ve hangi bölümlerin hangi segmentlerde olduğu gibi bilgileri içerir. Burada hem ELF hakkındaki bazı konfigürasyon bilgileri hem de dosyanın çözümlenmesi sırasında kullanılacak bilgiler tutulmaktadır.

- Program başlığı tablosu: Bu tablo, çalışma zamanında kullanılan segmentleri tanımlayan başlıktır. Programın bellekte nasıl yerleştirileceğini ve segmentlerin hangi konumlarda olduğunu belirtir. Segmentler, çalışma zamanında kullanılan veri ve kod parçalarını tanımlar ve yönetir. Bu bilgiler, çalışma zamanında programın belleğe yüklenmesi ve çalıştırılması için gereklidir.

- Bölüm başlığı tablosu: Bu tablo, dosya içerisindeki tüm bölümlerin yerlerini belirlemek için gerekli bilgileri ve her bir bölüm içerisindeki bilgilerin karakteristiği hakkındaki bilgileri tutmaktadır. Bu bilgiler, dosya içeriğini anlamak ve yönetmek için kullanılır.

ELF dosyalarında sıkça karşılaşılan bölümler aşağıda yer almaktadır:

- .text Bölümü: Programın çalıştırılabilir komutlarının tutulduğu bölümdür.

- .data Bölümü: İlklendirilmiş verilerin tutulduğu bölümdür.

- .bss Bölümü: İlklendirilmemiş verilerin tutulduğu bölümdür. İlklendirilmemiş ya da '0' ile ilklendirilmiş değişkenler burada tutulur.

- .comment Bölümü: Versiyon kontrolü ile ilgili bilgilerin tutulması için kullanılmaktadır.

- .debug Bölümü: Hata ayıklama sembolleri ile ilgili bilgilerin tutulması için kullanılmaktadır.

- .shstrtab Bölümü: Bölüm isimlerinin tutulduğu bölümdür.

ELF dosya yapısı, yazılım geliştirme sürecinde, çalıştırılabilir dosyaların içeriğini organize etmek ve bu dosyaların işlevselliğini sağlamak için önemli bir yapıya sahiptir. Bu yapı, dosyanın içeriğini organize ederken, dosyanın çalışma zamanındaki gereksinimlerini karşılar ve derleyici ile hata ayıklama arasında önemli bir köprü görevi görür. Bu sayede, yazılımın derlenmesi, yüklenmesi ve çalıştırılması süreçleri daha etkin ve güvenilir bir şekilde gerçekleştirilebilir.

6.3.3.5. ELF imajının hafızaya açılması

ELF imajı ile ilgili kontroller ve imajın yüklenmesi adımları aşağıdaki sıralama ile gerçekleştirilmektedir:

1. Başlangıç Adresi Kontrolü: ELF başlığı içerisinde yer alan başlangıç adresi belirtilir ve bu adres, geçici belleğin sınırları içinde mi kontrol edilir. Bu adım, ELF imajının geçici bellekte yüklenebilir olup olmadığını tespit etmek amacıyla gerçekleştirilir. ELF Başlığı içerisindeki bilgiler çekilir ve geçerli bir ELF imajı olup olmadığı kontrol edilir.

2. ELF Geçerliliği Kontrolü: ELF başlığındaki bilgiler çekilir ve geçerli bir ELF imajı olup olmadığı kontrol edilir. Bu aşamada, dosyanın doğru biçimde yapılandırılmış ve uygun biçimde bir ELF imajı olup olmadığı doğrulanır.

3. Program Başlangıç Adresi Düzenlemesi: ELF başlığındaki programın başlangıç adresi, imajın yükleneceği başlangıç adresiyle uyumlu hale getirilir. Bu adım, imajın hangi bellek bölgesine yerleştirileceğini belirlemek için yapılır.

4. Bölüm Yükleme İşlemi: ELF imajı içindeki bölümler, Bölüm Başlığı Tablosunda listelenen sırayla hafızaya yüklenir.

   - Bölümün Başlık Bilgileri: Her bir bölümün başlık bilgisi alınır.

   - Bölümün Yerleştirilmesi: Bölümün yükleneceği yer, imajın yüklenmiş olduğu adres ile bölüm başlığı içindeki ofset değerinin toplamı ile belirlenir.

- Hafızaya Yüklenme: Başlangıç adresi belirlenen her bir bölüm, bölüm başlığındaki adres bilgisinden itibaren bölüm başlığı içinde belirtilen boyut bilgisi kadar hafızaya yüklenir. Bu işlem, her bir bölümün imaj içindeki konumunun belirlenmesi ve ilgili bellek bölgelerine yerleştirilmesi anlamına gelir.

6.3.4. Yazılım yaması

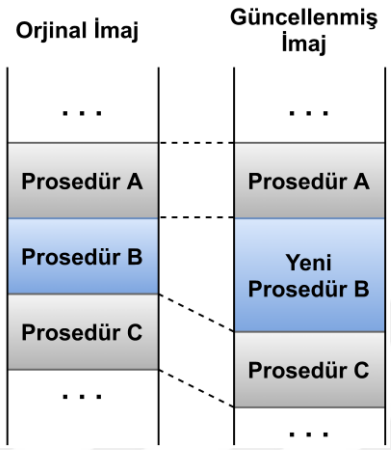
Uydu operasyonlarının kesintisiz bir şekilde devam etmesi için uçuş yazılımındaki hataların düzeltilmesi büyük önem taşımaktadır. Bu durum, uydunun kullanım ömrü boyunca doğru ve stabil bir şekilde çalışmasını sağlamak adına oldukça gereklidir. Uydu, görev süresince çeşitli sebeplerden dolayı uçuş yazılımında değişiklik gerektirebilir. Bu nedenler arasında, donanım sorunlarının giderilmesi, görev esnasında yazılımın performansını arttırmak ve yazılım hatalarını düzeltmek yer alabilir. Tipik bir uydunun ömrü boyunca, yaklaşık 15 kez veya yılda ortalama 2 kez değişiklik yapılması gerekebilir (Calder, ve Kutt, 2009). Yamalama süreci, bir uydu fırlatıldıktan sonra yazılımı değiştirmenin en pratik yoludur ve yazılım bakımı, genel olarak yazılım yaşam döngüsü boyunca dikkatlice planlanmalıdır (López ve Rodríguez, 1996).

Yamalama işlemi genellikle yeni işlevselliklerin eklenmesi veya hataların düzeltilmesi amacıyla gerçekleştirilmektedir. Yazılım yamasının uygulanması, orijinal yazılım imajının yalnızca bir bölümünü değiştirerek yazılımı güncelleme işlemidir. Bu sürecin ilk adımı ise yama verilerinin üretilmesidir. Bu süreçte orijinal ile güncellenmiş yazılım imajı arasındaki fark bulunup yama verileri üretilmektedir. Yama verilerinin verimli bir şekilde üretilmesi için geleneksel ve gelişmiş yöntemler kullanılmaktadır.

6.3.4.1. Geleneksel yöntem

Geleneksel yöntemlerle yazılım yama verilerinin oluşturulması, orijinal ve güncellenmiş imaj arasındaki farklı kod bölümlerinin belirlenmesi ve bu farkların yama verilerine dönüştürülmesi sürecini içerir. Ancak, küçültme teknikleri kullanılmadan bu yöntemle yama verileri oluşturulduğunda, iki imaj arasındaki farklılıklar o kadar büyük olabilir ki yama uygulamak yerine tüm imajın tekrar yüklenmesi daha verimli bir seçenek haline gelebilir.

Yazılımın kaynak koduna yalnızca küçük veya boş bir işlev eklenmesi veya mevcut bir işlevin güncellenmesi durumunda, ELF yapısına sahip imajın ".text" bölümünde yapılan değişiklikler altta yatan işlevleri kaydırır (López ve Rodríguez, 1996). Bu durum, Şekil 6.7'de görsel olarak gösterilmektedir.



Şekil 6.7. Geleneksel Yöntemde Prosedür Değişimlerinin Gösterimi

Ancak, bu kayma sadece ".text" bölümünü etkilemekle kalmaz, aynı zamanda sonraki bölümleri de etkiler. Ayrıca, kaydırılan işlevler için adres işaretçileri değiştiğinden, bu işlevler için yapılan tüm çağrıların güncellenmesi gerekecektir. Bu farklılıkların tümü birleştiğinde, büyük miktarda kod değişimi ortaya çıkacaktır. Bu ölçekteki bir güncelleme, bir yama olarak uygun değildir. Özellikle uydu sistemlerinde iletişim maliyeti ve uygulama süresi dikkate alındığında, uydu uçuş yazılımına yama yapılması için bu yaklaşım uygun olmayabilir. Dolayısıyla, daha gelişmiş ve uygun bir teknik gereklidir (Kutlu vd., 2021).

6.3.4.2. Gelişmiş yöntem

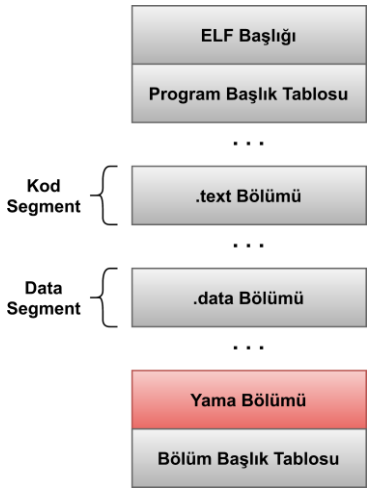
Yazılım yama verilerinin etkin bir şekilde üretilmesini sağlayan teknikler, iki temel noktayı ele almaktadır.

Bunlardan ilki olan ELF, yürütülebilir dosyalar, nesne kodu ve paylaşılan kütüphaneler için ortak bir standart dosya biçimidir. ELF içerindeki “.text “ve “.data” bölümleri yama verilerinin üretilmesinde önemli rol oynamaktadır. “.text” bölümü, bir programın "text" içeriğini veya yürütülebilir talimatlarını içerir. “.data” bölümü, programın bellek görüntüsüne katkıda bulunan başlatılmış verileri tutar. Bağlayıcı, standart bölümlere

ek olarak yeni bölümler oluşturulmasına izin verir ve bu özellik, yama işlemleri için önemlidir (Hofmann, 2019; Tool Interface Standard Committee, 1995; Sanmillan, 2023).

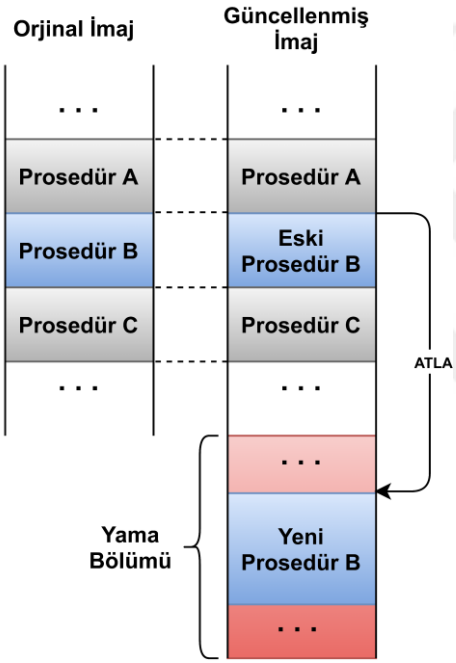
İkinci önemli nokta, GCC'deki güçlü ve zayıf sembollerdir. GCC, geliştiricilerin zayıf ve güçlü semboller tanımlamasına olanak tanır. Varsayılan olarak bir nesne dosyasındaki sembol güçlüdür. Bağlama işlemi sırasında, güçlü sembollerin birden çok kez tanımlanmasına izin verilmez, birden çok güçlü sembol tanımı olduğunda bu, bağlayıcının aynı ada sahip iki sembolü nasıl çözeceğini bilmediği için tanımlama hatalarına neden olur. Ancak yürütülebilir dosyada aynı ada sahip güçlü bir sembol ve birden çok zayıf sembol olabilir. Bu durumda güçlü sembol, aynı isimli zayıf sembolleri geçersiz kılar ve bağlama işlemi sırasında güçlü olan seçilir. GCC'nin bu davranışı, özellikle fonksiyonları tanımlarken kullanışlıdır. Bu davranışın yardımıyla, bir yürütülebilir dosya, önceden tanımlanmış fonksiyonları geçersiz kılabilir (Chamberlain ve Taylor, 1991).

GCC ve ELF formatı tarafından sağlanan işlevselliklerden faydalanmak için, varsayılan GNU bağlayıcısında bazı değişiklikler yapılmalıdır. Bu yeni bağlayıcı, orijinal bölümlerin en altına ELF formatıyla uyumlu bir şekilde içi sıfır değerleri ile dolu yeni bir bölüm oluşturmaktadır. Bu yeni bölüm yama bölümü olarak adlandırılır ve Şekil 6.8'de gösterilmektedir. Bu bölümün amacı, kodda her türlü değişikliği içeren yamaları depolamaktır. Orijinal bölümler değiştirilmeden yeni eklenen veya güncellenen fonksiyonlar yama alanında depolanabilmektedir. Bu nedenle, eklenen değişikliklerin varlığı imajın geri kalanını etkilemediği için, değişiklik en aza indirilir (Kutlu vd., 2021).



Şekil 6.8. Yama Bölümü Eklenmiş ELF Dosya Yapısı

Yama bölümünün eklenmesinden sonra, bu yama alanına erişim sağlanması amaçlanan kod bölümlerinin de güncellenmesi gerekir. Bu nedenle, güncellenen işlevlere yönlendirmeler yapılabilir. Bu, Sidebotham'a göre GCC'nin "güçlü-zayıf semboller" özelliğinin kullanılmasıyla sağlanır. Orijinal fonksiyon sembolü zayıf bir sembol olarak değiştirilir ve yama bölümündeki güncellenmiş fonksiyon güçlü bir sembole sahip olacak şekilde ayarlanır. Böylece, kodun geri kalanında bu işleve yapılan tüm prosedür çağrıları, Şekil 6.9'daki görselde de görüldüğü gibi güncellenmiş işleve yeniden yönlendirilecektir. Bu yöntem, yama verisinin boyutunu en aza indirerek genel yama sürecini optimize etmektedir (Kutlu vd., 2021).



Şekil 6.9. Geliştirilmiş Yöntemde Prosedür Değişimlerinin Gösterimi

6.3.4.3. Geleneksel ve gelişmiş yöntemin karşılaştırılması

Yama verilerinin üretilmesi için kullanılan geleneksel ve gelişmiş yöntemler, oluşturulan yama verilerinin nasıl etkilendiğini göstermek amacıyla karşılaştırılmıştır. Sadece bir değişkenin değerinin değiştirilmesi gibi kodda küçük bir değişiklik yapıldığında, iki yöntemler üretilen yama verileri arasında bir farklılık olmamaktadır. Ancak, kodda daha büyük ve fonksiyonel değişiklikler yapıldığında, geleneksel yöntem ile oluşturulan yama verisinin boyutu, gelişmiş yöntemler oluşturulan yama verisinin boyutundan önemli ölçüde

büyüktür. Büyük boyutlu veriler, yama uygulama süresinde gereksiz bir uzamaya neden olur ve bu da sistemdeki çalışmama süresini etkilemektedir.

Uydu uçuş yazılımda sistem erişilebilirliğini sürdürmek için ilk kısıtlama, sistem çalışmama süresini minimumda tutmaktır. İkinci olarak, büyük boyuttaki bir yama, uyduya gönderilmesi gereken büyük sayıda uzkomut anlamına gelir. Özellikle LEO uydu sistemleri için, uydu ile iletişim kurulabilecek sınırlı bir zaman bulunmaktadır. LEO uyduları, iletişimin mümkün olduğu, iletişim konisine düzenli olarak girip çıkarlar. Bu durum, uçuş yazılımını güncelleme için daha fazla sınırlama getirir. Ayrıca, yamalar için ayrılan bellek miktarı da düşünülmelidir. Çünkü bellek miktarı, maksimum uygulanabilir yama boyutunu belirler, bu da birinci ve ikinci kısıtlamalarla ilişkilidir ve birlikte düşünülebilir (Secretariat, 2013).

Uçuş yazılımı koduna yeni bir fonksiyon eklenerek, farklı iki yöntemle yama verileri oluşturulmuştur. Yöntemler, yama verisinin boyutu, uzkomut sayısı ve uzkomut işleme süresi açısından Çizelge 6.1'de karşılaştırılmaktadır (Kutlu vd., 2021). Orijinal uçuş yazılımı imajının boyutu 1.3 Megabayttır ve tüm yazılım 6133 uzkomutla yüklenebilir. Her bir komutun işlenme süresi 100 milisaniyedir. Bu kısıtlamalar göz önüne alındığında, gelişmiş yöntemin daha verimli olduğu sonucuna varılmaktadır.

Çizelge 6.1. Yama Verisi Üretme Yöntemlerinin Karşılaştırılması

Kullanılan Yöntem	**Yama Verisinin Boyutu**	**Uzkomut Sayısı**	**Uzkomut İşleme Süresi**
Geleneksel Yöntem	760916 bayt	3238	323800 milisaniye
Gelişmiş Yöntem	308 bayt	2	200 milisaniye

6.3.4.4. Yazılım yamasının uygulanması

Yazılım yamasının uygulanabilmesi için ilk olarak kullanılmakta olan ve üzerinde düzeltme yapılması gereken uçuş yazılımı ile hataların veya eksikliklerin giderildiği uçuş yazılımı arasındaki farklılıkların yer aldığı veri alanlarının ve verilerin belirlenmesi gerekir. Farklılığı oluşturan veriler ve bu verilerin yazılması gereken adres bilgileri, aşağıda

belirtilecek olan önyükleyici yazılım yama yapılandırma alanlarına uygun şekilde doldurulmalıdır (Secretariat, 2016).

İkinci aşama önyükleyici çalışırken, uçuş yazılımı imajını geçici bellek alanına açmadan önce bu yapılandırma bilgilerini okur. Eğer yama işleminin gerçekleştirileceğini bildiren yapılandırma alanı doğru şekilde doldurulduysa, uçuş yazılımını geçici hafızaya kopyalar, ardından yamayı uçuş yazılımına uygular ve sonrasında yamalanmış uçuş yazılımını geçici bellek alanına açar. Bu işlem, kalıcı bellekteki uçuş yazılımını değiştirmez. Yani, gerektiğinde yama yapılandırma alanları eski uçuş yazılımına dönmek üzere yapılandırılabilir.

Önyükleyici, önyükleyici yama alanı adı verilen bir alana sahiptir. Bu alan, yama verisi, yama veri boyutu, yama adres ofseti ve yamalanmış imajın CRC değeri gibi bilgileri tutar. Bu alan birden çok yamayı saklayabilir ve aktif yama seçimi yama konfigürasyon adresinde saklanır. Önyükleyici yama alanı ve yama konfigürasyon adresi Şekil 6.10'da verilmiştir.

Şekil 6.10. Önyükleyici Yazılımı Yama Alanı

Yer istasyonundan gönderilen uzkomutlarla önyükleyici yama alanı, yama verileri ile doldurulabilmektedir. Bu değişiklik sonrasında uydu bilgisayarının yeniden başlatılmasından sonra ilk önyükleme aşamasında ilgili yazılım yaması yapılabilmektedir.

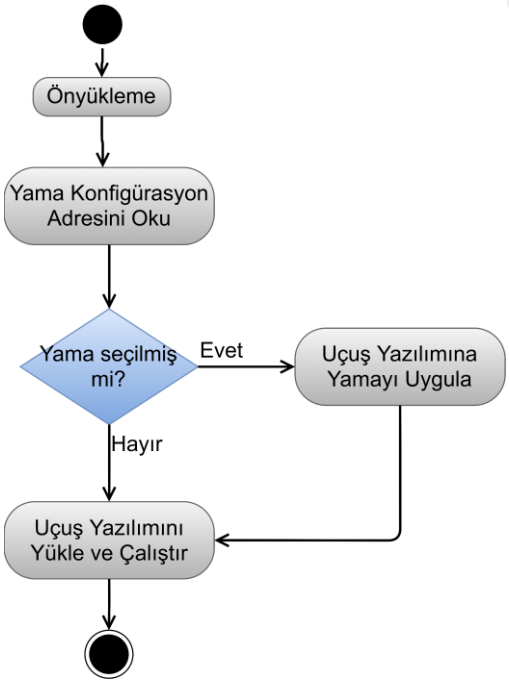
Bu süreç aşağıdaki sırayla gerçekleşir:

1. Her önyükleme işlemi, yama konfigürasyon adresini kontrol ederek kullanılabilir yamaların varlığını belirler. Eğer etkin bir yama seçilmişse, önyükleyici yama işlemini başlatır.

2. Uçuş yazılımı imajı, önyükleyici tarafından değiştirilmek üzere kalıcı bellekten geçici belleğe kopyalanır. Önyükleyici daha sonra seçilen yamayı uygular.

3. Önyükleyici, geçici bellek alanındaki yama uygulanmış uçuş yazılımında boyut ve CRC gibi kontrolleri gerçekleştirir. Başarılı bir şekilde yapılan kontrollerin ardından, yama uygulanmış imaj yüklenir ve güncellenmiş uçuş yazılımı çalıştırılır. Bu yöntemle, orijinal uçuş yazılımı değişmeden kalır. Yeni bir yama talebi için yama verilerinin bu orijinal imaja göre oluşturulması gerekmektedir.

Yama akışı, Şekil 6.11'de gösterilmektedir.



Şekil 6.11. Yama Akışı

Uçuş yazılımı kodunda yapılan değişikliklerle, gelişmiş yöntem kullanılarak yama verileri oluşturulmuştur. Uçuş yazılımı 1.3 Megabayt boyutundadır ve yama verileri ise 36 bayttır. Bu ölçümlerde, yalnızca önyükleyici yazılım aracılığıyla yamanın uygulanma süresi ölçülmüş olup, uzkomut iletim süresi ve imaj bütünlük kontrolü süreleri dikkate alınmamıştır. Elde edilen ölçümler Çizelge 6.2'de sunulmuştur. (Kutlu vd., 2021)

Çizelge 6.2. Önyükleyici Yazılım Yamalama Süresi

Uçuş Yazılımı Boyutu	**Yama Veri Boyutu**	**Yamalama Süresi**
1.3 Megabayt	36 bayt	434 milisaniye

7. ÖNYÜKLEYİCİ YAZILIMIN DOĞRULANMASI VE GEÇERLEMESİ

Önyükleyici yazılımının doğrulama ve geçerleme süreçleri, yazılım geliştirme ve test aşamalarının dikkatle yürütüldüğü aşamalardır. Bu süreçler, yazılımın ayrıntılı incelemesini, birim testlerinin gerçekleştirilmesini, kapalı kutu testlerin uygulanmasını ve sistem seviyesindeki son kullanım testlerini içerir.

Önyükleyici yazılımının doğrulanması ilk olarak, yazılım kodunun içsel kalitesinin gözden geçirilmesini içerir. Bu aşama, yazılım kodlarının yapısal bir incelemesini, olası hata noktalarının tespitini ve yazılım tasarımının uygunluğunun değerlendirilmesini içerir. Birim testleri, önyükleyici yazılımının her bileşeninin ayrı ayrı test edilmesini ve işlevselliğinin kontrol edilmesini sağlar. Bu testler, her bir bileşenin beklenen şekilde çalışıp çalışmadığını belirleyerek yazılımın doğruluğunu değerlendirir. Kapalı kutu testleri ise, önyükleyici yazılımın bütününü görmeden, sadece girdi ve çıktıları kullanarak sistemin beklenen şekilde işlev görmesini doğrular.

Bununla birlikte, önyükleyici yazılımının gerçek donanım üzerinde gerçek dünya koşullarında test edilmesi gereklidir. Bu, donanım seviyesinde gerçekleştirilen sistem testleri ile yapılır. Donanım testleri, yazılımın gerçek donanım ortamında performansını, uyumluluğunu ve güvenilirliğini değerlendirir. Yazılımın amaçlanan işlevlerini yerine getirip getirmediğini, donanım üzerindeki testlerle kontrol eder.

Bu süreçler, önyükleyici yazılımının doğru çalışmasını, güvenilir olmasını ve sorunsuz bir şekilde işlev görmesini sağlamak amacıyla titizlikle yürütülen adımlardır. Bu aşamalar, yazılımın hatalarını ve eksikliklerini minimize etmek, yazılımın güvenilirliğini artırmak ve kullanım ömrü boyunca istikrarlı bir performans sağlamak için önemlidir.

7.1. Gözden Geçirme

Gözden geçirme, potansiyel hataları ve eksiklikleri tespit etmek için önemli bir adımdır. Genellikle dinamik testler yapılmadan önce gerçekleştirilir ve yazılım ürünlerinin farklı aşamalarında gerçekleşebilir. Bu inceleme süreci, yazılım geliştirme yaşam

döngüsünde erken aşamalarda hataların tespit edilmesine yardımcı olur, böylece zamanla oluşabilecek maliyetleri minimize eder (Aydın, 2020).

Yazılımın farklı evrelerinde yapılan gözden geçirmeler, gereksinimlerin, tasarımların, kodun, test planlarının, kullanım kılavuzlarının ve diğer belgelerin ayrıntılı bir incelemesini içerir. Bu inceleme süreci, genellikle hazırlık aşaması, toplantıların düzenlenmesi, toplantılar sırasındaki süreçler, bulguların belirlenmesi ve kayıt altına alınması, düzeltmelerin yapılması ve kontrol edilmesi gibi aşamaları içerir.

Önyükleyici yazılım geliştirme sürecinde, gereksinimler, tasarımlar, kodlar ve birim test belgeleri için gözden geçirme faaliyetleri yapılmıştır. Bu faaliyetler, yazılım ekibi içerisinde teknik incelemeler ve proje ekibiyle resmi olarak yapılan gözden geçirmeleri içerir. Gözden geçirme süreçlerinde, "Crucible" gibi araçlar kullanılarak incelemeler gerçekleştirilmiş ve kayıtlar tutulmuştur (Atlassian, 2023).

Önyükleyici yazılımlarının geliştirilmesi aşamasında gereksinim, tasarım, kod, birim testi ve kapalı kutu testi belgelerinin tamamı için gözden geçirme faaliyetleri yürütülmüştür. Hem teknik olarak yazılım ekibinin kendi arasında, hem de resmi olarak projedeki tüm ekipler ile gözden geçirmeler iki fazda gerçekleştirilmiştir.

Kodlama aşamasındaki gözden geçirmeler için test sonuçlarının, kod kapsama analizinin, statik kod analizinin hazırlanması gerekmektedir. Birim testler tüm alt birimlerinin test edildiğini ve çalışabildiğini göstermektedir. Kod kapsama analizi ile birim testler ile kodların ne kadar kapsandığı tüm durumların ele alınıp alınmadığı görülmektedir. Kapalı kutu testler önyükleyici yazılımının bütünsel olarak test edildiğini ve çalışabildiğini göstermektedir. Statik kod analizinde ise kodların önceden belirlenmiş kodlama standartlarına uydun olup olmadığı kodlama kalitesinin yeterliliği görülmektedir. Bu süreçler, yazılımın sağlamlığı, doğruluğu ve güvenilirliği açısından önemlidir.

7.2. Statik Kod Analizi

Yazılım projelerinde, kullanılan programlama diline ve projenin yapısına bağlı olarak belirli kodlama standartları oluşturulur. Örneğin, Java dilinde yazılan bir yazılım için

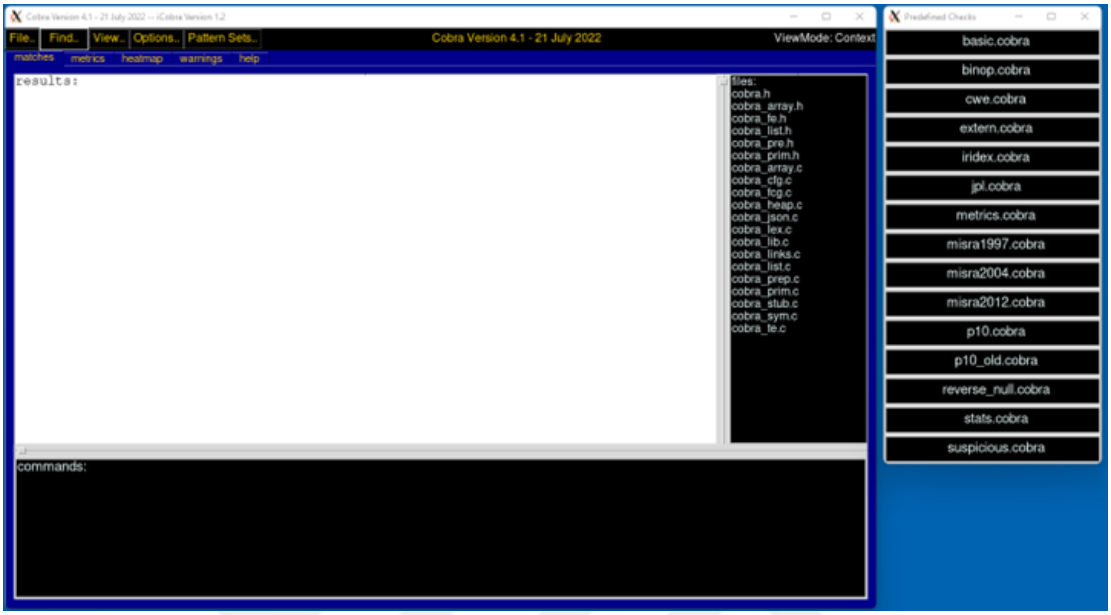
nesne tabanlı programlama kuralları geçerli olabilirken, C dilinde yazılan bir yazılım için gömülü yazılım ve alt seviye yazılım kurallar geçerli olabilir.

Kodlama standartları, genellikle deneyimlerden ve genel standartlardan yola çıkılarak belirlenir. Bu standartlar, farklı kişiler tarafından geliştirilen kodların uyumlu, düzenli, standartlara uygun ve hatadan uzak olmasını sağlar. Statik kod analizi, derleyicilerin derleme esnasında verdiği hataları ve uyarıları içerir. Ayrıca, genel C kodlama standartları olan MISRA 1997, MISRA 2004, MISRA 2012 gibi standartlar ve NASA'nın JPL gibi özel standartları da bulunmaktadır.

Statik kod analizi için, ücretli ve açık kaynaklı birçok araç bulunmaktadır. Bu araçlar genellikle kendi kurallarınızı tanımlama imkânı sunar ve genel kurallara göre analiz yapmanıza olanak sağlar. Statik kod analizi kapsamında "Parasoft" ve "Cobra" araçları üzerinden analizler gerçekleştirilmiştir. (Parasoft, 2023; Holzmann, 2023) Bu araçlar belirlenen kurallara uygunluk analizlerini gerçekleştirmekte ve raporlamaktadır. Tanımlanan kurallar çerçevesinde tüm kurallara uyulmuştur. Şekil 7.1'de "Parasoft" aracının çıktılarına bir örnek gösterilmektedir (Parasoft, 2023).

Şekil 7.1. Parasoft Statik Kod Analiz Aracı Rapor Örneği

"Cobra" aracının kural dizileri ve ekranı Şekil 7.2'de gösterilmektedir. Kural dizileri sol tarafta yer almaktadır (Holzmann, 2023).



Şekil 7.2. Cobra Statik Kod Analiz Aracı Ekranı

7.3. Birim Test ve Kod Kapsama Analizi

Birim test, yazılım birimlerinin en küçük test edilebilir bileşenlerini denetlemek amacıyla yapılan bir test sürecidir. Bu testler, belirli girdiler verildiğinde beklenen çıktıları üretip üretmediklerini kontrol ederek yazılımın küçük parçalarının doğru çalışıp çalışmadığını belirlemeye yöneliktir (Aslan, 2017).

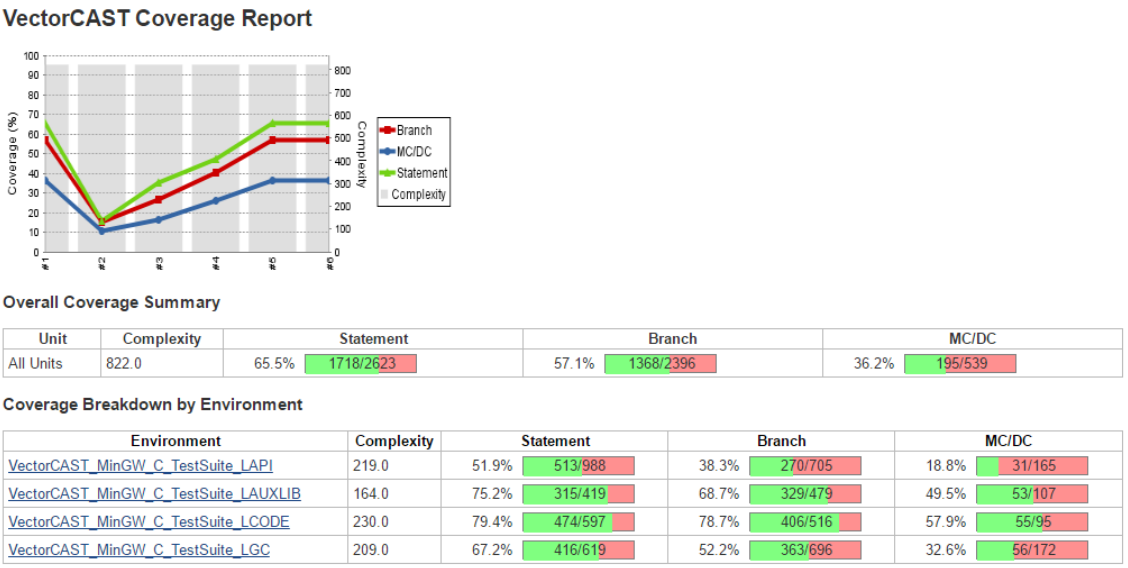
Birim testlerin kullanımı, yazılım geliştirme sürecinde kodda yapılacak değişikliklerin etkilerini hızlı bir şekilde anlamamızı sağlar. Birim testler, kod değişikliklerinin ardından ilgili parçaların beklenen işlevselliğini koruyup korumadığını kolayca değerlendirebilmemizi sağlar.

Bu testler ayrıca belirli zaman aralıklarında çalıştırılarak bize düzenli olarak test raporları sunar. Bu raporlar, kodun belirli bir zamandaki doğruluğunu ve istenilen işlevleri yerine getirip getirmediğini gösterir.

Birim test yazmak, kodlama sürecinde daha düzenli ve standartta uygun kod üretmeye teşvik eder. Çünkü birim testler, belirlenen kodlama standartlarına uygun olacak şekilde kod yazmayı gerektirir ve bu da daha temiz ve bakımı daha kolay kod parçaları oluşturulmasına yardımcı olur.

Önyükleyici yazılımlarının kodlanması sırasında birim testler paralel olarak hazırlanmıştır. Bu testler, önyükleyicinin genel kullanım durumunu test etmemekle birlikte, en küçük parçaların işlevselliğini kontrol etmek için kullanılmıştır. Bu bağlamda, "Unity" adlı açık kaynaklı birim test altyapısı kullanılmış ve testlerin otomatik olarak çalıştırılması ve raporlanması için "Jenkins" aracı kullanılmıştır (Unity, 2023; Jenkins, 2023).

Birim testlerin yanı sıra, kod kapsama analizi de gerçekleştirilmiştir. Bu analizler, yapılan birim testlerinin tüm kod dallanma durumlarını ve satırları kapsayıp kapsamadığını, ölü kodların olup olmadığını ve kodların belirlenen durumlarda doğru şekilde çalışıp çalışmadığını göstermektedir. Donanımsal bağımlılıklar dışında %100 satır ve %100 dal kapsaması elde edilmiştir. Kod kapsama verileri "VectorCAST" aracı kullanılarak elde edilmiştir. Şekil 7.3'te, Jenkins test otomasyon aracında çalıştırılan testler için VectorCAST aracı ile alınan bir kod kapsama raporu örneği sunulmuştur (Jenkins Plugins, 2023).



Şekil 7.3. Kod Kapsama Raporu Örneği

Birim testler ve kod kapsama analizi GitLab Continuous Integration/Continuous Deployment (CI/CD) kapsamında Pipeline üzerinden otomatik olarak koşturulmuştur. Bu süreç, otomatik test koşumu ve kapsama analizinin düzenli olarak yapılarak yazılımın doğruluğunun ve kapsamlılığının takibi açısından önemlidir. Bu süreç, yazılımın sürekli olarak geliştirilmesi ve sürdürülebilirliği için kritik bir adımdır.

7.4. Kapalı Kutu Testi

Önyükleyici yazılımının sağlam ve güvenilir olması, bir sistem başlatıldığında düzgün çalışmasını ve beklenen işlevleri yerine getirmesini gerektirir. Kapalı kutu testi, bu yazılımın kodunun görülmeden, sadece girdi ve çıktılar üzerinden test edildiği bir yöntemdir. Bu çalışmada, önyükleyici yazılımının otomatik test edilmesi amacıyla uzaktan bir test ortamı geliştirilmiştir.

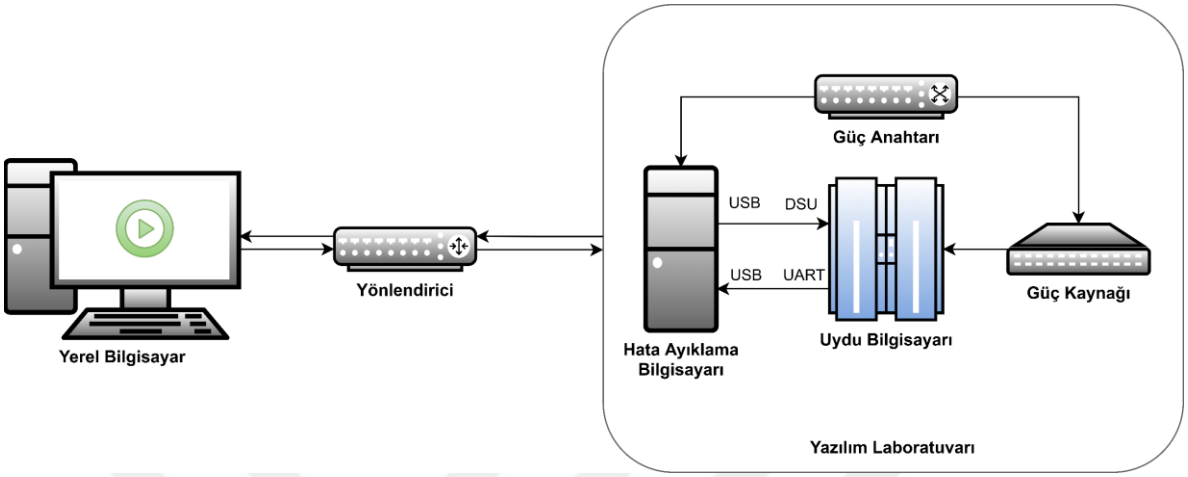
Uzaktan test ortamı, önyükleyici yazılımının uydu bilgisayarına erişimi için gerekli altyapıyı sağlamak üzere bir dizi bileşeni içermektedir. Bu bileşenler, önyükleyici yazılımını uzaktan ve otomatik olarak test etmek için gerekli olan alt yapıyı sağlamak amacıyla çeşitli araçlar ve yöntemler içermektedir.

Bu ortamın ana parçaları arasında, yerel bir bilgisayarın kullanılarak ethernet üzerinden hata ayıklama bilgisayarına erişim ve COM portu aracılığıyla uydu bilgisayarına hata ayıklama modunda erişim sağlanmaktadır. Bu süreçte "GRMON" gibi araçlar kullanılmaktadır.

Ek olarak, uzaktan erişim ile güç anahtarı ve güç kaynağına bağlantılar sağlanmıştır. Bu sayede, uydu bilgisayarının güç kaynağını uzaktan açma ve kapama işlemleri gerçekleştirilebilir hale getirilmiştir. Böylelikle, her başlangıçta önyükleyici yazılımının test edilmesi kolaylıkla yapılabilir.

Önyükleyici yazılımının işlevselliği test edildikten sonra, bu testlerin sonuçları belirli bir formatta UART haberleşme birimi üzerinden iletilir. Bu mesajlar, hata ayıklama bilgisayarına yönlendirilir ve yerel bir bilgisayar üzerinden erişilebilir hale getirilir.

Bu ortamın yapısı, Şekil 7.4'te gösterilmektedir. Bu şema, uzaktan test ortamının bileşenlerini ve ilişkilerini görsel bir biçimde açıklamaktadır.



Şekil 7.4. Test Ortamı Şeması

Önyükleyici yazılımının girdileri, kalıcı bellekte depolanan konfigürasyonları temsil eder. Bu konfigürasyonlar, uzaktan hata ayıklama modu aracılığıyla değiştirilebilir. Ardından, önyükleyici yazılımının başlatılması için güç kaynağı kullanılmaktadır.

Önyükleyici yazılımının çıktıları, yazılımın UART üzerinden gönderdiği mesajları temsil eder. Yazılımın geçerlilik testleri, önyükleyici yazılımının işlevselliğini doğrulamak için yapılmaktadır. Bu testler, yazılımın UART üzerinden gönderdiği mesajları analiz ederek gerçekleştirilir. Test süreci, Python programlama dili kullanılarak geliştirilen test senaryolarını Robot Framework altyapısı üzerine inşa edilmiştir.

Robot Framework, test otomasyonu için açık kaynaklı bir çerçeve olup, kullanıcı dostu bir arayüz sağlar. Bu çerçeve, kullanıcıların yazılım testlerini tasarlamalarını, yürütmelerini ve sonuçlarını analiz etmelerini kolaylaştırır. Özellikle önyükleyici yazılımı gibi karmaşık sistemlerin test edilmesinde, Robot Framework gibi araçlar test süreçlerini düzenlemek ve yönetmek için önemli bir rol oynar.

Bu test yöntemleri, önyükleyici yazılımının beklenen işlevselliğini sağlamak ve istikrarlı bir şekilde çalışmasını güvence altına almak amacıyla özenle tasarlanmıştır.

Özellikle test sürecinin otomatikleştirilmesi ve tekrarlanabilir olması, yazılımın doğru şekilde çalıştığından emin olmak için önemlidir.

Bu testlerin sonucunda, önyükleyici yazılımının doğrulanması ve geçerlemesi sağlanmıştır. Test süreci sonucunda elde edilen veriler, önyükleyici yazılımının beklentilere ne derece uygun olduğunu değerlendirmek için analiz edilmiştir. Bu analiz sonucunda, yazılımın başarılı olduğu alanlar belirlenirken, olası hata noktaları da tespit edilerek geliştirme sürecine katkıda bulunulmuştur.

7.5. Sistem Seviyesi Test

Yazılım geliştirme sürecinin olgunluk aşamasına ulaşıldığında gerçekleştirilen sistem testi seviyesinde, sistemdeki işlevsel ve işlevsel olmayan gereksinimler test edilir. Bu aşamada, genellikle sistemin her yönü test edilir.

Sistem testi seviyesinde, temel test unsurları sistem gereksinimleri, kullanım senaryoları, gereksinimler ve risk analizi raporlarından oluşur. Bu test aşamasında, test edilen nesneler arasında kullanıcı kılavuzları, sistem rehberleri veya sistem yapılandırmaları yer alır.

Sistem testi, test edilen sistemin genel davranışı ile ilgili olduğundan, test planında sistemin test kapsamı net ve anlaşılır bir şekilde belirtilmelidir. Bu seviyedeki testler sırasında dikkat edilmesi gereken önemli bir nokta ise test ortamının sağlanmasıdır. Test ortamı, canlı üretim ortamına mümkün olduğunca benzer olmalıdır; aksi takdirde hata riski artabilir.

Önyükleyici yazılımın sistem seviyesi testinde, gerekli olan tüm canlı ortamlar sağlanmıştır. Sistem gereksinimlerine dayalı olarak işlevsel testler yapılmış ve önyükleme yazılımının beklenildiği gibi çalışıp çalışmadığı gözlemlenmiştir. Ayrıca gereksinimlerde yer alan dış durum senaryoları da test edilerek beklenmeyen durumlarda önyükleyici yazılımının nasıl davrandığı gözlemlenmiştir.

8. BULGULAR VE TARTIŞMA

Bu çalışma, önyükleyici yazılımın tüm geliştirme aşamalarını, yani gereksinimlerin belirlenmesi, tasarım, uygulama, doğrulama ve geçerleme süreçlerini detaylı bir şekilde ele almaktadır. Elde edilen bulgular ve ortaya çıkan sonuçlar aşağıda sistematik bir şekilde değerlendirilmiştir.

Önyükleyici yazılımın başarılı bir şekilde geliştirilebilmesi için, hedeflenen uydu platformu ve donanım özellikleri titizlikle incelenmiştir. Alçak yörüngede konumlanan ve yer gözlemi amacıyla kullanılan uydu sistemlerinin ihtiyaçları belirlenerek, yazılımın çalışacağı uydu bilgisayarları üzerinde detaylı bir analiz gerçekleştirilmiştir. Bu aşama, yazılımın işlevsel ve donanımsal gereksinimlerinin doğru bir şekilde belirlenmesini sağlamıştır.

Önyükleyici yazılımlar genellikle tek aşamalı ve çok aşamalı önyükleme olarak adlandırılan iki temel tasarım yaklaşımını içermektedir. Bu çalışmada, iki aşamalı bir önyükleme yapısının tercih edilmesi, performansın kritik olduğu durumlarda daha verimli bir çalışma sağlamayı amaçlamaktadır. Seçilen tasarımın, önyükleyici yazılımın ihtiyaçlarına uygun bir denge sağladığı ve performans sınırlamalarını azalttığı gözlemlenmiştir.

Belirlenen gereksinimler ve tasarım aşamalarının tamamlanmasının ardından, yazılımın kodlama sürecine geçilmiştir. Bu aşamada, önyükleyici yazılımın temel işlevlerini gerçekleştirmek üzere uygun programlama dilleri, teknikler kullanılarak ve yazılım bileşenlere ayrılarak önyükleme kodları geliştirilmiştir. Ayrıca, önyükleyici yazılımı için test edilebilirlik, bakım kolaylığı ve sistem entegrasyonu gibi kritik faktörler de göz önünde bulundurulmuştur. Yazılım bileşenleri özelinde elde edilen bulgular ve ortaya çıkan sonuçlar aşağıda ele alınmaktadır.

İlklendirme süreci, önyükleyici yazılımın çalışma ortamını hazırlamada kritik bir rol oynamakta olup, bu aşamada gerçekleştirilen ilklendirmeler, yazılımsal ve donanımsal kesmeleri yönlendirerek C düzeyine geçişi başarıyla sağlamaktadır. Kontrol birimleri,

önyükleyici yazılım bileşenlerini düzenleyerek, hafıza kontrolcüsü bellek işlemlerini yönetir, kesme kontrolcüsü kesmeleri işler, zaman kontrolcüsü zamanlayıcıları yönetir, seri haberleşme birimi çevresel birimlerle iletişim kurar ve bekçi köpeği, yazılımın aktif olduğunu belirten bir sinyal gönderir. Bu bağlamda, ilklendirme adımının yazılımın istikrarlı çalışmasını sağlamada ve sürücülerin başlangıç yapılandırmalarını içermesindeki kritik önemi vurgulanmaktadır.

Cihaz içi testler, önyükleyici yazılımın performansını ve güvenilirliğini artırmak adına kritik bir rol üstlenmektedir. Ancak, bu testlerin etkinliğini sürdürebilmek ve uzay uygulamalarındaki çevresel etkenlere uyumunu artırmak için sürekli güncellenmeler ve iyileştirmeler gerekmektedir. Bulgular, bütünlük, hafıza, kesme ve EDAC testlerinin önyükleyici yazılımın sağlıklı bir şekilde çalışmasını güvence altına almak adına dikkatlice uygulanması gerektiğini ortaya koymaktadır.

Önyükleyici yazılımlarının imaj işlemleri, TMR yöntemi kullanılarak güvenilirlik sağlamakta ve imajların kalıcı belleğe yazılmasını içermektedir. İmaj boyut kontrolü, önyükleyici yazılımların önyükleme imajlarının boyutlarını kontrol ederek hataların önlenmesine olanak tanımaktadır. İmaj seçimi, önyükleyici yazılımların bağımsız olarak uçuş yazılımını seçme yeteneğini sunarken, ELF imaj yapısı çalıştırılabilir dosyaların içeriğini düzenleyerek yazılım geliştirme sürecinde önemli bir rol oynamaktadır. ELF imajının hafızaya açılması adımları, başlangıç adresi ve geçerlilik kontrolünden başlayarak programın başlangıç adresinin düzenlenmesi ve bölümlerin hafızaya yüklenmesini içermektedir. Bu süreçlerin başarıyla tamamlanması, önyükleyici yazılımların güvenilir ve etkili bir şekilde çalışmasını sağlamakta, bu da yazılımın sisteme başarılı bir şekilde yüklenip çalıştırılmasına olanak tanımaktadır.

Önyükleyici yazılımının yamalama işlevi, yazılım yamalarının geleneksel ve gelişmiş yöntemlerle uygulanmasının karşılaştırılmasını ele almaktadır. Bulgular, gelişmiş yöntemin daha etkili olduğunu göstermektedir; çünkü bu yöntemle oluşturulan yama verileri daha küçük boyutta ve işleme süresi daha kısadır. ELF formatı ve GCC'nin güçlü-zayıf semboller özelliklerini kullanarak geliştirilen yöntem, iletişim maliyetini minimize ederek uygulama süresini optimize etmektedir. Bu durum, özellikle sınırlı iletişim konisi olan sistemlerde, örneğin LEO uydu sistemlerinde, yazılım güncellemelerinin daha hızlı ve

verimli bir şekilde gerçekleştirilmesine olanak tanıyarak operasyonların güvenilirliğini artırmaktadır.

Önyükleyici yazılımının doğrulama ve geçerleme süreçleri, yazılımın içsel kalitesini ve işlevselliğini değerlendirmek amacıyla titizlikle yürütülmüştür. Bu aşamada, yazılım kodlarının yapısal bir incelemesi, olası hata noktalarının tespiti ve tasarım uygunluğunun değerlendirilmesi gerçekleştirilmiştir. Birim testleri ve kapalı kutu testleri, yazılımın bileşenlerinin ve bütününün beklenen şekilde çalışıp çalışmadığını belirlemiştir. Sistem testleri ise yazılımın gerçek donanım ortamında performansını, uyumluluğunu ve güvenilirliğini değerlendirmiştir.

Sonuç olarak, önyükleyici yazılımın başarıyla geliştirildiği ve tasarımın hedeflenen performans ve esneklik kriterlerini karşıladığı görülmektedir. Doğrulama ve geçerleme süreçleri, yazılımın güvenilir, doğru ve istikrarlı bir şekilde çalışmasını sağlamak adına etkili bir şekilde yürütülmüştür. Bu süreçler, önyükleyici yazılımının uydu sistemlerinde güvenilir bir performans sergilemesine katkı sağlamıştır.

9. SONUÇ VE ÖNERİLER

Bu çalışmada, uzay alanında uydu projelerinde sıklıkla kullanılan LEON3-FT işlemcinin bulunduğu SCOC3 ASIC üzerinde çalışan bir önyükleyici yazılımın gereksinim analizinden başlayarak tasarım, kodlama, doğrulama ve geçerleme süreçleri detaylı bir biçimde ele alınmıştır. Bu çalışma, SPARC mimarisi tabanlı diğer sistemler için de genişletilebilmektedir, böylece, çalışmadaki mühendislik çözümleri referans alınıp uygulanabilmektedir. Mevcut literatür, genellikle uzay görevlerinde kullanılan önyükleyici yazılımları tek bir bakış açısıyla ele almaktadır; ancak bu çalışma, önyükleyici yazılımların kapsamlı bir şekilde nasıl geliştirilebileceğine dair kapsayıcı bir çerçeve sunarak literatüre katkı sağlamayı hedeflemektedir.

Uzay sistemlerindeki kullanılabilirliğin arttırılması, sistemlerin radyasyona karşı dayanıklılığının yanı sıra hataya dayanıklı sistemler ve mekanizmaların tasarlanmasını gerektirmektedir. Bu bağlamda, önyükleyici yazılımlar için EDAC, CRC, TMR gibi yöntemler üzerinde durulmuş ve bu yöntemler cihaz içi testler sırasında kullanılmıştır; imaj bütünlük testi, hafıza testi, kesme mekanizması testi ve EDAC mekanizması testi gibi aşamalarda başarıyla uygulanmıştır.

Uyduların görev sürekliliğinin sağlanması için sistemlerin en kısa sürede tekrar devreye alınması gerekmektedir. Sistemlerin hızla devreye alınması, özellikle alçak yörünge gözlem uydu sistemleri için hayati öneme sahiptir. Bu tür sistemlerde iletişim imkanlarının sınırlı olması, uçuş yazılımının hızlı bir şekilde yüklenmesini zorunlu kılar. Ancak, cihaz içi testler gibi önyükleme süresini artırabilecek işlemler, önyükleyici yazılımların iki aşamada gerçekleştirilmesiyle optimize edilmiştir. Bu tasarım, ilk aşamada temel işlevleri yüksek verimlilikle yerine getirirken, özellikle cihaz içi testler gibi zaman alıcı ve maliyetli işlemlerin ikinci aşamada geçici hafızada yapılmasını sağlayarak önyükleme süresini minimuma indirmektedir.

Uydu görevlerinin kesintisiz bir şekilde yürütülebilmesi için yazılımda meydana gelen hatalar veya eksikliklerin, uydu yörüngede iken, planlanmış görevleri engellemeden düzeltilebilir olması büyük önem taşımaktadır. Yazılımdaki hataların giderilmesi, yazılımın

güncellenmesi ile mümkün olmaktadır. Bu süreçte, yamalama işlemi, yazılımın güncellenmesinde en pratik yol olabilir. Yamalar, uydu uçuş yazılımına yeni bir özellik eklenmesini, mevcut bir özelliğin değiştirilmesini veya yazılımdaki hataların düzeltilmesini sağlayabilir. Uydu uçuş yazılımındaki olası hataların düzeltilmesi için önyükleyici yazılımın güncelleme yeteneği, sistemlerin kesintisiz olarak işlemesini sağlamak açısından hayati bir nitelik taşımaktadır. Bu yaklaşım, mevcut yazılım güncelleme yöntemlerine alternatif bir çözüm sunarak, sistemin operasyonel sürekliliğini artırmayı amaçlamaktadır. Önyükleme aşamasında uçuş yazılımına yapılan yamalar sayesinde, geçici hafızadaki güncellenmiş uçuş yazılımının çalıştırılması, yazılım bakım süreçlerinin önemli bir parçasını oluşturmaktadır. Ayrıca, bu yöntem eski yazılıma geri dönülmesine olanak tanımaktadır.

Önyükleyici yazılım, uçuş yazılımının içeriği ile bağımsız bir şekilde çalışabilmektedir. Bu, uçuş yazılımında bir sorun olması durumunda alternatif bir uçuş yazılımını çalıştırabilme olanağı sunmaktadır. Ayrıca bu özellik, bir uçuş yazılımını kalıcı hale getirmeden önce çalıştırarak test etme amacıyla da kullanılabilmektedir. Aynı şekilde, farklı uçuş yazılımlarını uydu fırlatma aşamaları gibi farklı durumlarda kullanabilmek için kalıcı hafızadaki farklı uçuş yazılımlarını devreye alabilmektedir.

Önyükleyici yazılımın doğrulanma ve geçerleme süreçlerinde, yazılımın kalitesini ve doğrulunu belgeleyen gözden geçirme, statik kod analizi, birim test, kod kapsama analizi, kapalı kutu testi ve sistem seviyesi test adımları gerçekleştirilmektedir. Böylece önyükleyici yazılımları geliştirme süreci tamamlanmış olmaktadır.

Bu çalışmada sunulan önyükleyici yazılım geliştirme adımları, benzer donanımları kullanan diğer sistemler için de referans niteliğindedir. Ayrıca, hata dayanıklı yazılım tasarımı çözümleri, farklı uzay görevleri için de uyarlanabilir niteliktedir.

Gelecekteki çalışmalarda, sanal bir test platformunun tasarlanması, hata sentezlemesi için bu platformun kullanılması, önyükleyici yazılımının bu platform üzerinde test edilmesi, test senaryolarının otomatik olarak üretilmesi, testlerin otomatik olarak çalıştırılması ve raporlanması gibi otomasyonların gerçekleştirilmesi hedeflenmektedir.

KAYNAKLAR DİZİNİ

Aslan, A. (2017). *Unittest (Birim Testi) Nedir?*. Erişim: https://medium.com/@ahmetaslan_78691, Erişim tarihi: 13.12.2023

Aberg, M. *GR712RC Boot SW*. Gaisler. Erişim: https://www.gaisler.com/index.php/products/boot-loaders/gr712rc-bootsw, Erişim tarihi: 13.12.2023

Aberg, M. *GRBOOT - Flight Software Boot Loader*. Gaisler. Erişim: https://www.gaisler.com/index.php/products/boot-loaders/grboot, Erişim tarihi: 13.12.2023

Aberg, M. *Mkprom*. Gaisler. Erişim: https://www.gaisler.com/index.php/products/boot-loaders/mkprom2, Erişim tarihi: 13.12.2023

Atlassian. *Crucible: Collaborative Code Review*. Erişim: https://www.atlassian.com/software/crucible, Erişim tarihi: 13.12.2023

Aydın, N. C. (2020). *Yazilimda Gözden Geçirme (Review)*. Erişim: https://medium.com/@neseceylan.aydin, Erişim tarihi: 13.12.2023

Bayır, İ. (2013). Alçak Yörünge Optik Gözlem Uydu Sistemleri. *Mühendis ve Makina*, 28-31.

Bentoutou, Y. (2010). Program memories error detection and correction on-board earth observation satellites. *World Academy of sciece, Engineering and Technology, 66*. doi.org/10.5281/zenodo.1332742

Calder, A., & Kutt, P. (2009). Flight software design guidelines for enhancing on-orbit maintenance. *AIAA Infotech@Aerospace Conference*. https://doi.org/10.2514/6.2009-1818

Chamberlain, S., & Taylor, I. L. (1991). *The GNU linker*. Version 2.19.51

Clowes, S. (2019). *Fixing/Making Holes in Binaries*. Blackhat. Erişim: https://www.blackhat.com/presentations/bh-asia-02/Clowes/bh-asia-02-clowes.pdf, Erişim tarihi: 13.12.2023

Çoban, H. (2016). Türkiye'nin yer gözlem uydu sistemleri ve ormancılık uygulamalarında kullanılabilirliği. *Turkish Journal of Forestry, 17(1)*, 99-107. https://doi.org/10.18182/tjf.92256

Farcic, V. (2014). *Software Development Models: Iterative And İncremental Development*. Technology Conversations. Erişim: https://technologyconversations.com/2014/01/21/software-development-models-iterative-and-incremental-development/, Erişim tarihi: 13.12.2023

KAYNAKLAR DİZİNİ

Furano, G., Jansen, R., & Menicucci, A. (2013). Review of Radiation Hard Electronics Activities at European Space Agency. *Journal of Instrumentation*, *8*(02). https://doi.org/10.1088/1748-0221/8/02/c02007

Garg, S., Sharma, A. K., & Tyagi, A. K. (2016). An introduction to various error detection and correction schemes used in communication. *International Journal of Applied Research*, *2*(8), 216-218.

GRLIB, I. (2012). GRLIB IP Library User's Manual. *Lattice Semiconductor Corp*.

Hofmann, F. (2019). *Understanding the ELF File Format.* Linuxhint. Erişim: https://linuxhint.com/understanding_elf_file_format, Erişim tarihi: 13.12.2023

Holzmann, G. *Cobra Static Code Analyzer*. Cobra. Erişim: https://spinroot.com/cobra/index.html, Erişim tarihi: 13.12.2023

Jenkins Plugins. *VectorCAST Coverage.* Erişim: https://plugins.jenkins.io/vectorcast-coverage, Erişim tarihi: 13.12.2023

Jenkins. Erişim: https://www.jenkins.io, Erişim tarihi: 13.12.2023

Khalid, O., Rolfes, C., & Ibing, A. (2013). On implementing trusted boot for embedded systems. *2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*. https://doi.org/10.1109/hst.2013.6581569

Kutlu, E., & Gençtürk, C. (2021). Uydu Uçuş yazılımında Gerçek Zamanlı İşletim Sistemleri Zamanlama Algoritmaları. *Türkiye Bilişim Vakfı Bilgisayar Bilimleri ve Mühendisliği Dergisi*, *14*(1), 1–14. https://doi.org/10.54525/tbbmd.830659

Kutlu, E., Gencturk, C., Yigit, R., & Ozcan, F. N. (2021). Patch management of Satellite Flight Software. *2021 15th Turkish National Software Engineering Symposium (UYMS)*. https://doi.org/10.1109/uyms54260.2021.9659638

López, P., & Rodríguez, A. I. (1996). Patching Onboard Ada Software. In *On-board Real-time Software - ISOBRTS.* European Space Agency.

Marangoz, A. M., Sefercik, U. G., & Yüce, D. (2020). Three-dimensional Earth modelling performance analysis of GOKTURK-2 Satellite. *Turkish Journal of Engineering*, *4*(3), 164–168. https://doi.org/10.31127/tuje.650989

Parasoft. *C/C++Test Reporting & Analytics - İmprove Testing.* Erişim: https://www.parasoft.com/products/parasoft-c-ctest/c-c-reporting-analytics/, Erişim tarihi: 13.12.2023

KAYNAKLAR DİZİNİ

Parra, P., da Silva, A., Polo, Ó. R., & Sánchez, S. (2018). Agile deployment and code coverage testing metrics of the boot software on-board solar orbiter's energetic particle detector. *Acta Astronautica, 143,* 203–211. https://doi.org/10.1016/j.actaastro.2017.11.037

Polo, Ó. R., Sánchez, J., da Silva, A., Parra, P., Hellín, A. M., Carrasco, A., & Sánchez, S. (2021). Reliability-oriented design of on-board satellite boot software against Single Event Effects. *Journal of Systems Architecture, 114,* 101920. https://doi.org/10.1016/j.sysarc.2020.101920

Sanmillan, I. (2023). *Executable and Linkable Format 101 - Part 1 Sections and Segments*. Intezer. Erişim: https://www.intezer.com/blog/research/executable-linkable-format-101-part1-sections-segments/, Erişim tarihi: 13.12.2023

Secretariat, E. C. S. S. (2013). Software Engineering Handbook. *ECSS-E-HB-40A, ECSS Secretariat, Tech. Rep.*

Secretariat, E. C. S. S. (2016). Telemetry and Telecommand Packet Utilization. *ECSS-E-ST-41C, ECSS Secretariat, Tech. Rep.*

Sidebotham, B. *Weak Function Attributes.* Valvers. Erişim: https://www.valvers.com/open-software/gcc/weak-function-attributes, Erişim tarihi: 13.12.2023

Tool Interface Standard Committee. (1993). *Tool Interface Standard (TIS) Portable Formats Specification.* Version 1.1.

Tool Interface Standard Committee. (1995). *Tool Interface Standard (TIS) Executable and Linking Format (ELF) Specification*. Version 1.2.

Tool Interface Standard Committee. (2004). *Executable and Linkable Format (ELF).* Version 1.1.

TUSAŞ. *Göktürk-1.* Erişim: https://www.tusas.com/urunler/uzay/yer-gozlem-ve-kesif-uydulari/gokturk-1 , Erişim tarihi: 13.12.2023

TUSAŞ. *Göktürk-2.* Erişim: https://www.tusas.com/urunler/uzay/yer-gozlem-ve-kesif-uydulari/gokturk-2 , Erişim tarihi: 13.12.2023

TUSAŞ. *Göktürk-3.* Erişim: https://www.tusas.com/urunler/uzay/yer-gozlem-ve-kesif-uydulari/gokturk-3 , Erişim tarihi: 13.12.2023

TUSAŞ. *Göktürk-Y.* Erişim: https://www.tusas.com/urunler/uzay/modernizasyon-programlari/gokturk-y , Erişim tarihi: 13.12.2023

KAYNAKLAR DİZİNİ

TÜBİTAK UZAY, *İMECE*, Erişim: https://uzay.tubitak.gov.tr/tr/uydu-uzay/imece, Erişim tarihi: 13.12.2023

TÜBİTAK UZAY, *RASAT*, Erişim: https://uzay.tubitak.gov.tr/tr/urunlerimiz/rasat, Erişim tarihi: 13.12.2023

Unity. *Unit Testıng For C (Especıally Embedded Software)*. Throw The Switch. Erişim: http://www.throwtheswitch.org/unity, Erişim tarihi: 13.12.2023

Wikimedia Foundation. (2023) *Iterative and Incremental Development.* Wikipedia Erişim: https://en.wikipedia.org/wiki/Iterative_and_incremental_development, Erişim tarihi: 13.12.2023

Williams, M. (2017). *What is low Earth orbit?* Universe Today. Erişim: https://www.universetoday.com/85322/what-is-low-earth-orbit/, Erişim tarihi: 13.12.2023

Wirthlin, M. J., Keller, A. M., McCloskey, C., Ridd, P., Lee, D., & Draper, J. (2016). SEU mitigation and validation of the LEON3 soft processor using triple modular redundancy for space processing. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (pp. 205-214). https://doi.org/10.1145/2847263.2847278

Xu, W., & Piao, Y. (2010). Bootstrap loader design of aerospace payload controller based on TSC695F. *2010 Second International Conference on Computational Intelligence and Natural Computing*. https://doi.org/10.1109/cinc.2010.5643789