T.C.
İSTANBUL ÜNİVERSİTESİ-CERRAHPAŞA
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

YÜKSEK LİSANS TEZİ

FLASH BELLEK DEPOLAMA SİSTEMLERİNDE DAYANIKLILIĞIN ARTTIRILMASI İÇİN VERİMLİ ÖNYÜKLEYİCİ TASARIMI VE UYGULANMASI

Mehmet Uğur KEBİR

Elektrik-Elektronik Mühendisliği Anabilim Dalı

Elektrik-Elektronik Mühendisliği Programı

DANIŞMAN
Prof. Dr Fırat KAÇAR

Haziran, 2022

İSTANBUL

Bu çalışma 07.06.2022 tarihinde aşağıdaki jüri tarafından Elektrik-Elektronik Mühendisliği Anabilim Dalı Elektrik-Elektronik Mühendisliği Programında Yüksek Lisans Tezi olarak kabul edilmiştir.

Tez Jürisi

Prof. Dr Fırat KAÇAR (Danışman)
İSTANBUL ÜNİVERSİTESİ-CERRAHPAŞA
MÜHENDİSLİK FAKÜLTESİ

Doç. Dr. Cengiz Polat UZUNOĞLU
İSTANBUL
ÜNİVERSİTESİ-CERRAHPAŞA
MÜHENDİSLİK FAKÜLTESİ

Doç. Dr. Yunus BABACAN
ERZİNCAN BİNALİ YILDIRIM
ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ

İÇİNDEKİLER

Sayfa No

ÖNSÖZ ..... **iv**
İÇİNDEKİLER ..... **vii**
ŞEKİL LİSTESİ ..... **ix**
TABLO LİSTESİ ..... **x**
SİMGE VE KISALTMA LİSTESİ ..... **xii**
ÖZET ..... **xiii**
SUMMARY ..... **xiv**
1. GİRİŞ **1**
2. GENEL KISIMLAR **3**
2.1. MİKRODENETLEYİCİLER VE GÖMÜLÜ SİSTEMLER 3
2.2. FLASH BELLEK 3
2.2.1. Flash Bellek Tutma Süresi 4
2.2.2. Flash Bellek Dayanıklılığı 4
2.2.3. Flash Belleklerde Yazma ve Silme Operasyonu 4
2.3. ÖNYÜKLEYİCİ 6
2.3.1. Klasik Önyükleyici Tasarımları 6
3. MALZEME VE YÖNTEM **8**
3.1. İMPLEMENTASYON İÇİN MİKRODENETLEYİCİ SEÇİMİ 8
3.2. GÖMÜLÜ SİSTEM TEST DÜZENEĞİ 8
3.2.1. Sisteme Genel Bakış 8
3.2.2. Geliştirme Ortamı 9
3.3. WEAR LEVELİNG ALGORİTMALARI 11
3.3.1. Dinamik Wear Leveling 11
3.3.2. Statik Wear Leveling 12
3.3.3. Verimli Önyükleyici Tasarımına Özel Wear Leveling Algoritması 12
3.3.4. No Wear Leveling Algoritması 13

3.4. VERİMLİ ÖNYÜKLEYİCİ YAZILIM TASARIMI 14
3.4.1. Verimli Önyükleyici Yazılım İzolasyonu 14
3.4.2. Programlanabilir Sanal Bellek Blokları 16
3.4.3. Sanal Bellek Blok Bilgilerinin İzolasyonu 16
3.5. VERİMLİ ÖNYÜKLEYİCİ YAZILIM İMPLEMENTASYONU 18
3.5.1. Önyükleyici Komut Satırı Arayüzü(CLI) Uygulaması 18
3.5.1.1. CLI Önyükleyici Menüsü 19
3.5.1.2. CLI Önyükleyici Yazılım Versiyon Komutu 20
3.5.1.3. CLI Önyükleyici Yardım Komutu 20
3.5.1.4. CLI CID Komutu 21
3.5.1.5. CLI RDP Durum Komutu 21
3.5.1.6. CLI Flash Adresine Atlama Komutu 22
3.5.1.7. CLI FLash Silme Komutu 22
3.5.1.8. CLI Flash Yazma Komutu 24
3.5.1.9. CLI Flash Sektör Okuma/Yazma Erişim Aktif Etme Komutu 26
3.5.1.10.CLI Flash Sektör Okuma/Yazma Erişim Pasif Etme Komutu 28
3.5.1.11.CLI Flash Sektör Okuma/Yazma Erişim Durum Komutu 28
3.5.1.12.CLI OTP Okuma Komutu 31
3.5.1.13.CLI Flash Adres Okuma Komutu 32
3.5.1.14.CLI Sanal Bellek Blok Bilgilerini Okuma Komutu 33
3.5.1.15.CLI Yürütülebilir Dosya Hazırlık Komutu 35
3.5.2. Kullanıcı Uygulama Projesinden Yürütülebilir Dosya Üretimi 38
3.5.3. Bellek Dayanıklılığını Arttırmak İçin Optimum Yazma Adresinin Belirlenmesi 40
3.5.4. Sanal Bellek Blok Bilgilerinin Saklanması 41
4. BULGULAR 43
4.1. VERİMLİ ÖNYÜKLEYİCİ PERFORMANS ANALİZİ 43
4.1.1. Verimli Önyükleyici Yazılımının Bulunduğu Sektörlerin Performansı 43
4.1.2. Programlanabilir Sanal Bellek Bloklarının Bulunduğu Sektörlerin Performansı 43
4.1.3. Sanal Bellek Blok Bilgilerinin Bulunduğu Sektörlerin Performansı 50
4.1.4. Verimli Önyükleyici İle Flash Bellek Bozulmasını Tahmin Etme 52
5. TARTIŞMA VE SONUÇ 54
KAYNAKLAR 56

ÖZGEÇMİŞ 59

ŞEKİL LİSTESİ

Page No

Şekil 2.1: Klasik Önyükleyici Tasarımlarının Genel Akış Diagramı 7

Şekil 3.1: Deney Düzeneğinin Yapısı..... 10

Şekil 3.2: Verimli Önyükleyici Tasarımı..... 15

Şekil 3.3: CLI Önyükleyici Menüsü 19

Şekil 3.4: CLI Geçersiz Komut Cevabı 20

Şekil 3.5: CLI Yazılım Versiyon Komut Cevabı..... 20

Şekil 3.6: CLI Yazılım Versiyon Komut Cevabı..... 20

Şekil 3.7: CLI CID Komut Cevabı 21

Şekil 3.8: CLI RDP Durum Komut Cevabı 21

Şekil 3.9: CLI RDP Flash Adresine Atlama Komutu Cevabı..... 22

Şekil 3.10: Önyükleyici RDP Flash Adresine Atlama Komutu Cevabı 22

Şekil 3.11: CLI Flash Silme Komut Cevabı 22

Şekil 3.12: Önyükleyici Flash Silme Komut Cevabı..... 23

Şekil 3.13: STM32CubeProgrammer ile Silinen Sektörün Okunması..... 23

Şekil 3.14: Yürütülebilir Dosya Olmadığı Zaman CLI Flash Yazma Komut Cevabı..... 24

Şekil 3.15: Yürütülebilir Dosya Hazır Olduğu Zaman CLI Flash Yazma Komut Cevabı..... 24

Şekil 3.16: Flash Yazma İşlemi Başlarken Önyükleyici Cevabı..... 25

Şekil 3.17: Flash Yazma İşlemi Tamamlandıktan Sonra CLI Cevabı 26

Şekil 3.18: Flash Yazma İşlemi Tamamlandıktan Sonra Önyükleyici Cevabı..... 26

Şekil 3.19: CLI Flash Sektör Okuma/Yazma Erişim Aktif Etme Komut Cevabı 27

Şekil 3.20: Önyükleyici Flash Sektör Okuma/Yazma Erişim Aktif Etme Komut Cevabı..... 27

Şekil 3.21: CLI Flash Sektör Okuma/Yazma Erişim Pasif Etme Komut Cevabı 28

Şekil 3.22: Önyükleyici Flash Sektör Okuma/Yazma Erişim Pasif Etme Komut Cevabı 28

Şekil 3.23: CLI Flash Sektör Okuma/Yazma Erişim Durum Sorgu Komut Cevabı - 1 29

Şekil 3.24: Önyükleyici Sektör Okuma/Yazma Erişim Durum Sorgu Komut Cevabı..... 30

Şekil 3.25: CLI Flash Sektör Okuma/Yazma Erişim Durum Sorgu Komut Cevabı - 2 30
Şekil 3.26: CLI Örnek OTP Sorgu Komut Cevabı..... 31
Şekil 3.27: Önyükleyici Örnek OTP Sorgu Komut Cevabı 31
Şekil 3.28: CLI Flash Bellek Adres Okuma Sorgu Komut Cevabı 32
Şekil 3.29: Önyükleyici Flash Bellek Adres Okuma Sorgu Komut Cevabı 32
Şekil 3.30: CLI Sanal Bellek Blok Bilgilerini Okuma Sorgu Cevabı - 1 33
Şekil 3.31: CLI Sanal Bellek Blok Bilgilerini Okuma Sorgu Cevabı - 2 33
Şekil 3.32: CLI Sanal Bellek Blok Bilgilerini Okuma Sorgu Cevabı - 3 34
Şekil 3.33: CLI Sanal Bellek Blok Bilgilerini Okuma Sorgu Cevabı - 4 34
Şekil 3.34: CLI Sanal Bellek Blok Bilgilerini Okuma Sorgu Cevabı - 5 35
Şekil 3.35: Önyükleyici Sanal Bellek Blok Bilgilerini Okuma Sorgu Cevabı..... 35
Şekil 3.36: CLI Yürütülebilir Dosya Hazırlık Komutu Cevabı - 1..... 36
Şekil 3.37: CLI Yürütülebilir Dosya Hazırlık Komutu Cevabı - 2..... 37
Şekil 3.38: Önyükleyici Yürütülebilir Dosya Hazırlık Komutu Cevabı 37
Şekil 3.39: İki Sayfalı Sayfa Değiştirme Şeması[1]..... 42

TABLO LİSTESİ

Page No

Tablo 4.1: Farklı Uygulama Boyutlarına Göre STM32F407VGT6 Flash Bellek Dayanıklılık Verimliliği 45

Tablo 4.2: STM32F407VGT6 Flash Bellek Sektör Silinme Süreleri[2] 46

Tablo 4.3: Farklı Boyutlardaki Uygulamalar İçin Flash Bellek Yazılma Süreleri 47

Tablo 4.4: Farklı Boyutlardaki Uygulamalar İçin Verimli ve Klasik Önyükleyici Flash Belleklerine Yazılma Süresi Oranları 48

Tablo 4.5: Farklı Boyutlarda Kullanıcı Uygulamaları İçin Verimli Ve Klasik Önyükleyici Yazma Süresi Verimliliği 50

Tablo 4.6: STM32F407VGT6 Mikrodenetleyicisinin EEPROM Emülasyon Flash Bellek Dayanıklılık Verimliliği 51

SİMGE VE KISALTMA LİSTESİ

Simgeler **Açıklama**

KB : kilobyte
MB : megabyte
ms : millisecond
us : microsecond

Kısaltmalar **Açıklama**

BDM : Background Debug Mode interface
CAN : Controller Area Network
CLI : Comman Line Interface
CPU : Central Process Unit
CRC : Cyclic Redundancy Check
DAO : Dayanıklılık Artış Oranı
DWWT : Double Word Writing Time
IDE : Integrated Development Environment
ISP : In-system programming
I2C : Inter-Integrated Circuit
I/O : Input/Output
JTAG : Joint Test Action Group
KÖ : Klasik Önyükleyici
MCU : Microcontroller Unit
OTP : One Time Programmable
PK : Programlama Kapasitesi
P/E : Program/Erase
RAM : Read Access Memory
SM : Silme Miktarı
SPI : Serial Peripheral Interface
SS : Silme Süresi
SWD : Serial Wire Debug
TSS : Toplam Silme Süresi

TYS : Toplam Yazma Süresi

USART : Universal Synchronous and Asynchronous Receiver-Transmitter

USB : Universal Serial Bus

VÖ : Verimli Önyükleyici

YB : Yetersiz Bellek

YS : Yazma Süresi

YSV : Yazma Süresi Verimliliği

ÖZET

YÜKSEK LİSANS TEZİ

FLASH BELLEK DEPOLAMA SİSTEMLERİNDE DAYANIKLILIĞIN ARTTIRILMASI İÇİN VERİMLİ ÖNYÜKLEYİCİ TASARIMI VE UYGULANMASI

Mehmet Uğur KEBİR

İstanbul Üniversitesi-Cerrahpaşa

Lisansüstü Eğitim Enstitüsü

Elektrik-Elektronik Mühendisliği Anabilim Dalı

Danışman: Prof. Dr Fırat KAÇAR

Nihai hale gelen endüstriyel ürünler çok fazla güncellenmese de geliştirme ve test safhasında çok sık programlama işleminin yapılması flash bellek dayanıklılığını verimsiz bir şekilde azalmaktadır. Özellikle pandemi sonrası yarı iletken teknolojisi kullanan ürünlerin üretim/tedarik süreçlerinde yaşanan sorunlar ve termin sürelerinin çok uzun olması, geliştirme ve test safhasında kullanılan ürünlerin de ömürlerinin uzatılması için yaptığımız çalışmaya ilham kaynağı olmuştur.

Çalışma kapsamında mikrodenetleyicilerin kullanım ömrünü etkileyen flash belleğin dayanıklılığının artırılmasına ilişkin bir önyükleyici tasarımı ve implementasyonu anlatılmıştır.

Yeni bir uygulama yükleme talebi geldiğinde herhangi bir ek donanıma ihtiyaç duymadan aşağıdakilerin sağlanması amaçlanır; flash belleğe yazılacak verilerin sektörler arasında homojen bir şekilde paylaşılmasını sağlamak, gereksiz bellek silinmesini önlemek, uygulama yüklemek için harcanan sürelerin kısaltılması, bellek dayanıklılığını arttırmak ve olası bellek dayanıklılık problemini önceden tahmin edebilmektir.

Haziran 2022, 58 sayfa.

Anahtar kelimeler: Flash Bellek, Bellek Dayanıklılığı, Önyükleyici, Gömülü Sistem.

SUMMARY

M.Sc. THESIS

DESIGN AND IMPLEMENTATION OF EFFICIENT BOOTLOADER FOR ENDURANCE ENHANCEMENT IN FLASH MEMORY STORAGE SYSTEMS

Mehmet Uğur KEBİR

İstanbul University-Cerrahpaşa

Institute of Graduate Studies in Sciences

Department of Electrical-Electronics Engineering

Supervisor: Prof. Dr. Fırat KAÇAR

Although the final industrial products are not updated much, frequent programming during the development and testing phase inefficiently reduces the endurance of flash memory. Especially after the pandemic, the problems experienced in the production/supply processes of products using semiconductor technology and the very long deadlines have inspired our work to extend the life of the products used in the development and testing phase.

Within the scope of the study, a bootloader design and implementation related to increasing the durability of flash memory, which affects the lifetime of microcontrollers, are explained.

When a new application installation request comes, it is aimed to provide the following without the need for any additional hardware; It is to ensure that the data to be written to the flash memory is shared homogeneously between sectors, to prevent unnecessary memory deletion, to shorten the time spent on installing applications, to increase memory endurance and to predict possible memory endurance problems.

Jun 2022, 58 pages.

Keywords: Flash Memory, Endurance, Bootloader, Embedded Systems.

1. GİRİŞ

Gömülü sistemler için geliştirilen uygulamaların hedef sisteme yüklenebilmesi, test edilebilmesi, oluşan hataların giderilmesi, çalışma akışının gözlemlenmesi ve gerektiğinde müdahale edilebilmesi için kullanılan araçlara debugger denilmektedir. [3] Birçok modern mikroişlemci tasarım aşamasında debugger araçlarıyla uyumlu olacak şekilde tasarlanır.[4] Nihai ürün haline getirilen geliştirme kitlerinde veya mikrodenetleyicilerde JTAG, SWD, ISP, BDM gibi debug portları, debugger araçlarının çalışabilmesi için tasarlanan çip düzeyinde bağlantı arabirimleridir. Debug portları, emniyet kritik projelerde sistem doğrulaması sağlamak veya tasarım değişikliğinden kaynaklanacak yüksek maaliyetlerden kaçınılan ürünler gibi spesifik amaçlar dışında genellikle son kullanıcı ürünlerinde güvenlik zaafiyeti oluşturmamak için gizlenmekte veya devre dışı bırakılmaktadır.[5] Debug portlarının son kullanıcı ürünlerinde ekseriyetle kullanılmaması, debugger araçlarının yüksek maaliyetli olması, elektronik ürün yazılımlarının programlama ve güncelleme ihtiyaçlarından dolayı önyükleyici yazılımları oldukça yaygın olarak tercih edilmektedir.

Önyükleyici yazılımlarının birincil amacı, harici programlayıcı donanımına ihtiyaç duymadan yüklenecek uygulama kodunun belleğe yazılması için donanım cihazlarını başlatmaktır.[6] Flash bellekler, gömülü sistemlerde depolama, kullanıcı uygulama programının ve önyükleyici yazılımlarının gömülmesi için en iyi seçenekler arasında yer almaktadır.[7] Geliştiriciler, test ve deneysel işlemler gibi bazı durumlarda hızlı program yükleme amacıyla RAM bellekleri tercih etseler de, flash bellek alanları genellikle uygulamaları kalıcı hale getirmek için kullanılmaktadır. Flash bellekler, elektrikle silinip programlanabildiği için, bellekte tutma süresi ve bellek dayanıklılığı konularında kısıtlamalar söz konusudur.[8]

Mikrodenetleyicilerdeki flash belleklerin tutma süresi ve bellek dayanıklılığı ile ilgili bilgiler üretici firmalar tarafından sağlanmaktadır.[2] Flash bellek sektörlerinde gerçekleştirilen P/E döngülerinin sayısı ile sektörde programlanan verileri güvenilir bir şekilde tutacağı süre arasında da ters bir ilişki vardır.[9] Üretici firmalar flash belleklerde tipik olarak veri saklama süresini flash bellek sektöründe silme işleminin olmadığını varsayılarak belirtilir. Bir flash bellek sektöründe P/E işlemi arttıkça, verilerin güvenilir bir şekilde tutulacağı süre azalır.[10]

Genellikle bu, sistem çalışması için bir soruna neden olmaz çünkü sıklıkla silinen bir sektör, herhangi bir veri kaybı tehlikesi olmadan çok önce tekrar silinebilir. Bununla birlikte, wear leveling algoritmaları, mantıksal sektörlerdeki ortalama P/E döngü sayısını azaltır ve böylece bu sektörlerdeki veri saklama süresini, tüm P/E döngülerinin aynı fiziksel sektöre uygulanması durumunda olacağından daha uzun sürmesini sağlar.[11]

Mikrodenetleyicilerde kullanılan flash belleklerin tutma sürelerinin kapasitesi, uzun yıllar süren aralıklarla programlama yapılmaması durumunda bellek bütünlüğüne engel oluşturmadığı için çalışmanın odağını bellek dayanıklılığının verimli kullanımı alanına kaydırmıştır. Bu yüzden tez kapsamında yapılan bütün çalışmalarda ve uygulamalarda bellek tutma süreleri göz ardı edilerek, bellek dayanıklılığına odaklanılmıştır.

2. GENEL KISIMLAR

2.1. MİKRODENETLEYİCİLER VE GÖMÜLÜ SİSTEMLER

Mikrodenetleyiciler(MCU), genellikle tek çip içerisinde CPU, flash bellek, RAM, I/O portları ve çeşitli çevresel birimlerin bir arada bulunduğu bilgisayar sistemleridir.[12]

Gömülü sistemler, sınırlı bellek ve işlem kapasitesiyle spesifik bir amaç için geliştirilen düşük maaliyetli ve yüksek performanslı özel işleve sahip bilgisayar sistemleridir. Gömülü sistemler genellikle MCU kullanılarak geliştirilmektedir. Günümüzde gömülü sistemler, otomotiv, havacılık, uzay, tıp, savunma, raylı sistemler, tarım ve tüketici elektroniği gibi her alanda kullanılmaktadır. Gömülü sistemlerin özellikle güç tüketimi, maaliyet ve performans gibi belirli amaçlar için optimize edilmesi kullanım alanını oldukça arttırmıştır.

Gömülü sistemlerde kullanılan mikrodenetleyicilerin yeteneklerinin artmasında ve geliştiriciler tarafından daha erişilebilir olmasında hiç şüphesiz flash bellek teknolojisinin gelişmesinin çok büyük bir katkısı vardır. Gömülü flash bellekler, sistem maliyetini, performansını ve gücünü tanımlamak için açık ara en önemli temel teknolojidir.[13]

2.2. FLASH BELLEK

Flash bellekler özellikle gömülü sistem tasarımlarında kalıcı hafıza ve programlanabilme özelliklerinden dolayı kritik öneme sahiptir. Mikrodenetleyicilerde flash bellek kullanmanın yüksek erişim hızı, güvenlik, düşük ısı dağılımı ve gürültüsüz olması gibi birçok avantajı vardır.[14]

Mikrodoenetleyicilerde bellek sistemleri olarak RAM ve flash belleklerin bir arada kullanılmasının birçok nedeni vardır. En önemli nedeni her iki bellek sisteminin farklı özelliklere ve kullanım şekillerine sahip olmasıdır. RAM bellekler elektriksel güç olmadan kalıcı bilgi saklayamazken flash bellekler yıllarca herhangi bir güç kaynağına ihtiyaç duymadan içerisinde bilgileri saklayabilirler. Dolayısıyla mikrodenetleyicilerde genellikle kalıcı hafıza ihtiyacı primer olarak flash bellekler tarafından sağlanır.

RAM belleklerin flash belleklere göre avantajlı olduğu yerler de vardır. Örnek vermek gerekirse, RAM bellekler flash belleklere göre daha hızlı yazma işlemi yapabilme ve daha fazla yeniden yazma kapasitesine sahiptir.

Flash belleklerde okuma ve yazma işlemleri RAM belleklerden daha farklı gerçekleşir. Flash belleklerde "sil" ve "yaz" olmak üzere iki temel işlem vardır. Silme işlemi için bellekte yer bit değeri bir olarak ayarlanır. Yazma işlemi için bellekte bir değerine sahip olan bit sıfıra ayarlanır. Yazma işleminin birimi tez kapsamında kullanılan mikrokontrolör için bir word birimidir. Silme işlemi ise sektör bazlı veya kütlesel olarak iki farklı şekilde gerçekleştirilebilir.

2.2.1. Flash Bellek Tutma Süresi

Flash belleğe yazılan verilerin bellekte ne kadar süre boyunca saklanabileceği veya geri çağırılıp kullanılabileceği, bellekte tutma süresi ile ifade edilir. Başka bir deyişle, flash bellekteki veri bitlerinin programlandıktan sonra aynı bit değerini minimum süre boyunca düzgün bir şekilde koruma ve istenildiği zaman bit değerini sağlama yeteneğidir. Günümüz flash bellek teknolojilerinde kılavuz olarak verilerin 10 yıldan fazla tutulacak şekilde saklanabildiği ifade edilse de, flash bellekte yazma silme işlemleri ve sıcaklık gibi bellekte tutma sürelerini etkileyen faktörler bulunmaktadır.

2.2.2. Flash Bellek Dayanıklılığı

Flash bellek dayanıklılığı, verilerin güvenli bir şekilde flash belleğe yazılıp silinebilme sayısı veya "P/E cycle" olarak belirtilen kapasite ifadesidir. Bellek dayanıklılık kapasitesine ulaşıldıktan sonra, bellekteki verilerin bütünlüğü ve doğruluğu güvenilmez hale gelebilmektedir. Dolayısıyla bellek kullanılmaz hale gelmektedir. Sıcaklı flash bellek tutma süresini etkilediği gibi bellek dayanıklılığını da etkileyen faktörlerden biridir.

2.2.3. Flash Belleklerde Yazma ve Silme Operasyonu

Mikrodenetleyicilerde kullanılan flash belleklerinin bir adresine yazılan veriler silinmedikçe aynı adresin üzerine tekrar veri yazılamaz. Flash belleğin silinmesi toplu veya sektörel olmak üzere iki farklı şekilde yapılabilir. Flash belleğe yazma işlemi ise belleğin mikro elektronik tasarımı nedeniyle toplu veya sektörel olmak zorunda değildir. Bu durum yazma

ve silme döngülerinin farklı zaman alabileceğini ve silme işleminin yazma işleminden daha uzun sürdüğü durumların daha fazla olduğunu ortaya koymaktadır. Bu durumu bir örnekle açıklamak istersek, belirli bir zaman aralığında 16 kilobayt sektör boyutuna sahip bir flash belleğe 1 kilobayt büyüklüğünde bir program yüklediğimizi varsayalım. Flash belleği önceden programlanmış adres aralığındaki bir bölgede 1 kilobayt boyutunda yeni bir programla yeniden programlamak için önce 16 kilobaytlık bellek sektörünü silmemiz ve ardından yeni 1 kilobaytlık programı yeniden yüklememiz gerekecek. Sektör boyutunun 1/16'sına karşılık gelen bir veri bloğunu programlamak için tüm sektörü silmek, hem silme-yazma döngü sürelerini artırır hem de sektör dayanıklılığını azaltır. Ayrıca 16 kilobaytlık flash bellek sektörü silinirken elektrik kesintisi olursa 1 kilobayt veri yerine 16 kilobayt veri kaybolacaktır.

Mikrodenetleyici üreticilerinin kullandığı flash belleklerin sektör büyüklükleri değişse de, sektör boyutundan daha küçük olan bir programın aynı sektörün başlangıç adresinden yazılması için yazılacak sektörün tamamının silinmesi gerekmektedir. Sektörün tamamının silinmesi, silme döngülerinin yazma işlemine kıyasla yavaş olması ve kısıtlı P/E kapasitesinin çabuk dolması gibi dezavantajları barındırır. Bu dezavantajlarla birlikte mikrodenetleyici üreticilerinin kullanıcılara sunduğu geliştirme platformları ve yazılımlar, bu yazıda bahsedilen klasik önyükleyici programlarının iskelet halini oluşturmaktadır. Mikrodenetleyici üreticileri örnek olarak yüklenecek linker dosyalarını, önyükleyici yazılımlarını, demo projelerini ve benzeri programları sıfır veya sabit bir flash bellek adresinden yüklenecek şekilde yapılandırmışlardır[1] [2] .

Sonuç olarak, yüklenen programın boyutu flash belleğin sektör kapasitesinden oldukça az olsa bile, bir sonraki program yüklemesi flash belleğin sıfır veya sabit bir adresinden başlayacağından sırasıyla silme ve yazma işlemleri aynı bellek bölgesinde yapılacaktır. Gömülü yazılım geliştirme süreçlerindeki deney ve test aşamaları göz önüne alındığında mikrodenetleyicilerin klasik yöntemlerle programlanması flash bellek yönetiminin oldukça verimsiz olduğu birkaç noktayı ortaya koymaktadır. Özellikle yüklenecek programın boyutunun flash bellek sektöründen daha küçük olduğu durumlar, optimizasyon için çok uygun bir konudur. Bu tez kapsamında mikrodenetleyicilerin içerisinde bulunan flash belleklerin bahsedilen dezavantajları ek bir donanıma ihtiyaç duymadan, wear leveling

[1] https://git.ti.com/cgit/hercules_examples/hercules_examples/tree/Bootloaders/, Temmuz 2021
[2] https://github.com/STMicroelectronics/STM32CubeF4/, Temmuz 2021

algoritmaları ve optimizasyon çalışmaları ile giderildiği gösterilecektir.

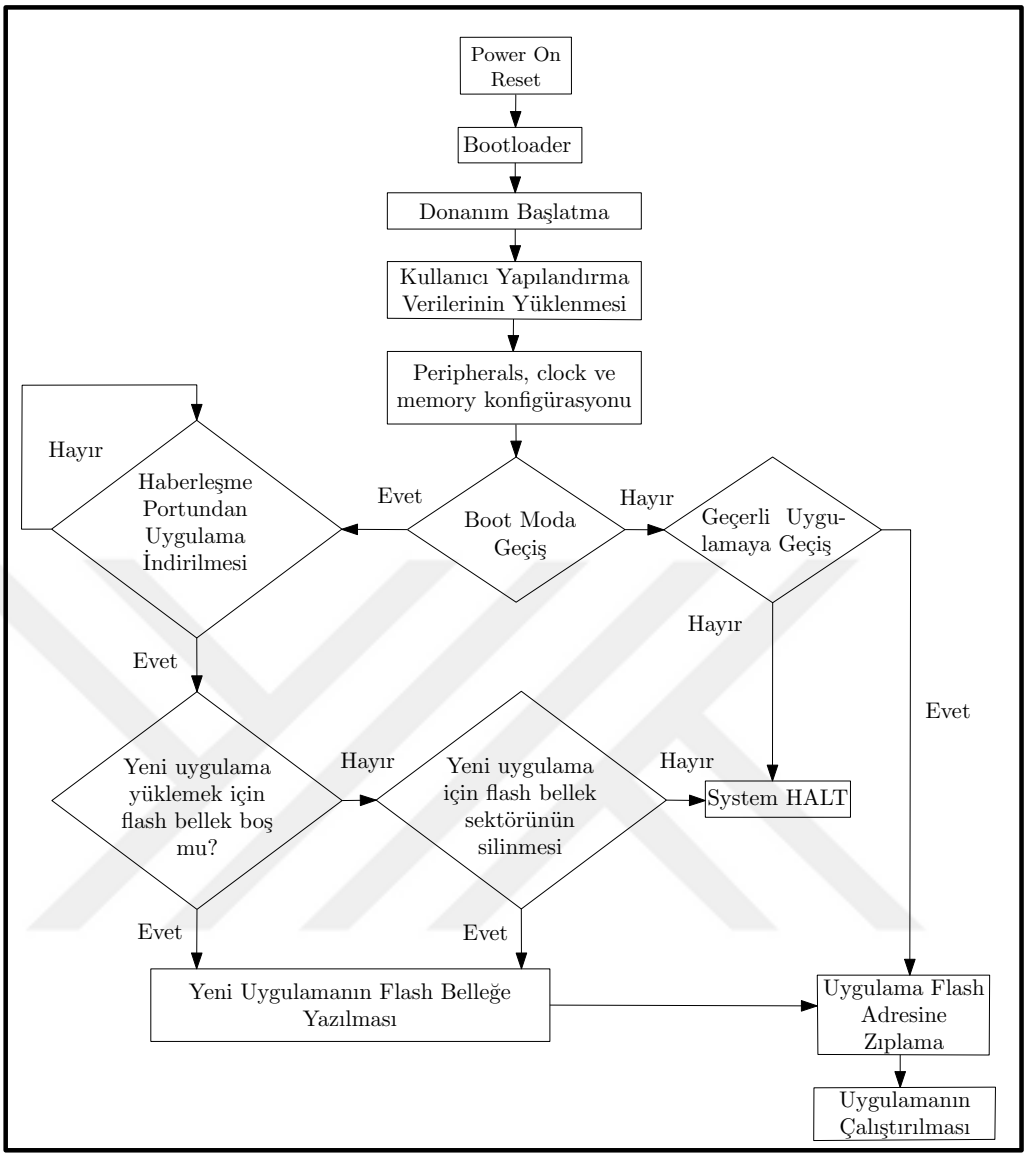
2.3. ÖNYÜKLEYİCİ

Önyükleyici sistem başlangıcında mikrodenetleyici üzerindeki flash bellekten bir uygulama yükleyen veya çalışan uygulamalar için bir güncelleme mekanizması olarak işlev görecek şekilde programlanabilen, bellek boyutuna kıyasla nispeten küçük yazılımlardır.[15] İşletim sisteminden bağımsız bare metal gömülü sistemlerde önyükleyici tasarımlarının birçok farklı boyutu ve çeşidi vardır. USART, CAN, I2C, SPI, Ethernet, USB gibi çeşitli protokoller üzerinden iletişim kurabilirler.[16][17] Her mikrodenetleyici ailesinin donanım konfigürasyonları farklı olabileceğinden, tüm mikrodenetleyiciler için standart bir önyükleyici yazılımı yoktur. Bu nedenle, donanım özelliklerine göre her bir mikrodenetleyici türü için kendi flash bellek organizasyonuna ve diğer çevresel birim bileşenlerine ait önyükleyici yazılımı gereklidir.[18] Ancak işlevsellik açısından genel bir önyükleyici mekanizmasını tanımlamak mümkündür.

2.3.1. Klasik Önyükleyici Tasarımları

Mikrodenetleyici üreticilerinin geliştiricilere sunduğu önyükleyici yazılımı, linker dosya örnekleri ve demo uygulamalarının programlama sürecinde flash belleğin sıfır veya sabit adresinden başlaması nedeniyle klasik önyükleyici tanımı yapılmıştır. Klasik önyükleyici yazılımı çeşitli boyutlarda ve farklı özelliklerde olabilir, ancak genel olarak, klasik bir önyükleyici yazılımının çalışması nispeten standarttır.[16] Bare metal sistemlerdeki klasik önyükleyici programları yedeklilik, güvenlik, emniyet gibi çeşitli özelliklerden bağımsız olarak değerlendirildiğinde genel akış şemasını Şekil 1.1'de verildiği gibi göstermek mümkündür.

Sistem açıldıktan sonra, işlemcinin erişilebilir durumda olmasını sağlamak için mikrodenetleyici register ve donanım birimleri başlatılır. Ardından, önyükleyici ana programı mevcut flash durumunu kontrol eder ve kullanıcı yapılandırma verilerini yükler. Yapılandırma tamamlandıktan sonra çevre birimleri başlatılır. Şu ana kadar akışta herhangi bir sorun yok ise boot veya uygulama moduna geçiş ile ilgili bir seçim aşaması vardır. Önyükleyici moduna girmenin birkaç yöntemi vardır, ancak mikrodenetleyicilerde önyükleme modu seçimi için I/O pinleri veya buton kullanmak yaygın bir yöntemdir.



Şekil 2.1: Klasik Önyükleyici Tasarımlarının Genel Akış Diagramı

Bu seçimde sistem boot moduna geçerse yeni program yüklemek için gerekli işlemler başlayacaktır. Uygulama moduna geçilmesi durumunda flash belleğe yüklenen geçerli bir uygulama yazılımı varsa uygulamanın başlangıç adresine sıçranır ve sistem çalışmaya başlar. Klasik önyükleyici yazılımı, standart bir uygulama yazılımından çok farklı değildir. Klasik önyükleyici yazılımını standart bir uygulama yazılımından ayırt eden özelliği, flash bellek alanını başka bir uygulama ile paylaşması ve onun yerine yeni bir uygulamayı silip programlayabilmesidir. [16]

3. MALZEME VE YÖNTEM

3.1. İMPLEMENTASYON İÇİN MİKRODENETLEYİCİ SEÇİMİ

Performans ölçümü için tasarım ve uygulamada ARM Cortex-M4 çekirdekli STM32F407VGT6 mikrodenetleyici kullanılmıştır. STM32F407VGT6 mikrodenetleyici, programları ve verileri depolamak için kullanılabilecek 1 Megabayt flash belleğe sahiptir[2]. STM32F407VGT6 mikrodenetleyicinin flash belleğinde bulunan toplam 12 sektörün boyutları sırasıyla , 4 sektör x 16 kilobayt, 1 sektör x 64 kilobayt ve 7 sektör x 128 kilobayt'dan oluşmaktadır[2]. STM32F407VGT6 mikrodenetleyicisinin flash bellek dayanıklılığı üretici firma tarafından minimum 10000 P/E döngüsü olarak belirtilmiştir[2]. Verimli önyükleyici tasarımı ve uygulamasında, ilk iki 16 kilobayt sektörler(Sektör 0 ve Sektör 1) verimli önyükleyici yazılımı için kullanılmaktadır. Üçüncü ve dördüncü sektörler(Sektör 2 ve Sektör 3) boot bilgilerini kaydetmek için eeprom emülasyon alanı olarak kullanılmaktadır. Beşinci sektörden on ikinci sektöre kadar olan sektörler(Sektör 4,5,6,7,8,9,10,11) kullanıcı uygulamalarının yükleneceği program alanı olarak kullanılmaktadır. Verimli önyükleyici tasarımında flash bellek blokları toplamda 3 bölüme ayrılmış ve her parça için ayrı bir silme algoritması kullanılmıştır. Kullanılan algoritmalar ilerleyen bölümlerde ayrıntılı olarak ele alınacaktır.

3.2. GÖMÜLÜ SİSTEM TEST DÜZENEĞİ

3.2.1. Sisteme Genel Bakış

Test ortamı oluşturulurken kullanılan araçların geliştirilmeye açık, erişilebilir, ücretsiz ve/veya açık kaynak kodlu olmasına özen gösterilmiştir. Bu amaç doğrultusunda çalışmaların daha sonraki aşamalarda geliştirilmeye açık ve kapsamının genişletilebilir olması hedeflenmektedir.

Verimli önyükleyici tasarım ve implementasyonu kapsamında ana mikrodenetleyici sistemi

olarak STMicroelectronics firmasının STM32F4DISCOVERY[3] geliştirme kartı seçilerek, tüm araştırma çalışmaları USB Portlu bilgisayar ve UART bağlantılı önyükleyici cihazı üzerinden gerçekleştirilmektedir.

STM32F4DISCOVERY Discovery kiti, kullanıcıların Arm Cortex-M4 32-bit çekirdekli STM32F407VG yüksek performanslı mikrodenetleyici ile uygulamaları kolayca geliştirmelerini sağlar.

STM32F407xx mikrodenetleyici ailesi, yüksek hızlı gömülü bellekler(1 MB'a kadar flash bellek, 192 KB'a kadar SRAM), 4 KB'a kadar yedek SRAM ve iki APB veriyoluna, üç AHB veriyoluna ve bir 32-bit çoklu AHB veriyolu matrisine bağlı çok çeşitli gelişmiş I/O'lar ve çevre birimleri içerir.[2]

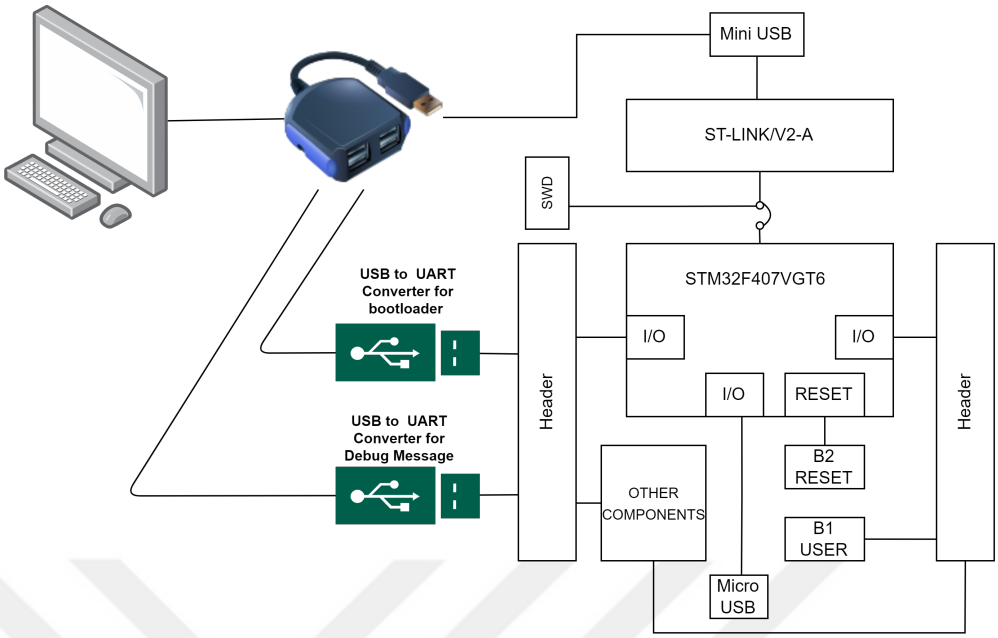
Verimli önyükleyici yazılımında, tasarım ve test aşamasında hızlı ve yaygın bir ilerleme sağlamak için UART iletişim portları kullanılmaktadır. STM32F407VG mikrodenetleyici USB, CAN, SPI, I2C çevre birimleri içerdiğinden UART yerine alternatif bir iletişim portu olarak kolayca yapılandırılabilir. Ayrıca, verimli önyükleyici tasarımı, flash bellek boyutu ve cihaza özgü işlevlerdeki farklılıkları ortadan kaldırarak ARM Cortex M0, M3 ve M4 çekirdekli farklı mikrodenetleyici sınıflarına kolayca uyarlanabilir.

Deney düzeneğinin yapısı Şekil 3.1'de gösterilmiştir. Tüm sistem, bilgisayarın USB portu tarafından beslenmektedir. Bilgisayarın USB portuna bir USB hub bağlanarak 2 adet USB to UART çevirici ve 1 adet Mini USB to USB kablo bağlantısı yapılmıştır. USB'den UART'a dönüştürücülerden biri, önyükleyici ile Komut Satırı Arayüzü(CLI) ile iletişim kurmak için kullanılır. Diğer USB'den UART'a dönüştürücü, hata ayıklama mesajları göndermek için kullanılır. Güç açık sıfırlama için STM32F4DISCOVERY üzerinde bulunan B2 RESET düğmesi kullanılır. STM32F4DISCOVERY üzerinde bulunan B1 USER düğmesi, sistem sıfırlandıktan sonra boot modu veya kullanıcı uygulama modu seçimi için kullanılır.

3.2.2. Geliştirme Ortamı

Yazılım geliştirme amacıyla tümleşik geliştirme ortamı(IDE) çalıştıran ve yedek USB bağlantısı olan bir bilgisayar kullanılmıştır. Bu makalede açıklanan verimli önyükleyici programı, STM32CubeIDE(Sürüm 1.6.0) geliştirme ortamı ve GNU Make aracı kullanılarak

[3] https://www.st.com/en/evaluation-tools/stm32f4discovery.html, Haziran 2021.



Şekil 3.1: Deney Düzeneğinin Yapısı

geliştirilmiş ve test edilmiştir. İşletim sistemi olarak açık kaynaklı bir işletim sistemi olan Linux tabanlı Ubuntu (20.04 focus) kullanılmıştır.

Mini USB bağlantısı hem sisteme güç sağlamak hem de önyükleyici yazılımını STM32F4DISCOVERY'ye yüklemek için kullanılır. Yüklenecek önyükleyici yazılımı, geliştirme kitinin içerisinde yer alan ST-LINK/V2-A[4] donanımına aktarılır ve oradan MCU'ya yüklenir. ST-LINK/V2-A ayrıca yerleşik bir hata ayıklayıcı olarak kurulur ve MCU'da hata ayıklamak için kullanılır. Önyükleyici yazılımı bir kez yükledikten sonra uygulama yazılımları artık seri haberleşme portu aracılığıyla CLI programı üzerinden yüklenmeye başlar. Bu aşamadan sonra debug dışında ST-LINK/V2-A donanımına ihtiyaç kalmamaktadır. Böylece önyükleyici yazılımlarının donanım gereksinimini giderme ve maaliyeti düşürme gibi önemli amaçlarına ulaşılmış olunur.

Önyükleyici yazılımında C ve C++ programlama dilleri hibrit olarak kullanılmıştır. C++ dilinin sahip olduğu standart kütüphaneden yararlanmak ve yazım kolaylığı sağlamak amacıyla yardımcı fonksiyonların bir kısmında C++ kullanılırken, yazılımın büyük çoğunluğunda üretici firmanın sağladığı donanım soyutlama katman kütüphanelerinin direk kullanılması amacıyla C dili tercih edilmiştir. Mikrodenetleyici ile bilgisayar arasındaki CLI yazılımında python dili tercih edilmiştir.

[4] https://www.st.com/en/development-tools/st-link-v2.html, Haziran 2021

3.3. WEAR LEVELİNG ALGORİTMALARI

Daha önceki başlıklarda da bahsedildiği üzere gömülü sistemlerde flash bellekler hem uygulama kodunun saklanması hem de veri depolama amacı için kullanılmaktadır. Özellikle test amaçlı kullanımlarda veya ürün geliştirme aşamalarında mikrodenetleyiciler oldukça yoğun bir şekilde programlanmaktadır. Fakat flash bellekler sınırlı sayıda P/E döngüsüne sahiptir. Dolayısıyla flash belleklerin üretici firmaları tarafından garanti edilen P/E döngü sayısı gömülü sistem ürünü olarak geliştirilen bir ürünün ömür beklentisini etkileyen önemli bir faktör olmaktadır. Gömülü sistem ürünlerinin ömür beklentisini arttırmak amacıyla geliştirilen flash bellek kullanım yöntemlerden biri wear leveling algoritmasıdır.

Wear leveling algoritması flash belleğe yazılacak verilerin flash bellek sektörleri arasında eşit bir şekilde paylaştırılmasını ve gereksiz yere bellek silme işlemlerini önleyerek flash belleklerin dayanıklılığını arttırmaya yardımcı olan bir algoritmadır.[19][20] Wear leveling algoritması her veri yazma işlemi esnasında kullanılacak fiziksel adres konumunu belirleyerek verilerin bellek dizisi içinde herhangi bir yerde saklanabilmesini sağlar.[21]

Wear leveling algoritması aynı amaca hizmet eden iki farklı yöntemle uygulanabilmektedir. Bu yöntemler statik ve dinamik wear leveling algoritması olarak ifade edilmektedir. Flash belleklerde statik ve dinamik olmak üzere iki tür veri bulunmaktadır. Statik veri uzun süre bekleyen, nadiren güncellenen verilerdir. Statik verileri sıklıkla okunabilir ama nadiren değişen veriler olarak da düşünebiliriz. Dinamik veriler ise sürekli silinen veya güncellenen ve bu yüzden sık sık programlanan verilerdir.

3.3.1. Dinamik Wear Leveling

Dinamik wear leveling algoritmasında, programlanmamış flash bellek blokları içerisinde en düşük silme sayısına sahip bellek bloğu seçilir. Dinamik wear leveling algoritması, P/E döngülerinin ve güncellemelerin düzenli olarak(veya sıklıkla) yapıldığı toplam bellek alanının bir bölümünde uygulanır.[11][22] Dolayısıyla dinamik wear leveling algoritması kullanan sistemler algoritmayı tüm sektörlerde uygulamak yerine sadece dinamik verilerin güncellendiği sektörlere uygular. Bu yüzden statik verilerin bulunduğu sektörlere algoritma uygulanmaz. Dinamik wear leveling algoritması kullanan sistemler herhangi bir wear leveling algoritması kullanmayan sistemlere göre daha fazla flash bellek dayanıklılığı sunar fakat mevcut flash belleğin sadece belli bir bölümüne yoğunlaşır.

Dinamik wear leveling algoritmasında sık güncellenmeyen(soğuk veriler) bölümler ile ilgili herhangi bir işlem yapılmadığı ve sadece belleğin sık güncellenen(sıcak veri) bölümleri ile ilgili işlem yapıldığı için flash bellek bloklarındaki aşınma dağılımında yüksek farklar oluşmaktadır. Dinamik wear leveling algoritması sık güncellenen veri bloklarında verimli olurken, sistem genelinde performans kaybı olarak değerlendirilmektedir.

Bu tez kapsamında sadece eeprom emülasyonu için ayrılan sektörlerde dinamik wear leveling algoritması kullanılmıştır. Eeprom emülasyonu için ayrılan sektörler sistemin genelinden bağımsız olarak çalıştığı ve diğer flash bellek bölgelerine göre daha sık güncellendiği için dinamik wear leveling algoritması tercih edilmiştir.

3.3.2. Statik Wear Leveling

Statik wear leveling algoritmasında dinamik wear leveling algoritmasından farklı olarak soğuk veriler daha fazla aşınmış bloklara taşınmaya çalışılarak, aşınmanın daha eşit yayılmasını amaçlanır. Statik wear leveling algoritmasında flash bellekte bulunan tüm bloklar için P/E eşiği önceden tanımlanır.[23] Flash belleğe yazma işlemi için başvurulduğunda önceden tanımlanan P/E eşik değerine ulaşılmışsa, P/E döngü sayısı en yüksek olan bloklardan en az olanlara veriler taşınır. Böylece flash bellek bloklarının tamamında aşınma yoğunluğu eşit tutulmaya çalışılır.

Statik wear leveling algoritmasında sıcak veri bölgesi ile soğuk veri bölgesi arasında verilerin yer değiştirmesi flash belleğin genel performansını düşürebilir. Çünkü sisteme fazladan veri değiş tokuş yükü eklenmiş olur. Ek olarak soğuk verileri herhangi bir güncelleme talebi olmadan sıcak blokta hareket ettirmek gereksiz aşınmaya yol açabilmektedir.

Düşük P/E sayısına sahip bloklarda yer açmak için statik veri taşıma adımı yazma performansını yavaşlatabilir ve bellek blok ömrünün bir kısmını tüketebilir. Dinamik wear leveling yöntemi ile karşılaştırıldığında, statik wear leveling algoritması, flash bellek dayanıklılığı açısından daha iyi yöntem olarak kabul edilir.

3.3.3. Verimli Önyükleyici Tasarımına Özel Wear Leveling Algoritması

Dinamik ve statik wear leveling algoritmalarının yukarıdaki başlıklarda belirtilen dezavantajlara sahip olması implementasyona özel bir algoritma ihtiyacını doğurmuştur.

Yeni tasarlanan wear leveling algoritması dinamik ve statik wear leveling algoritmalarının sahip olduğu özelliklerden faydalanılarak türetilmiş ve daha optimize bir çözüm amaçlanmıştır.

Verimli önyükleyici tasarımında yüklenecek uygulamalar için ayrılmış, implementasyonda kullanılan mikrodenetleyicinin flash bellek sektörleri, flash sektör boyutundan daha küçük ve eşit sanal bellek bloklarına ayrılmıştır. İmplementasyonda sanal bellek blokları fiziksel adreslerine göre sıralanmıştır. Verimli önyükleyici yazılımında yeni uygulama yükleme talebi geldiğinde yüklenecek uygulama boyutuna göre en uygun sanal bellek blok adresi belirlenmeye çalışılır. En uygun sanal bellek blok adresi uygulama boyutuna göre yeterli büyüklükte ve yeterli sayıda sıralı sanal bellek blokları içerisinden seçilir. Seçim koşulunu sağlayan sanal bellek blokları içerisinde blok numarası en küçük olan bloğun önceliği en yüksek olacak şekilde algoritma oluşturulmuştur. Oluşturulan algoritma soğuk ve sıcak veri alanlarından bağımsız olarak tüm bellek bloklarında eşit aşınma yoğunluğunu hedeflemektedir. Flash bellek alanı için belli bir eşik seviyesi tanımlanmamıştır. Flash bellek bloklarında yüklenecek yeni uygulama için yeterli alan bulunmuyorsa daha önce yazma işlemine uğramış bütün bloklar silinir ve ilk bloktan itibaren yazma işlemi uygulanır. Böylece soğuk ve sıcak bölgeler arasında veri aktarımı yapılmamış olunur. Silme işlemi sadece uygulama yükleme anında meydana gelebileceği için harcanan süre çalışma zamanındaki performansı etkilememektedir. İmplementasyona özel wear leveling algoritmasıyla ilgili detaylar sonraki başlıklarda ayrıntılı olarak ele alınacaktır.

3.3.4. No Wear Leveling Algoritması

Bazı uygulamalar sık sık güncellenmediği için sistem ömrü gereksinimi, wear leveling algoritması olmadan karşılanabilir. Verimli önyükleyici tasarımına bütünsel olarak baktığımızda önyükleyici yazılımının sadece bir kez programlanacağı veya sadece önyükleyici yazılımına ek özellikler eklemek istersek güncelleme ihtiyacının olacağı sonucuna varabiliriz. Bu tez kapsamında tasarlanan sistem üç ana bölümden oluşmaktadır. Bu bölümler önyükleyici yazılımı, eeprom emülasyonu ve uygulama yazılımlarından oluşmaktadır. Önyükleyici yazılımı için ayrılan flash bellek sektörlerinde herhangi bir wear leveling algoritması kullanılmamıştır. Önyükleyici yazılımı ile ilgili detaylar sonraki başlıklarda ayrıntılı olarak ele alınacaktır.

3.4. VERİMLİ ÖNYÜKLEYİCİ YAZILIM TASARIMI

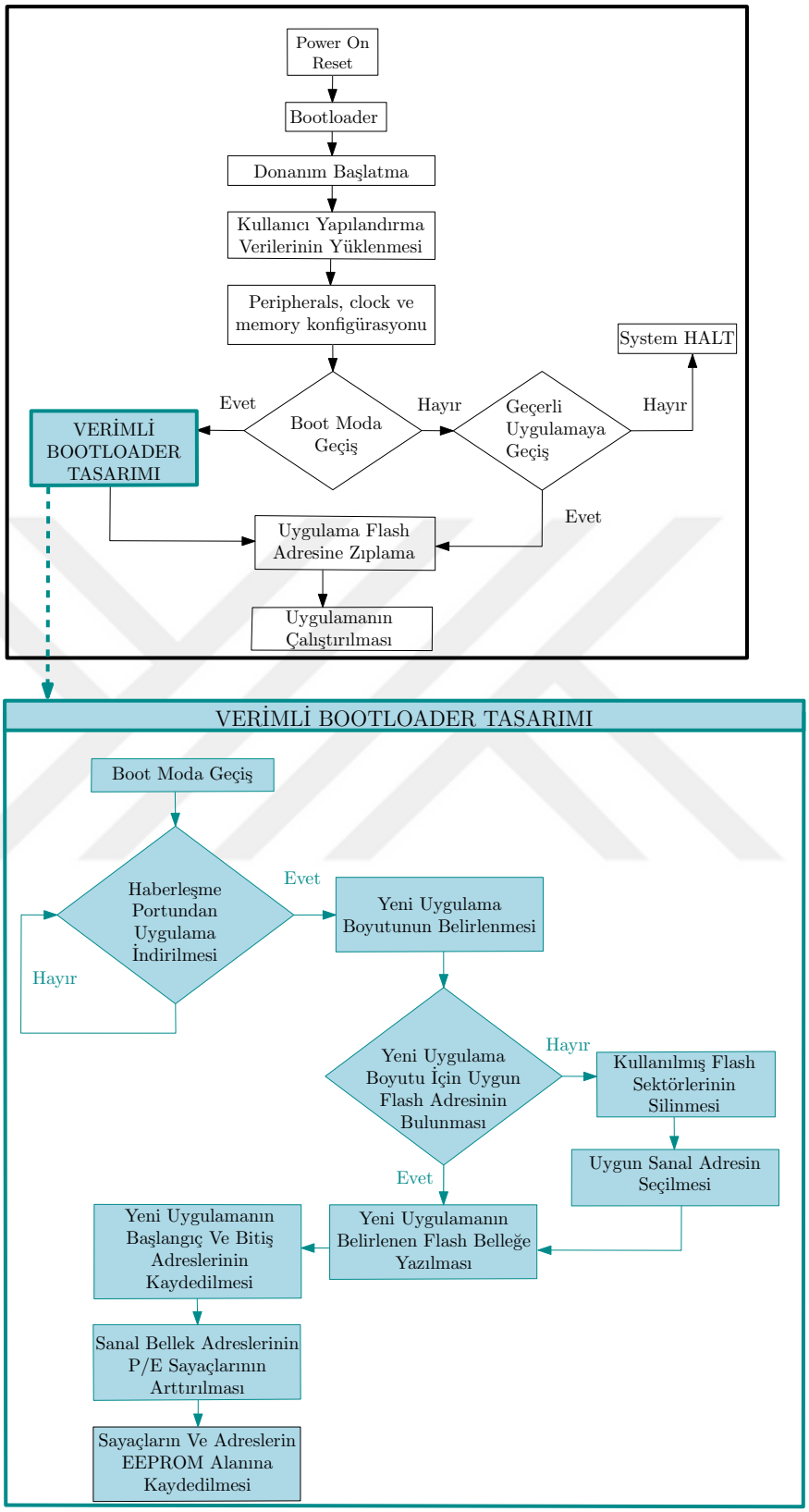
Verimli önyükleyici tasarımı, önyükleme modundan sonra flash belleğe yazılacak verilerin boyutuna göre en uygun flash bellek başlangıç adresini belirlemeyi amaçlar. Belirtilen adreslerle olabildiğince homojen bir bellek yazma dağılımına sahip olunması istenmektedir. Böylece flash belleğin dayanıklılığı artarken flash yazma sürelerinde de etkili bir azalma olacaktır. Verimli önyükleyici tasarımı ve klasik önyükleyici tasarımları, temel olarak önyükleme moduna kadar aynı işlevselliğe sahiptir. Verimli önyükleyici örnek tasarımı Şekil 3.2'de gösterildiği gibidir.

Şekil 3.2'de klasik önyükleyici ve verimli önyükleyici tasarımlarının boot moduna geçilene kadar ki yaşam döngüsünde benzer aşamalardan geçildiği ifade edilmektedir. Verimli önyükleyici tasarımında boot moduna geçtikten sonra yüklenecek yeni uygulamanın boyutuna göre bellekte en uygun flash adresi belirlenir. Klasik önyükleyici tasarımlarından farklı olarak uygulama yazılımının yükleneceği flash bellek başlangıç adresi, uygulama boyutuna göre çalışma anında dinamik olarak atanır. Bellek dayanıklılığını arttırmak için tasarlanan en uygun adres belirleme algoritması, yeterli büyüklükte ve yeterli sayıda sıralı sanal adres bloklarının olduğu durumlarda, uygulama yükleme aşamasında sektörü silme işlemine maruz bırakmadan uygulamanın belleğe yazılmasını sağlar. Böylece flash bellekte yazma işlemi için yeterli alan kalmayıncaya kadar flash bellek sektörlerinde silme işlemi uygulanmaz. Sonuç olarak bellek dayanıklılığı arttırılır ve bellekte yazma işlemi için harcanan süreler kısaltılmış olur.

3.4.1. Verimli Önyükleyici Yazılım İzolasyonu

Verimli bir önyükleyici tasarımının uygulanabilmesi için flash bellekte belirli bir alanın önyükleyici yazılımı için ayrılması gerekmektedir. Verimli önyükleyici yazılımı sürekli olarak güncellenmeyeceği ve sadece bir kere yükleneceği için flash bellek alanında sanal bloklar yerine flash bellek sektörü veya sektörleri kullanılmaktadır. Bu alanlar, uygulamaların ve eeprom emülasyonunun yazıldığı bellek alanlarından izole edilmektedir. Böylece diğer flash bellek alanları yazma ve silme işlemine maruz kalırken, verimli önyükleyici tasarımının bulunduğu flash bellek alanlarında herhangi bir yazma veya silme işlemi olmaycaktır.

İzolasyon, bu faydalarının yanı sıra, önyükleyici tasarımının bulunduğu sektörlere



Şekil 3.2: Verimli Önyükleyici Tasarımı

erişim yetkilendirmesi, uygulama yazılımlarının şifrelenmesi veya şifresinin çözülmesi, kriptografik işlemler için anahtarların saklanması, önyükleyici yazılımına ek özellikler getirmesi ve farklı haberleşme protokollerinin kullanılması gibi uygulama yazılımından bağımsız olarak güvenlik, emniyet ve kullanışlı özelliklerin geliştirilmesine olanak tanır.

3.4.2. Programlanabilir Sanal Bellek Blokları

Mikrodenetleyicilerin flash bellek sektör boyutları birbirinden farklı olabilir ve flash sektörlerinin boyutu eşit olmayabilir.[24] Önyükleyici tasarımının kapsamlı ve verimli olmasını sağlamak için flash bellek alanları, sektörlerden bağımsız olarak sektör boyutundan daha küçük sanal bloklara bölünmüştür. Bu çalışma kapsamında sanal bellek blokları 16 kilobayt veri bloklarından oluşturulmuştur.

Yeni bir uygulama yükleneceği zaman, eğer bellekte yeterince sıralı ve boş bellek alanı varsa flash bellekte oluşturulan sanal bloklarda yüklenecek programın boyutuna göre flash bellek sektörleri silinmeden yeni uygulama yüklenebilecektir.

Yazma işlemi, doldurulmuş her sanal bloktan sonra, bir sonraki sanal bellek başlangıç adresinden yazmaya devam edecektir. Her yeni program yükleme talebi, uygun ve sıralı bellek adresleri olduğu sürece flash bellek sektörlerini silmeden yazmaya devam eder. Yeni programlar flash belleğe yazıldıktan sonra bellekteki başlangıç ve bitiş adresleri kaydedilir. Yazma işlemine maruz kalan sanal bellek bloklarının P/E sayısı artırılır. Böylece tüm sanal bellek blok adreslerinin dayanıklılığını gösteren sayaçlar elde edilmiş olur.

Flash bellekteki sanal bloklar yeni bir uygulama yüklemek için yeterli veya sıralı bellek adreslerine sahip olmadığında, daha önce yazılmış tüm sektörler silinir ve yeni uygulama ilk sektördeki ilk sanal blok adresinden itibaren yüklenmeye başlar. Uygulama yüklendiğinde verilerin yazıldığı sanal blokların P/E sayaçları artırılarak kayıt edilir. Program yükleme işlemi başarılı bir şekilde tamamlandıktan sonra uygun adrese atlama ve uygulamayı başlatma işlemleri klasik önyükleyici tasarımındaki mekanizmaya benzer işlemlere sahiptir.

3.4.3. Sanal Bellek Blok Bilgilerinin İzolasyonu

Flash bellek sektörlerinde verimli önyükleyici yazılımı ve sanal bellek bloklarının sayaç-adres bilgileri kendi içlerinde izole edilmiştir. Sanal bellek blok bilgileri

her kurulumdan sonra güncellenmektedir. Uygulama ve önyükleyici yazılımının bu güncellemelerden etkilenmemesi için ayrı bir sektörde olmasına karar verilmiştir. Sanal bellek blok bilgilerinin bulunduğu sektörlerin dayanıklılığı diğer sektörlerin dayanıklılık anlamında verimliliğini etkileyen önemli bir faktör olduğu için sanal bellek blok bilgisinin bulunduğu sektörlerin verimli kullanılması başlı başına bir tasarım konusu olarak ele alınmaktadır.

Verimli önyükleyici tasarımında sanal bellek blok bilgilerinin saklandığı flash bellek alanları en az iki sektörden oluşmalıdır. En az iki sektör kullanılmasının nedeni sektör silme işlemi gerektiğinde silinecek sektördeki verilerin yedeklenebilmesi içindir. Sanal bellek blok bilgisi, uygulama yazılımını belleğe kaydetme algoritması ile benzer bir yapıda kurulmuştur. Böylece tasarımın verimliliğini azaltacak etkilerin ortadan kaldırılması hedeflenmiştir. Yüklenecek uygulama ile ilgili sanal flash bellek bloklarının başlangıç adres bilgileri, bitiş adres bilgileri ve sayaç bilgileri her güncellemeden sonra sıralı olarak izole flash bellek sektörüne kaydedilmektedir.

Kayıt için ayrılan birinci flash bellek sektörünün sonuna ulaşıldığında, birinci sektörden gelen veriler ikinci ayrılmış sektör ile değiştirilir. Daha sonra ayrılan ve kullanılan ilk flash bellek sektörü tamamen silinir. Kayıt için ayrılan ikinci flash bellek sektörünün sonuna ulaşıldığında, veriler ilk silinen sektör ile değiştirilir ve bu silme-değiştirme işlemleri döngüsel olarak devam eder.

Daha önce de ifade edildiği gibi, bellek blok bilgilerinin saklanması için en az iki flash bellek sektörünün kullanılması, sanal bellek blok bilgilerinin sürekli güncel olmasını ve sektör yedekliliğini sağlar. Dikkat edilmesi gereken noktalardan biri, sanal bellek blok bilgilerinin tutulduğu sektörlerin boyutları, kaydedilen verilerin boyutundan yeterince büyük değilse, kayıt için kullanılan sektörlerin hızlı bir şekilde dolmasına ve silinmesine neden olacaktır. Sanal bellek blok bilgilerini içeren sektörlerin P/E sayısı, uygulamaların yüklendiği sektörlerin P/E sayısından fazlaysa, önyükleyici yazılımının dayanıklılık anlamında verimliliği düşecektir. Bu nedenle, flash bellek sektörlerinin seçimi, her bir önyükleyici yazılımı, sanal bellek blok bilgisi ve uygulama yazılımı hususları için kritik öneme sahiptir.

3.5. VERİMLİ ÖNYÜKLEYİCİ YAZILIM İMPLEMENTASYONU

Verimli önyükleyici tasarımı klasik önyükleyici tasarımlarından farklı olarak sabit bir flash adresinden itibaren uygulama yükleme işlemi yapmayarak yükleme anında dinamik adresleme ile uygulamanın yükleneceği adresi belirlemektedir. Verimli önyükleyici yazılımı için flash bellekte belirli sektörler seçilip, diğer flash bölgelerinden izole edilmiştir. Verimli önyükleyici yazılımının kolay ve etkin kullanımı için komut satır arayüzü(CLI) uygulaması geliştirilmiştir. Böylece kullanıcı yeni bir uygulama yükleyeceği zaman veya önyükleyici ile ilgili çeşitli bilgilere erişmek istediği zaman CLI ile etkileşim halinde olması gerekmektedir. Verimli önyükleyici yazılım implementasyonu ARM Cortex M4 ailesine ait spesifik bir MCU kullanılarak geliştirilmiş olsa da, ARM Cortex M0/M0+, M3 ve diğer M4 ailesinden gelen MCU'lar ile birkaç değişiklik yapılarak adapte edilebilir halde tasarlanmıştır.

3.5.1. Önyükleyici Komut Satırı Arayüzü(CLI) Uygulaması

Verimli önyükleyici uygulaması ile bilgisayar arasındaki iletişimi sağlamak için Python dili kullanılarak seri port haberleşmesi sağlayan CLI uygulaması geliştirilmiştir. CLI yazılımı ile önyükleyici yazılım versiyon bilgisi, MCU ID kodu, okuma koruma seviyesi (RDP), one time program (OTP) alan bilgisi, sektörlerin okuma/yazma durum bilgisi ve önyükleyici log bilgisi sorgulanabilmektedir. Ayrıca CLI yazılımı, programı belirtilen adrese atlama, flash sektörlerini silme, sektörlerin erişim kontrolü ve yeni uygulamanın kaynak dosyasından uygun formatta yürütülebilir dosyalar oluşturma işini yapar. Yeni uygulamaların kaynak dosyasından yürütülebilir dosya üretim prosesi ileride ki başlıklarda ayrıntılı olarak ele alınacaktır.

Seri bağlantı portuna erişmek için pyserial kitaplığı kullanılmıştır. Önyükleyici yazılımı ile CLI arasındaki seri haberleşme, Döngüsel Yedeklilik Kontrolü(CRC)[25] tarafından kontrol edilir. Böylece, CLI ile önyükleyici yazılımı arasındaki iletişimde veri bütünlüğü garanti edilir.

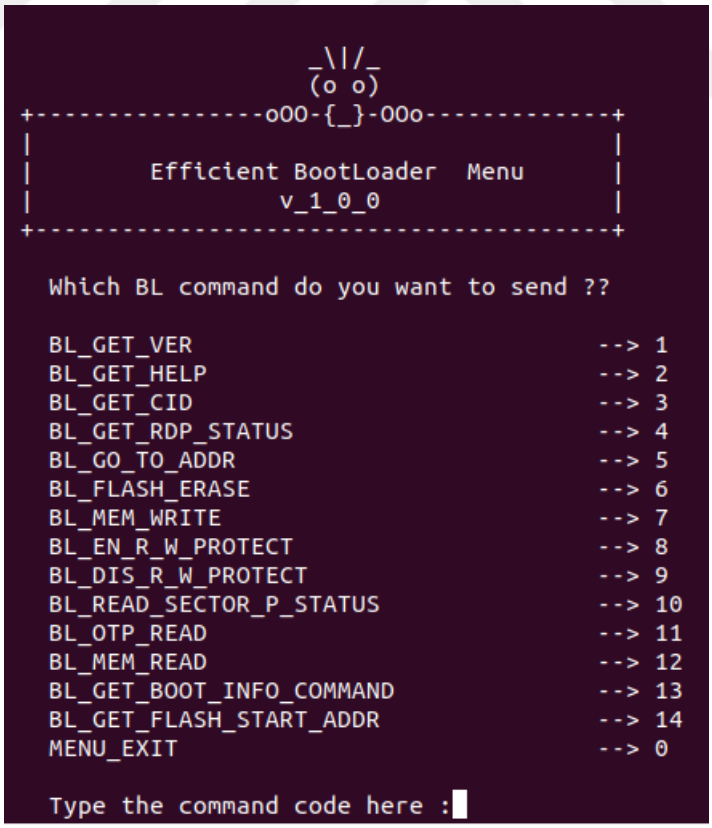
Boot mod seçimi her sistem için farklı sekanslar seçilerek düzenlenebilecek bir program akışıdır. Boot mod seçimi ile ilgili standartlaşmış bir proses olmamakla birlikte, donanımsal veya yazılımsal süreçler belirlenerek boot sekansı elde edilebilmektedir. Boot mod seçiminin temel amacı, sistem resetlendikten sonra yeni bir program akışının mı sağlanacağını yoksa mevcut program akışından devam mı edileceğini kontrol edebilmektir. Bu tez kapsamında

boot mod seçimi için geliştirme kiti üzerinde bulunan bir buton kullanılmıştır.

Her sistem resetinden sonra program akışı main fonksiyonuna geldiğinde çevre birimleri ve saat başlatma işlemlerinden sonra kullanıcı buton durumu kontrol edilir. Kullanıcı buton durumuna göre program akışının boot modundan veya kullanıcı uygulama modundan devam edeceği belirlenir. Bu işlemler için STM32F4 Discovery geliştirme kartı üzerinde bulunan B1 Kullanıcı butonu kullanılmaktadır. Mod seçimi esnasında buton durumu pasifse program akışı kullanıcı uygulamasına atlayarak devam eder. Mod seçimi esnasında buton durumu aktifse, program akışı boot moduna geçer. Bu aşamadan sonra verimli önyükleyici programı çalışır ve CLI'den veri bekler. Boot modunda CLI'den gelen komutlara göre alınan aksiyonlar sonraki konu başlıklarında incelenecektir.

3.5.1.1. CLI Önyükleyici Menüsü

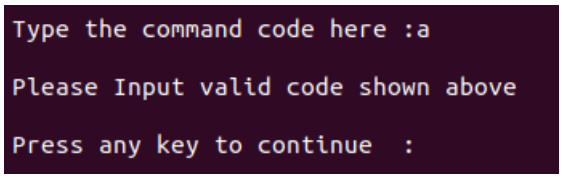
CLI önyükleyici, kullanıcıların cihaz içerisinde gömülü olan önyükleyici yazılımı ile etkileşimini sağlamak amacıyla geliştirilmiştir.



Şekil 3.3: CLI Önyükleyici Menüsü

Şekil 3.3'te CLI önyükleyici menü ekranı görülmektedir. CLI önyükleyici menü ekranı,

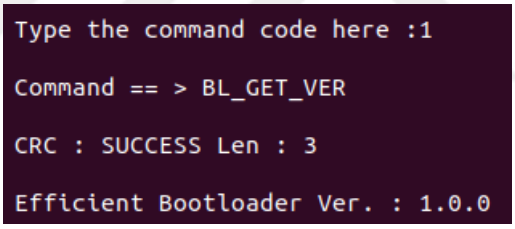
kullanıcıların listelenen opsiyonlar ile seçim yapabileceği bir yönlendirme ekranıdır. Her bir opsiyona bir numara atanmıştır ve kullanıcıdan bu numaraları kullanarak bir seçim yapması beklenir. Listede olmayan bir numara veya geçersiz bir karakter girildiğinde, CLI uygulaması geçerli bir komut girilmesi için uyarır ve tekrar seçim yapılmasını bekler. Şekil 3.4'te CLI uygulamasının geçersiz komut cevabı gösterilmektedir.



Şekil 3.4: CLI Geçersiz Komut Cevabı

3.5.1.2. CLI Önyükleyici Yazılım Versiyon Komutu

Verimli önyükleyici yazılımında versiyon takibi yapabilmek için versiyon sorgu komutu eklenmiştir. Şekil 3.5'te CLI uygulamasında yazılım versiyon sorgu komutunun cevabı gösterilmektedir.

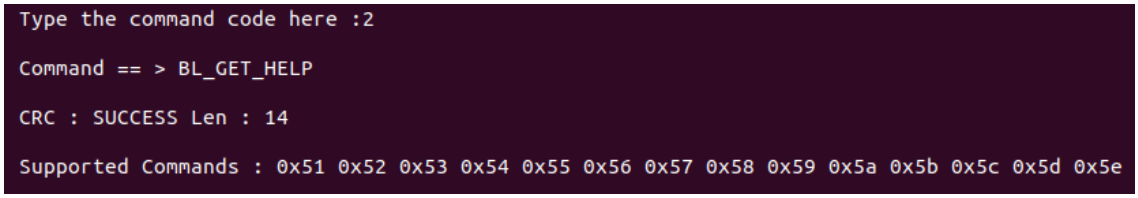


Şekil 3.5: CLI Yazılım Versiyon Komut Cevabı

Versiyon bilgisi sırasıyla major, minor ve build değişikliklerini ifade eden üç kategoriden oluşmaktadır.

3.5.1.3. CLI Önyükleyici Yardım Komutu

CLI önyükleyici sorgularının sayısal değerlerini elde etmek için yardım komutu eklenmiştir.



Şekil 3.6: CLI Yazılım Versiyon Komut Cevabı

Şekil 3.6'da yardım sorgu komutunun cevabı gösterilmektedir.

3.5.1.4. CLI CID Komutu

Önyükleyici yazılımının yüklü olduğu cihazın kart kimlik numarası(CID) CLI CID komutu ile sorgulanabilmektedir. Şekil 3.7'de CID sorgu komutunun cevabı gösterilmektedir.

```
Type the command code here :3

Command == > BL_GET_CID

CRC : SUCCESS Len : 2

Chip Id. :  0x413
```

Şekil 3.7: CLI CID Komut Cevabı

3.5.1.5. CLI RDP Durum Komutu

Flash bellek üzerinde okuma erişimini kontrol etmek amacıyla okuma koruma seçeneği(RDP) baytı bulunmaktadır. Okuma koruması, flash bellekte saklanan yazılım kodunu korumak için kullanılır.

İmplementasyonda kullanılan MCU'da üç farklı koruma seviyesi bulunmaktadır. "0xAA" değeri "Level 0" seviyesini belirtir ve herhangi bir korumanın olmadığı anlamına gelmektedir. "0xCC" değeri "Level 2" seviyesini belirtir ve belleğe herhangi bir okuma erişim izni verilmeyeceği anlamına gelmektedir. Bu iki değer dışında ki tüm değerler "Level 1" seviyesini belirtir ve boot modu olmadan herhangi bir erişim veya herhangi bir debugger erişimi algılanma durumunda sistemin donanım hatası oluşturacağı anlamına gelmektedir. Şekil 3.8'de RDP durum sorgu komutunun cevabı gösterilmektedir.

```
Type the command code here :4

Command == > BL_GET_RDP_STATUS

CRC : SUCCESS Len : 1

RDP Status :  0xaa
```

Şekil 3.8: CLI RDP Durum Komut Cevabı

RDP durum kontrolü ile ilgili en önemli noktalardan biri okuma koruma seviyesinin tasarımı engellemeyecek şekilde ayarlanmış olması gerektiğidir. Unutulmamalıdır ki erişim seviyesi Level 2 olarak ayarlandığında MCU ömrü boyunca flash belleğe içeriden veya dışarıdan herhangi bir müdahale ile koruma seviyesi değiştirilemez.

3.5.1.6. CLI Flash Adresine Atlama Komutu

Sistem boot modundayken CLI uygulamasını kullanarak herhangi bir adrese atlama yapılabilmektedir. Flash adresine atlama komutu test amaçlı kullanılabildiği gibi birden fazla uygulamayı flash belleğe yükleyip çoklu program kullanabilme imkani sağlamaktadır. Burada dikkat edilmesi gereken nokta, çoklu program yükleme durumunda sistem boot aşamasına geçmeden uygulamaya geçiş yapılmak istenirse, otomatik olarak son yüklenen program çalıştırılacaktır. Ek olarak sistem boot modundayken başlangıç adresine atlama yapılarak resetlenebilmektedir. Şekil 3.9'da CLI uygulamasının flash adresine atlama komutunun cevabı gösterilmektedir. Şekil 3.10'da önyükleyici yazılımının belirtilen adrese atladıktan sonra onay mesajı gösterilmektedir.

```
Command == > BL_GO_TO_ADDR

Please enter 4 bytes go address in hex:0x20000021

CRC : SUCCESS Len : 1

Address Status :  0x1
```

Şekil 3.9: CLI RDP Flash Adresine Atlama Komutu Cevabı

```
BL_INFO: bootloader Go specified Address command
BL_INFO: Checksum control...
BL_INFO: checksum success !!
BL_INFO: GO addr: 0x20000021
BL_INFO:  jumping to 0x20000021 address!
```

Şekil 3.10: Önyükleyici RDP Flash Adresine Atlama Komutu Cevabı

3.5.1.7. CLI FLash Silme Komutu

Şekil 3.11'de flash silme komutunun cevabı gösterilmektedir.

```
Type the command code here :6

Command == > BL_FLASH_ERASE

Enter sector number(1-12 or 0xff) here :8

Enter number of sectors to erase(max 12) here :1

CRC : SUCCESS Len : 1

Erase Status: Success  Code: FLASH_HAL_OK
```

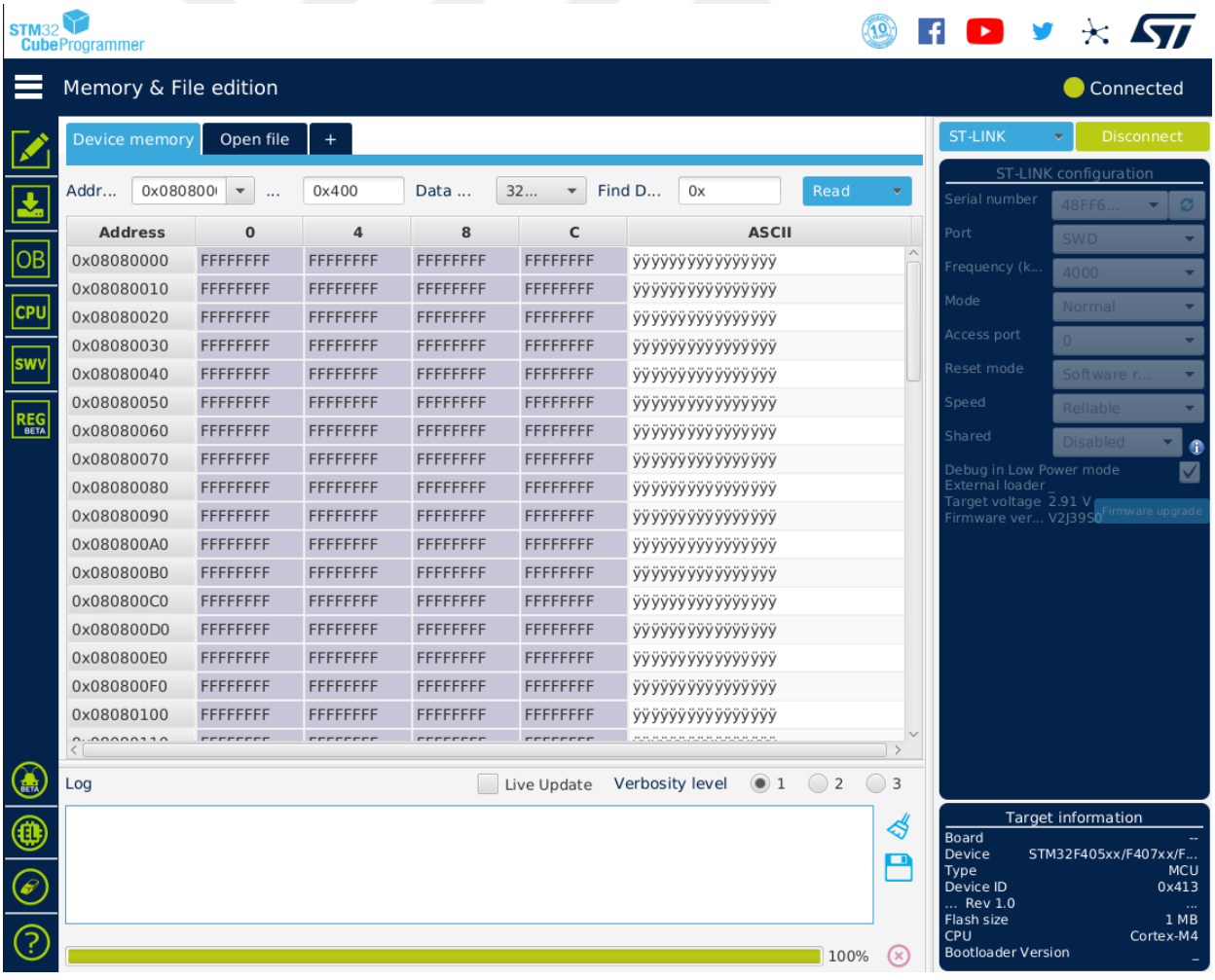
Şekil 3.11: CLI Flash Silme Komut Cevabı

Şekil 3.12'de önyükleyici yazılımının flash silme komutunu aldıktan sonra onay mesajı gösterilmektedir.

```
BL_INFO: Flash erase status: 0x4
BL_INFO: Bootloader handle flash erase cmd
BL_INFO: Checksum control...
BL_INFO: checksum success !!
BL_INFO: Initial_sector : 8  no_ofsectors: 1
BL_INFO: Flash erase status: 0
```

Şekil 3.12: Önyükleyici Flash Silme Komut Cevabı

CLI uygulamasıyla sistem içerisinde yer alan flash bellek sektörleri manuel olarak silinebilmektedir. Normal şartlarda sanal blok adreslerinde yeterli yer bulunmaması durumunda sektör silme işlemi otomatik olarak yapılmaktadır. Fakat kullanıcıya sektör silme opsiyonu CLI uygulaması ile sağlanmaktadır. Şekil 3.13'te vendor firma tarafından geliştirilmiş STM32CubeProgrammer programı ile silinen sektörler gösterilmektedir.



Şekil 3.13: STM32CubeProgrammer ile Silinen Sektörün Okunması

3.5.1.8. CLI Flash Yazma Komutu

CLI flash yazma komutu, önyükleyici yazılımı ile sisteme yeni uygulamaların flash belleğe yazılmasını sağlamaktadır. Flash yazma komutu verimli önyükleyici implementasyonunun en önemli operasyonlarından birini oluşturmaktadır. Flash yazma komutu ile daha önce belirlenen en uygun flash bellek adresine yazma işlemi yapılabileceği gibi manuel olarak belirtilecek herhangi bir bellek adresinden itibaren de yazma işlemi yapılabilmektedir. Manuel flash bellek adresi belirtilmesi durumunda geçersiz adres denetimi ve bellek adres uygunluğu kullanıcıya bırakılmıştır. Bu yüzden manuel adresleme işleminde operasyonun başarılı olması kullanıcı sorumluluğuna bırakılmıştır.

Flash yazma komutunun çalıştırılabilmesi için daha önceden CLI dosya yükleme adres yolunda geçerli bir yürütülebilir dosya bulunması gerekmektedir. Geçerli yürütülebilir dosya üretime komutu "CLI Uygulama Yükleme İçin Hazırlık Komutu" konu başlığında detaylı olarak anlatılmaktadır. Ek olarak, geçerli yürütülebilir bir dosya, CLI komutu kullanılmadan manuel olarak üretilebilir. Bu durumda geçerli yürütülebilir dosyayı CLI dosya yükleme adres yoluna kopyaladıktan sonra flash yazma komutu çalıştırılarak uygulama yükleme işlemi gerçekleştirilebilir. Şekil 3.14'te yürütülebilir dosya üretilmediği ve flash yazma komutu gönderildiği durumlarda CLI uygulamasının komut cevabı gösterilmektedir.

```
   Type the command code here :7

   Command == > BL_MEM_WRITE

.bin file is not configured yet. Please create bin file with command 14 first!

   Press any key to continue  :
```

Şekil 3.14: Yürütülebilir Dosya Olmadığı Zaman CLI Flash Yazma Komut Cevabı

```
  Type the command code here :7

  Command == > BL_MEM_WRITE

File 0 : User_App7.bin

  Please enter file number here :0

  File size is 99476 bytes

  Number of sectors required 7

  Enter the memory write address here (auto or 0x08000000 - 0x080FFFFF) : auto
```

Şekil 3.15: Yürütülebilir Dosya Hazır Olduğu Zaman CLI Flash Yazma Komut Cevabı

Şekil 3.15'te CLI dosya yükleme adres yolunda geçerli yürütülebilir dosya olduğu ve flash yazma komutu gönderildiği durumlarda CLI uygulamasının komut cevabı gösterilmektedir.

CLI uygulamasına flash yazma komutu gönderildiğinde, CLI dosya yükleme adres yolunda geçerli olan ".bin" uzantılı tüm yürütülebilir dosyalar listelenmektedir. Kullanıcı bu aşamada yüklemek istediği yürütülebilir dosyanın dosya numarasını input bilgisi olarak CLI uygulamasına girmesi beklenmektedir. Dosya numarası girildikten sonra dosya boyut bilgisi ve gerekli olan sanal bellek blok sayısı CLI ekranında gösterilmektedir. Yükleme öncesi son aşama olarak kullanıcıdan, yürütülebilir dosyanın yazılacağı flash bellek başlangıç adresi beklenmektedir. Kullanıcı bu aşamada manuel olarak geçerli bir adres veya "auto" anahtar sözcüğünü belirterek yazma işlemini başlatabilmektedir. Anahtar sözcük olarak belirtilen "auto", yükleme adresi için verimli önyükleyici yazılımının belirleyeceği en uygun flash adresini hesaplama işlemini başlatmak için kullanılmaktadır. Kullanıcı "auto" anahtar sözcüğünü yazdıktan sonra en uygun flash adresi otomatik olarak belirlenip yazma işlemi başlatılmaktadır.

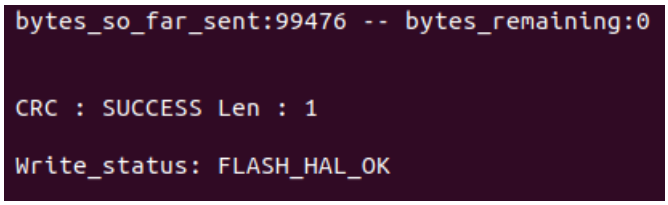
Şekil 3.16'da flash yazma işlemi başlarken önyükleyici uygulamasının bilgi mesajı gösterilmektedir. Önyükleyici bilgi mesajının içerisinde flash yazma komutunun başarılı bir şekilde alındığı, gelen mesaj paketinin CRC kontrolünün sonucu, sanal adres bloklarından en uygun adres bloğunun indeksi, yükleme için seçilen sanal blok adresinin flash bellekte karşılık gelen fiziksel adresi ve seçilen adresin geçerli bir adres olup olmadığı bilgileri bulunmaktadır.

```
BL_INFO: bootloader_handle_mem_write_cmd
BL_INFO: Checksum control...
BL_INFO: checksum success !!
BL_INFO: Best index = 10
BL_INFO:  mem write address : 0x8038000
BL_INFO:  valid mem write address
```

Şekil 3.16: Flash Yazma İşlemi Başlarken Önyükleyici Cevabı

Şekil 3.17'de flash yazma işlemi tamamlandıktan sonra CLI ekranına gönderilen bilgi mesajları gösterilmektedir. CLI ekranında gösterilen bilgi mesajlarında başarılı bir şekilde gönderilen mesajların toplam bayt sayısı, gönderilmek üzere bekleyen kalan mesajların toplam bayt sayısı, önyükleyici uygulamasından mesajın doğru bir şekilde alındığını gösteren CRC denetiminin sonucu ve flash yazma işleminin sonucu gösterilmektedir.

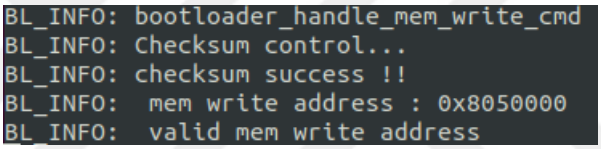
Şekil 3.17'de görüldüğü üzere mesajın başarılı bir şekilde gönderilememesi durumunda



Şekil 3.17: Flash Yazma İşlemi Tamamlandıktan Sonra CLI Cevabı

veya mesajın başarılı gönderilip flash yazma işleminin başarısız olması durumunda, yürütülebilir dosya gönderimi başarısız olmaktadır ve önyükleyici yazılımı flash yazma durumundan komut bekleme durumuna geçmektedir. Başarısızlık durumunda önyükleyici ve CLI ekranlarında hata mesajları gösterilmektedir.

Şekil 3.18'de flash yazma işlemi tamamlandıktan sonra önyükleyici uygulaması kullanıcıya bilgi amaçlı mesaj göndermektedir.



Şekil 3.18: Flash Yazma İşlemi Tamamlandıktan Sonra Önyükleyici Cevabı

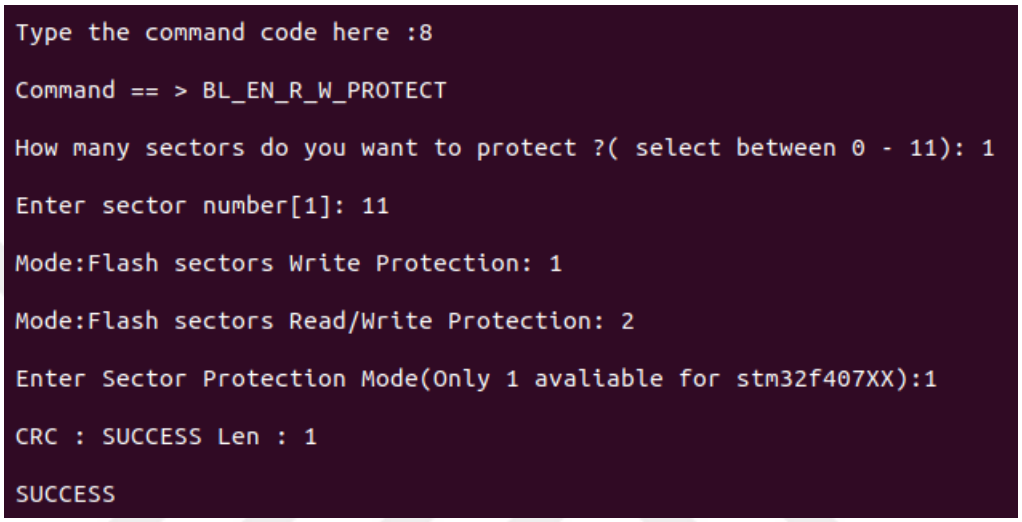
Şekil 3.18'de gösterilmekte olan önyükleyici bilgi mesajının içeriği Şekil 3.16'da gösterilen bilgi mesajı ile çok benzemektedir. Önyükleyici başlangıç ve bitiş mesajları arasındaki fark, başlangıç aşamasında bir defaya mahsus olmak üzere flash başlangıç adresi belirlenir ve sanal bellekte hesaplanan blok indeksi mesaj olarak gönderilmektedir. Daha sonraki önyükleyici mesajlarında sanal blok indeksi hesaplanmaz ve en son yazılan flash adresine eklenerek yazılmaya devam edilir. Bu işlem yürütülebilir dosyanın gönderimi tamamlanana kadar devam eder.

3.5.1.9. CLI Flash Sektör Okuma/Yazma Erişim Aktif Etme Komutu

Flash bellek erişiminin kontrolü güvenlik açısından oldukça önemli bir konudur. İmplementasyonun yapıldığı mikrodenetleyicide olduğu gibi birçok üretici firma flash bellek koruması için kullanıcılara çeşitli seçenekler sunmaktadır. Bu seçenekler genellikle okuma koruması(RDP), yazma koruması ve okuma/yazma koruması olarak sınıflandırılmaktadır. [26] Tez kapsamında kullanılan mikrodenetleyici serisinin sadece yazma koruma modu bulunmaktadır. Fakat aynı mikrodenetleyici ailesinin bazı serilerinde her iki koruma modu bulunduğu ve register adreslerinin her iki mod ile ilgili işlemlere uygun olduğu

için CLI uygulamasında sembolik olarak her iki mod bilgisi de gösterilmektedir. Ek olarak CLI uygulamasında STM32F407XX serisinde sadece yazma koruma modu olduğu belirtilmektedir.

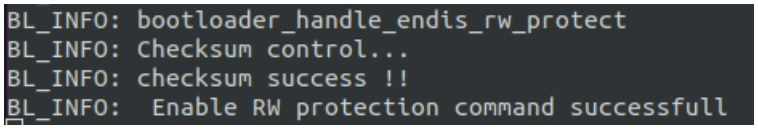
Şekil 3.19'da CLI uygulamasının flash sektör okuma/yazma erişim aktif etme komut cevabı gösterilmektedir.



Şekil 3.19: CLI Flash Sektör Okuma/Yazma Erişim Aktif Etme Komut Cevabı

Şekil 3.19'da örnek olarak mikrodenetleyici flash belleğinin 11. sektörünün yazma koruma modu aktif edilmiştir. CLI uygulamasında birden fazla sektörün koruma modu aynı anda değiştirilebilmektedir. Bu yüzden CLI uygulamasında ilk olarak kaç tane sektör üzerinde işlem yapılacağı kullanıcıdan istenir. İşlem yapılacak sektör sayısı belirlendikten sonra okuma veya yazma koruma modu seçimi yapılır. Şekil 3.19'da CLI ekranında implementasyon yapılan mikrodenetleyici serisinde sadece yazma koruma modu aktif edilebileceği belirtilmiştir. Son olarak koruma modu işleminin başarı durumu CLI ekranında gösterilmektedir.

Şekil 3.20'de önyükleyici flash sektör okuma/yazma erişim aktif etme komut cevabı gösterilmektedir.



Şekil 3.20: Önyükleyici Flash Sektör Okuma/Yazma Erişim Aktif Etme Komut Cevabı

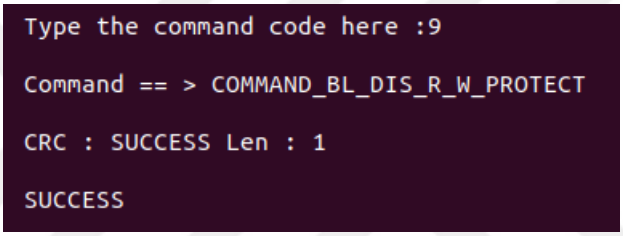
Şekil 3.20'de gösterilen önyükleyici bilgi mesajında CLI uygulamasından gelen sorgu

mesajının tanımı, CRC başarı durumu ve komut başarı durumu bulunmaktadır.

3.5.1.10. CLI Flash Sektör Okuma/Yazma Erişim Pasif Etme Komutu

CLI ugulaması ile daha önce yazma koruma modu aktif edilmiş flash bellek sektörlerinin koruma modları pasif edilebilmektedir. Verimli önyükleyici tasarımında yazma koruma modu sektör bazlı pasif edilememektedir. Yazma koruma modunu pasif etme komutu gönderildiğinde, daha önce koruma modu aktif edilmiş tüm sektörlerin koruma modu pasif edilmektedir.

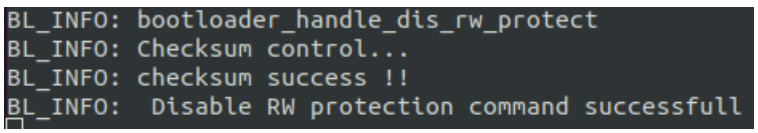
Şekil 3.21'de CLI uygulamasının flash sektör okuma/yazma erişim pasif etme komut cevabı gösterilmektedir.



Şekil 3.21: CLI Flash Sektör Okuma/Yazma Erişim Pasif Etme Komut Cevabı

Şekil 3.21'de gösterilen CLI uygulamasının bilgi mesajında sırasıyla CLI komut mesaj tanımı, CRC başarı durumu ve flash sektör okuma/yazma erişim pasif etme komutunun başarı durumu gösterilmektedir.

Şekil 3.22'de önyükleyici uygulamasının flash sektör okuma/yazma erişim pasif etme komut cevabı gösterilmektedir.



Şekil 3.22: Önyükleyici Flash Sektör Okuma/Yazma Erişim Pasif Etme Komut Cevabı

Şekil 3.22'de gösterilen önyükleyici bilgi mesajında CLI uygulamasından gelen sorgu mesajının tanımı, CRC başarı durumu ve komut başarı durumu bulunmaktadır.

3.5.1.11. CLI Flash Sektör Okuma/Yazma Erişim Durum Komutu

CLI uygulamasıyla mikrodenetleyicinin flash bellek sektörlerinin okuma/yazma erişim durumları sorgulanabilmektedir. Şekil 3.23'te CLI uygulaması flash sektör okuma/yazma

erişim durum sorgu komut cevabı gösterilmektedir.



Şekil 3.23: CLI Flash Sektör Okuma/Yazma Erişim Durum Sorgu Komut Cevabı - 1

Şekil 3.23'te gösterilen CLI uygulamasının bilgi mesajında sırasıyla CLI komut mesaj tanımı, CRC başarı durumu ve flash sektör okuma/yazma erişim durumları sektör bazlı gösterilmektedir. Şekil 3.23'te daha önce "3.5.2.9." konu başlığında örnek olarak koruma modu aktif edilen 11. sektörün erişim durumu aktif olarak gösterilmektedir.

Şekil 3.24'te önyükleyici uygulaması flash sektör okuma/yazma erişim durum sorgu komutunun cevabı gösterilmektedir. Şekil 3.24'te gösterilen "nWRP" değişkeni, sektör bazlı koruma durumlarını ifade eden 11 bitlik bir değerdir. Her bir bit indeksi karşılık gelen sektör değerinin koruma mod durumunu ifade etmektedir. Koruma modu aktif durumunun bit değer karşılığı 0, pasif durumunun bit değer karşılığı 1'dir. Örnek olarak sadece 11. sektörün koruma modu aktif edildiği için nWRP değeri "b01111111111" binary değerine karşılık

gelen “0x7FF” değeri gösterilmektedir.

```
BL_INFO: bootloader_handle_read_sector_protection_status
BL_INFO: Checksum control...
BL_INFO: checksum success !!
BL_INFO: nWRP status: 0x7ff
```

Şekil 3.24: Önyükleyici Sektör Okuma/Yazma Erişim Durum Sorgu Komut Cevabı

Şekil 3.25’te daha önce “3.5.2.10.” konu başlığında örnek olarak koruma modu pasif etme komutu gönderildikten sonra flash sektör okuma/yazma erişim durumları sektör bazlı gösterilmektedir ve tüm flash bellek sektörlerinin okuma/yazma erişim durumlarının pasif edildiği görülmektedir.

```
 Type the command code here :10

 Command == > COMMAND_BL_READ_SECTOR_P_STATUS

 CRC : SUCCESS Len : 2

 Sector Status :  0xfff

==================================

Sector                            Protection

==================================

Flash protection mode :       Write Protection


 Sector0                           No protection

 Sector1                           No protection

 Sector2                           No protection

 Sector3                           No protection

 Sector4                           No protection

 Sector5                           No protection

 Sector6                           No protection

 Sector7                           No protection

 Sector8                           No protection

 Sector9                           No protection

 Sector10                           No protection

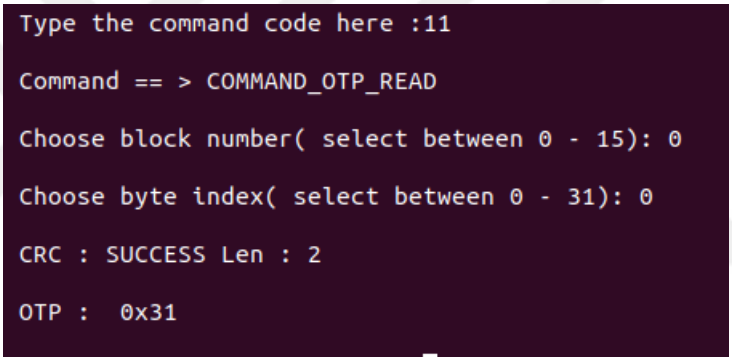
 Sector11                           No protection
```

Şekil 3.25: CLI Flash Sektör Okuma/Yazma Erişim Durum Sorgu Komut Cevabı - 2

3.5.1.12. CLI OTP Okuma Komutu

İmplementasyonun yapıldığı mikrodenetleyici flash belleğinde tek seferlik programlanabilir(OTP) alan için 512 baytlık yer ayırılmıştır. OTP alanı sıradan flash bellek alanlarından farklı olarak yapısı gereği kilitlendikten sonra sadece bir kere programlanabilir ve cihaz ömrü boyunca bu değer değiştirilemez. OTP alanı karşılık gelen OTP veri bloğunu kilitlemek için kullanılan 16 ek bayt içerir[24]. Dolayısıyla OTP için toplamda 528 bayt veri alanı ayrılmıştır. OTP alanı, 32 baytlık 16 OTP veri bloğuna ve 16 baytlık bir kilit OTP bloğuna bölünmüştür.

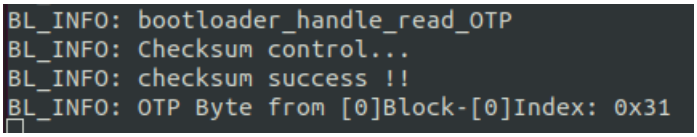
Şekil 3.26'da örnek göstermek amacıyla CLI uygulamasını kullanarak sıfırıncı OTP bloğunun sıfırıncı bayt indeksindeki OTP değeri sorgu komutu ve cevabı gösterilmektedir.



Şekil 3.26: CLI Örnek OTP Sorgu Komut Cevabı

Şekil 3.26'da okunmuş olarak gösterilen "0x31" OTP değeri program başlangıcında örnek olarak atanmış sıfırıncı OTP bloğunun sıfırıncı bayt indeksindeki OTP değeridir.

Şekil 3.27'de önyükleyici uygulamasının örnek OTP sorgu komutu ve cevabı gösterilmektedir.



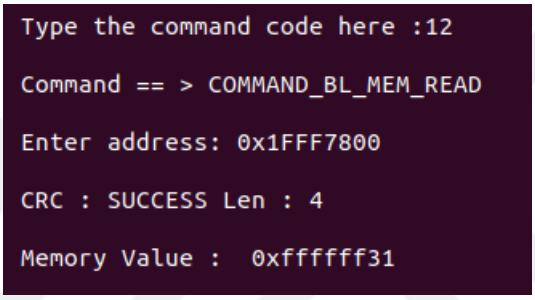
Şekil 3.27: Önyükleyici Örnek OTP Sorgu Komut Cevabı

Şekil 3.27'de gösterilen önyükleyici bilgi mesajında OTP değeri, ait olduğu blok ve indeks değerleriyle beraber gösterilmektedir.

3.5.1.13. CLI Flash Adres Okuma Komutu

CLI uygulamasını kullanarak herhangi bir bellek adresindeki değer sorgulanabilmektedir. CLI uygulamasında girilen adres değerinin geçerliliği kullanıcının sorumluluğuna bırakılmıştır. Adres denetiminin bu implementasyonda yapılmamış olmasının nedeni daha jenerik bir yapı sunmak ve esnekliği arttırmak amacıyladır. Kullanılacak mikrodenetleyicinin bellek organizasyonuna göre flash adres okuma komutu özelleştirilebilir halde bırakılmıştır.

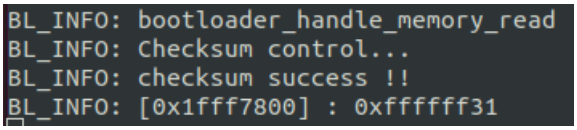
Şekil 3.28'de CLI uygulaması üzerinden STM32F407VGT6 mikrodenetleyicisinin OTP alanının başlangıç adresinde ki 32 bitlik değer sorgulanmıştır.



Şekil 3.28: CLI Flash Bellek Adres Okuma Sorgu Komut Cevabı

Şekil 3.28'de okunan değerin adresi daha önceki konularda bahsedilen OTP alanının değerini doğrulamak için verilmiştir. Şekil 3.28'de görüleceği üzere sıfırıncı OTP bloğunun sıfırıncı bayt indeksindeki OTP değeri "0x31" olup, geri kalan 3 bayt işlem görmediği için "0xFF" olarak okunmaktadır.

Şekil 3.29'da önyükleyici uygulamasının flash bellek adres okuma sorgu komut cevabı gösterilmektedir.



Şekil 3.29: Önyükleyici Flash Bellek Adres Okuma Sorgu Komut Cevabı

Şekil 3.29'da gösterilen önyükleyici bilgi mesajında okunan flash bellek adresi ve bu adreste bulunan 4 baytlık hex değeri beraber gösterilmektedir.

3.5.1.14. CLI Sanal Bellek Blok Bilgilerini Okuma Komutu

İmplementasyonda kullanılan mikrodenetleyicinin flash belleği üç temel parçaya bölünmüştür. Bu parçalardan biri sanal bellek blok bilgilerinin bulunduğu eeprom emülasyon bölümüdür. Sanal bellek blok bilgileri mevcut uygulamanın flash bellek başlangıç adresi, boyutu, P/E sayısı ve P/E sayısı geçmiş değerlerinden oluşmaktadır. CLI uygulaması ile eeprom emülasyonu ile flash belleğe kaydedilmiş olan sanal bellek blok bilgilerine ulaşabiliriz. Şekil 3.30'da sanal bellek blok bilgilerinden mevcut uygulamanın flash bellek başlangıç adresi ve boyutu bilgileri gösterilmektedir.

```
Type the command code here :13

Command == > COMMAND_BL_GET_BOOT_INFO

CRC : SUCCESS Len : 188

currentMemoryBlock :  0x8054000

Last payload len :  0x3b28
```

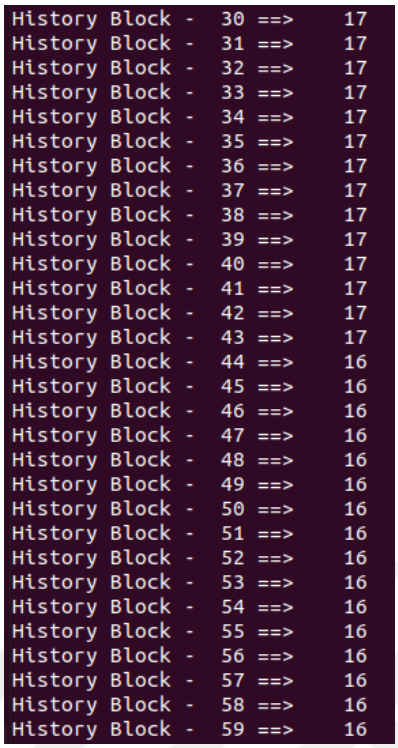
Şekil 3.30: CLI Sanal Bellek Blok Bilgilerini Okuma Sorgu Cevabı - 1

Şekil 3.31'de ilk 30 sanal bloğunun P/E sayısı geçmiş değerleri gösterilmektedir.

```
History Block -   0 ==>    18
History Block -   1 ==>    18
History Block -   2 ==>    18
History Block -   3 ==>    18
History Block -   4 ==>    18
History Block -   5 ==>    18
History Block -   6 ==>    18
History Block -   7 ==>    18
History Block -   8 ==>    18
History Block -   9 ==>    18
History Block -  10 ==>    18
History Block -  11 ==>    18
History Block -  12 ==>    18
History Block -  13 ==>    18
History Block -  14 ==>    18
History Block -  15 ==>    19
History Block -  16 ==>    19
History Block -  17 ==>    19
History Block -  18 ==>    18
History Block -  19 ==>    18
History Block -  20 ==>    18
History Block -  21 ==>    18
History Block -  22 ==>    18
History Block -  23 ==>    18
History Block -  24 ==>    18
History Block -  25 ==>    18
History Block -  26 ==>    18
History Block -  27 ==>    18
History Block -  28 ==>    17
History Block -  29 ==>    17
```

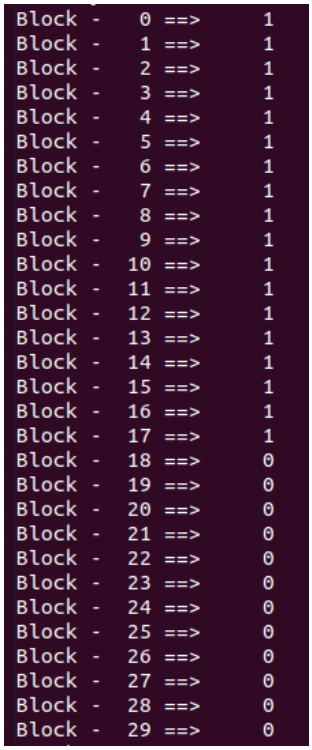
Şekil 3.31: CLI Sanal Bellek Blok Bilgilerini Okuma Sorgu Cevabı - 2

Şekil 3.32'de son 30 sanal bloğunun P/E sayısı geçmiş değerleri gösterilmektedir.



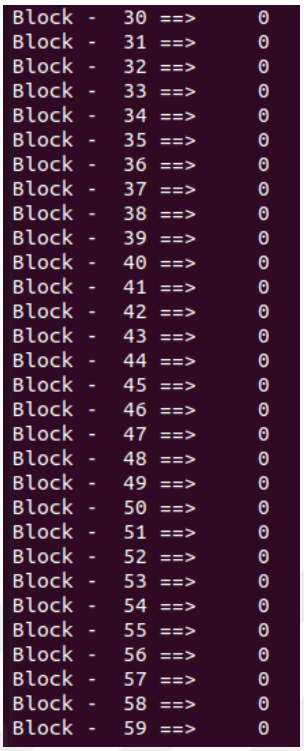
Şekil 3.32: CLI Sanal Bellek Blok Bilgilerini Okuma Sorgu Cevabı - 3

Şekil 3.33'de ilk 30 sanal bloğunun P/E sayısı gösterilmektedir.



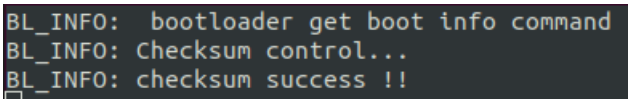
Şekil 3.33: CLI Sanal Bellek Blok Bilgilerini Okuma Sorgu Cevabı - 4

Şekil 3.34'de son 30 sanal bloğunun P/E sayısı gösterilmektedir.



Şekil 3.34: CLI Sanal Bellek Blok Bilgilerini Okuma Sorgu Cevabı - 5

Şekil 3.35'de gösterilen önyükleyici bilgi mesajında CLI uygulamasından gelen sorgu mesajının tanımı ve CRC başarı durumu gösterilmektedir.

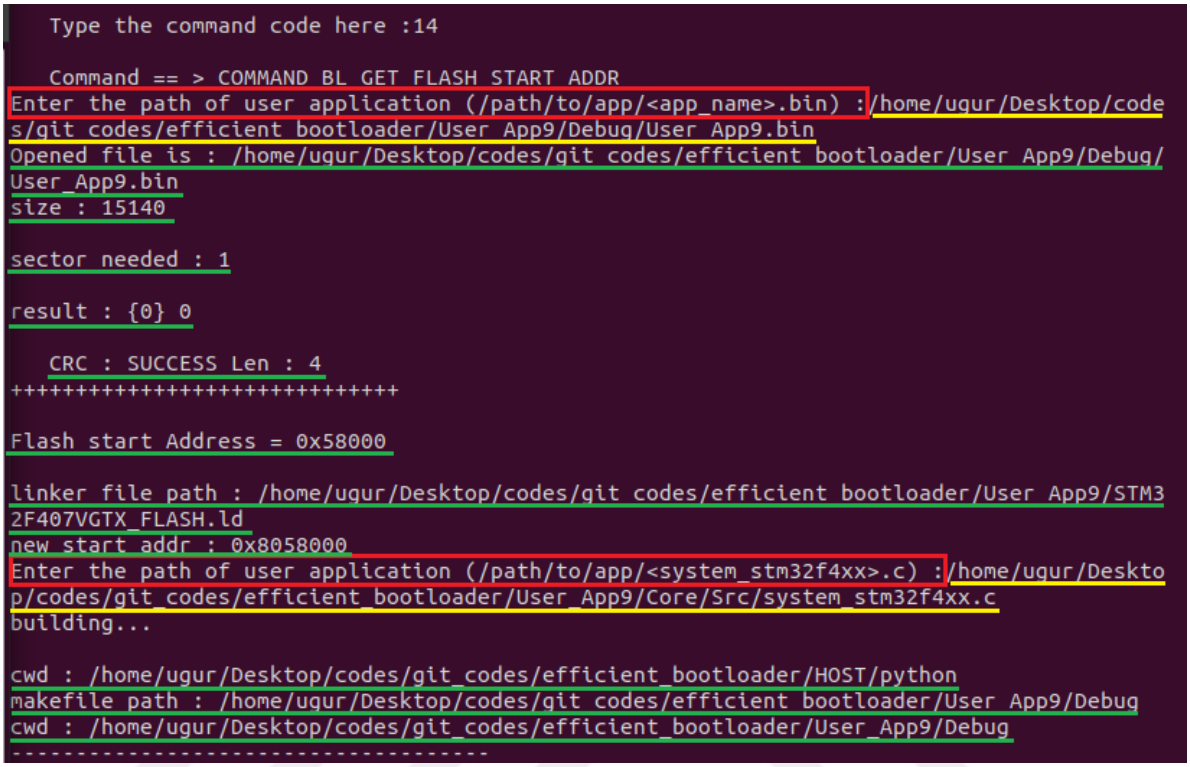


Şekil 3.35: Önyükleyici Sanal Bellek Blok Bilgilerini Okuma Sorgu Cevabı

3.5.1.15. CLI Yürütülebilir Dosya Hazırlık Komutu

Verimli önyükleyici tasarımı yapılırken kolay kullanılabilirlik ve ek donanım ihtiyacı olmaması amaçlanmıştır. Bu doğrultuda kullanıcıya mümkün olan en az alt yapı değişikliği ön görülerek tasarım yapılmıştır. CLI uygulamasının en temel amaçlarından biri de flash bellekte dinamik adresleme doğrultusunda yürütülebilir bir dosyayı proje yolundan otomatik olarak üretmek olmuştur. Tasarımın ilk aşamalarında linker file üzerindeki flash adres değişiklikleri, sistem dosyasında ki vector tablosu adres ofset değer ataması ve değişikliklerin uygulanabilmesi için tekrar derleme işlemi manuel olarak yapılmaktaydı. Daha sonra bu manuel işlemlerin hem proje dosyasında değişikliklere sebep olması, hem hata yapma olasılığının daha fazla artması, hem de geliştiriciler için bir zaman kaybı oluşturduğu için yürütülebilir dosya üretim prosesinin tamamı otomatize edilmiştir.

Şekil 3.36'da CLI uygulaması üzerinden yürütülebilir dosya hazırlık komutu gönderildikten sonra uygulama cevabının bir kısmı gösterilmektedir.



Şekil 3.36: CLI Yürütülebilir Dosya Hazırlık Komutu Cevabı - 1

Şekil 3.36'da kırmızı kutu içerisinde gösterilen alan CLI uygulamasının kullanıcıdan istediği bilgileri göstermektedir. İlk kırmızı kutuda kullanıcıdan mevcut projesinin yürütülebilir dosya yolunu girmesi beklenmektedir. İkinci kırmızı kutuda kullanıcıdan sistem dosyasının yolunun girilmesi beklenmektedir.

Altı sarı çizgiyle çizilmiş olan kısımlar manuel olarak gönderilen ilgili girdi değerleridir. Altı sarı çizgiyle çizilmiş olan ilk kısım mevcut projenin yürütülebilir dosya yolunu ve ikinci sarı çizgili kısım mevcut projenin sistem dosya yolunu göstermektedir.

Altı yeşil çizgiyle çizilmiş olan kısımlar CLI uygulamasının çıktılarını göstermektedir. CLI çıktılarında sırasıyla yürütülebilir dosya yolu bilgisi, yürütülebilir dosyanın yüklenebilmesi için gereken sanal bellek blok sayısı, operasyon başarı durumu, CRC sonucu, en uygun flash bellek başlangıç adresi, linker file dosya yolu bilgisi, en uygun flash bellek başlangıç adresinin fiziksel adres karşılığı, mevcut çalışma dizini ve Makefile dosya yolu belirtilmiştir.

Şekil 3.37'de CLI uygulaması üzerinden yürütülebilir dosya hazırlık komutu gönderildikten

sonra uygulama cevabının kalan diğer kısmı gösterilmektedir.

```
arm-none-eabi-gcc "../Core/Src/system_stm32f4xx.c" -mcpu=cortex-m4 -std=gnu11 -g3 -DDEBUG
 -DUSE_HAL_DRIVER -DSTM32F407xx -c -I../Core/Inc -I../Drivers/STM32F4xx_HAL_Driver/Inc -I
../Drivers/STM32F4xx_HAL_Driver/Inc/Legacy -I../Drivers/CMSIS/Device/ST/STM32F4xx/Include
 -I../Drivers/CMSIS/Include -O0 -ffunction-sections -fdata-sections -Wall -fstack-usage -
MMD -MP -MF"Core/Src/system_stm32f4xx.d" -MT"Core/Src/system_stm32f4xx.o" --specs=nano.sp
ecs -mfpu=fpv4-sp-d16 -mfloat-abi=hard -mthumb -o "Core/Src/system_stm32f4xx.o"
arm-none-eabi-gcc -o "User_App9.elf" @"objects.list"   -mcpu=cortex-m4 -T"/home/ugur/Desk
top/codes/git_codes/efficient_bootloader/User_App9/STM32F407VGTX_FLASH.ld" --specs=nosys.
specs -Wl,-Map="User_App9.map" -Wl,--gc-sections -static --specs=nano.specs -mfpu=fpv4-sp
-d16 -mfloat-abi=hard -mthumb -Wl,--start-group -lc -lm -Wl,--end-group
Finished building target: User_App9.elf

arm-none-eabi-size   User_App9.elf
   text    data     bss     dec     hex filename
  14964     120    1800   16884    41f4 User_App9.elf
Finished building: default.size.stdout

arm-none-eabi-objdump -h -S  User_App9.elf  > "User_App9.list"
Finished building: User_App9.list

arm-none-eabi-objcopy  -O ihex  User_App9.elf  "User_App9.hex"
Finished building: User_App9.hex

arm-none-eabi-objcopy  -O binary  User_App9.elf  "User_App9.bin"
Finished building: User_App9.bin

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

cwd : /home/ugur/Desktop/codes/git_codes/efficient_bootloader/HOST/python
------------------------------------

Filename is : User_App9.bin
finish linker process

restoring files...
```

Şekil 3.37: CLI Yürütülebilir Dosya Hazırlık Komutu Cevabı - 2

Şekil 3.37'de gösterilen uygulama çıktısında özet olarak yeni konfigüre edilmiş proje dosyalarının tekrar derlenmesi, yeni yürütülebilir dosya üretimi ve yürütülebilir dosya üretim prosesinin tamamlanmasından sonra, değiştirilen dosyaların eski haline getirilmesi işlemleri gösterilmektedir.

Şekil 3.39'da gösterilen önyükleyici bilgi mesajında CLI uygulamasından gelen sorgu mesajının tanımı ve CRC başarı durumu gösterilmektedir.

```
BL_INFO:  bootloader process output file command
BL_INFO: Checksum control...
BL_INFO: checksum success !!
BL_INFO: Flash address start must 360448 bytes far away from origin in linker file.
```

Şekil 3.38: Önyükleyici Yürütülebilir Dosya Hazırlık Komutu Cevabı

3.5.2. Kullanıcı Uygulama Projesinden Yürütülebilir Dosya Üretimi

Verimli önyükleyici yazılımından bağımsız olarak, kullanıcı uygulamalarının düzgün çalışması için yürütülebilir dosyaların belirli bir işlemle oluşturulması gerekir. Mevcut önyükleyici tasarım yöntemleri genellikle hazır yürütülebilir dosyanın flash belleğe yazılması sürecine odaklanmaktadır. Çünkü yürütülebilir dosyanın kullanıcı tarafından kurulacak sistemin gereksinimlerini karşılayacak şekilde üretilmesi beklenir. Klasik önyükleyici tasarımlarında flash belleğe yazım adresleri statik olduğu için vendor firmaların geliştiricilere sunduğu örnek projelerde veya IDE'lerde yürütülebilir dosyalar üretilirken standart bir yol izlenir. [16][14][27][6][28] Burada standart yol ile kullanıcı manuel olarak değiştirmediği sürece linker dosyalarında ki RAM ve FLASH adreslerinin otomatik olarak değiştirilmemesi ve vector offset adreslerinin sabit kalması kastedilmektedir.

Bu projedeki en önemli noktalardan biri, yürütülebilir dosyanın çalışma zamanında proje dosyalarından dinamik olarak elde edilmesidir. CLI-önyükleyici iletişimi aracılığıyla kullanıcı uygulama projesinden yürütülebilir dosya oluşturma algoritması aşağıda gösterilmektedir.

Algorithm 1: Yürütülebilir Dosya Üretimi

Require: Yürütülebilir Dosya Yolu, Sistem Dosya Yolu
Ensure: UART Haberleşmesi

Yürütülebilir dosya boyutunu hesapla
Gerekli sanal bellek blok sayısını hesapla
Linker ve sistem dosyalarını yedekle
while $timeout \neq 0$ **do**
    $timeout \leftarrow timeout - 1$;
    if $yeniAdresOfset$ **then**
        $yerelAdresOfset = yeniAdresOfset$
    end if
end while
if $timeout = 0$ **then**
    return $fail()$
end if
Linker dosyasının belirlenen adrese göre güncelle
Vektör tablosunun base ofset değerinin belirlenen adrese göre güncelle
Projenin GNU Make ile yeniden derle
Yürütülebilir dosyanın CLI yükleme dosya yoluna kopyala
Linker ve sistem dosyalarının eski haline getir

Algoritma 1'de belirtildiği üzere, yürütülebilir dosyanın üretilebilmesi, sistem ve linker dosyalarının dosya yolları kullanıcıdan istenir. Ardından yürütülebilir dosyanın boyutu hesaplanarak flash bellekte ne kadar sanal blok alanı kullanılacağı hesaplanır. Gerekli blok sayısı verimli önyükleyici yazılımına gönderilir ve verimli önyükleyici yazılımından "$yerelAdresOfset$" olarak belirtilen optimum flash bellek başlangıç adresi beklenir. Verimli önyükleyici yazılımının optimum başlangıç adresini nasıl belirlediği bir sonraki konuda açıklanacaktır. Verimli önyükleyici, optimum flash yazma adresi belirledikten sonra CLI uygulamasına bu adresi gönderir. Bir sonraki aşamaya geçmeden önce proje dosyasında bulunan linker ve sistem dosyalarının yedeği alınır.

CLI uygulaması, adres bilgilerini aldıktan sonra linker dosyasındaki flash adresleri yeniden düzenler. Linker dosyasını düzenledikten sonra, projenin sistem dosyasındaki vektör tablosu ofset değerlerini güncellenir. CLI uygulaması dosya düzenlemeyi bitirdikten sonra projeyi GNU make aracıyla derler ve yürütülebilir bir dosya üretir. Oluşturulan yürütülebilir dosyanın bir kopyasını özel bir konuma(CLI dosya yükleme yolu) kopyalar. Son olarak, CLI uygulaması projede değişen dosyaları daha önce yedeklediği dosyalar ile eski haline getirerek prosedürü tamamlar. Bu aşamadan sonra kullanıcı flash yazma komutu gönderirse, CLI uygulaması dosya yükleme yolunda bulunan yürütülebilir dosyaları listeler ve kullanıcının listeden seçim yapması beklenir. Yürütülebilir dosya seçimi yapıldıktan sonra seri port üzerinden yürütülebilir dosya gönderilmeye başlanır. Verimli önyükleyici yazılımı seri port üzerinden gönderilen dosyanın bütünlüğünü CRC kontrolü ile sağlar. CRC kontrolünden geçen her paket flash belleğe yazılmaya devam eder. Dosya aktarımı sırasında herhangi bir pakette CRC hatası veya timeout hatası alınırsa, işlem tamamlanması beklenmeden iptal edilir. Dosya aktarımı tamamlandıktan sonra sistem boot modda kalmaya devam eder ve kullanıcının CLI uygulamasından göndereceği komutları bekler. Eğer kullanıcı boot moda geçiş butonuna basmadan sistemi yeniden başlatırsa veya CLI ekranından farklı bir flash adresine atlama komutu verirse boot modundan çıkılır. Kullanıcı yürütülebilir dosya oluşturma yapmadan yazma komutu gönderirse, CLI uygulaması "Yürütülebilir dosya henüz yapılandırılmadı. Lütfen önce bin dosyasını oluşturun" uyarısını verir.

Algoritma 1 ile kurulan yapıda yeni uygulamanın yüklenmesi için gereken linker ve sistem dosyalarındaki değişiklikler CLI uygulaması ile otomatik olarak arka planda yapılmaktadır. Değişikliklerin yürütülebilir dosyada uygulanabilmesi için GNU Make aracı ile yeniden

derleme işlemi CLI uygulaması ile otomatize edilmiştir. Böylece geliştiriciler proje dosyalarında herhangi bir değişiklik yapmadan ve yürütülebilir dosya üretiminden sonra da dosyaları değiştirmeden yüklenecek dosya üretimini gerçekleştirmiş olur.

3.5.3. Bellek Dayanıklılığını Arttırmak İçin Optimum Yazma Adresinin Belirlenmesi

Optimum flash bellek dayanıklılığını elde etmek için en önemli noktalardan biri de çalışma zamanında en uygun bellek yazma adreslerinin belirlenmesidir. Flash belleğin dayanıklılığını artırmak için her yeni program yükleme aşamasında en uygun flash bellek başlangıç adresi belirlenir. Yeni uygulama boyutu ve flash bellekteki boş sanal bloklar dikkate alınarak, yazma işleminin başlayacağı flash bellek başlangıç adresi hesaplanır. Hesaplanan flash bellek adresi daha sonra CLI uygulamasına gönderilir. Flash bellek başlangıç adresi hesaplanırken, eeprom emülasyon yazılımında kullanılan wear leveling algoritmasına benzer bir algoritma kullanılır. Bu yöntem ile flash belleğin kullanıcı uygulamalarına ayrılan tüm bloklarının mümkün olduğunca homojen bir bellek yazım dağılımına sahip olması amaçlanmaktadır.Flash bellek dayanıklılığının arttırılması için en uygun adresi belirleme algoritması, Algoritma 2'de gösterilmiştir.

Algorithm 2: Bellek Dayanıklılığını Arttırmak İçin Optimum Yazma Adresinin Belirlenmesi

Require: P/E Sayacı, P/E Sayacı Geçmiş Değerleri, Sanal Bellek Blok Sayısı
Ensure: UART Haberleşmesi

```
minSanalBellekBlokAdresi bul
if sıralıAdresKontrol(minSanalBellekBlokAdresi) then
    minSanalBellekBlokAdresi indeksini hesapla
else
    for i ← blokIndeks_başlangıç to blokIndeks_bitiş do
        i ← i + 1
        if sanalBlokSayacı_i ≠ 0 then
            Sektör Sil
            sanalBlokSayacı_i = 0
        end if
    end for
end if
Mevcut sanal bellek blok adresini güncelle
P/E sayacını güncelle
P/E sayacı geçmiş değerlerini güncelle
```

CLI tarafından yeni bir uygulama yükleme talebi alındığında, blokların mevcut P/E sayısı kontrol edilir ve "*minSanalBellekBlokAdresi*" olarak belirtilen minimum silme sayısına

sahip blok adresi elde edilir. Ardından "*minSanalBellekBlokAdresi*" ile flash belleğin tüm sanal blokları kontrol edilerek yeterli sayıda sıralı ve boş sanal blok olup olmadığı kontrol edilir. Bellekte yeterli ve sıralı boş blok varsa, en uygun bellek yazma adresi olarak "*minSanalBellekBlokAdresi*" değeri geri döndürülür. Bellekte yeterli ve/veya sıralı boş blok yoksa, silme sayısı 0'dan büyük olan(daha önce yazma işlemine maruz kalmış) bloklara ait tüm sektörler silinir. Her işlemden sonra blokların bellek blok adresleri, P/E sayısı ve P/E sayısı geçmiş değerleri güncellenir. Güncellenen değerler, eeprom emülasyonu için ayrılmış sektörlerde kaydedilir.

Verimli önyükleyici yazılımı mikrodenetleyiciye yüklendikten sonra P/E sayısı geçmiş değerleri istatistiksel amaçlar için tutulur ve eeprom emülasyon yazılımının bulunduğu flash bellek sektörleri silinmeyene kadar saklanır. Sanal bellek bloklarına yazma işlemi uygulandığında ilgili bloğun P/E sayısı ve P/E sayısı geçmiş değeri bir arttırılır. Aynı sanal bellek bloğu silindiğinde P/E sayısı sıfırlanırken, P/E sayısı geçmiş değeri asla sıfırlanmaz. Özetle P/E sayısı geçmiş değerleri sadece yazma işlemi yapıldığında arttırılan değerlerdir. Bellek bloklarının P/E sayısı geçmiş değerleri istenildiği zaman CLI üzerinden sorgulanabilmektedir.

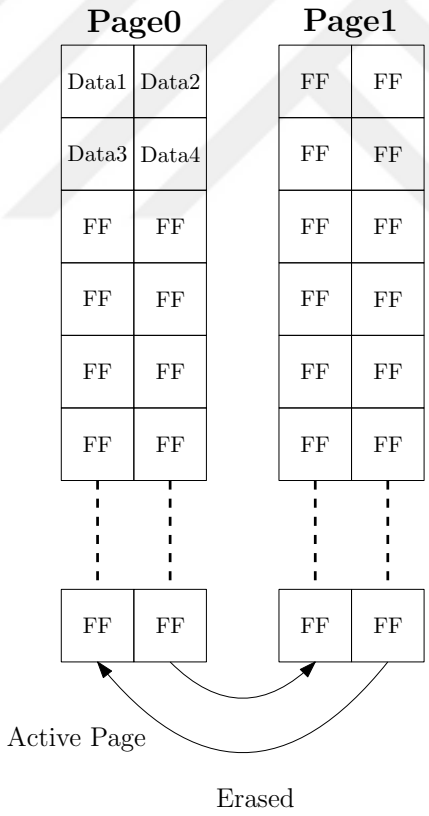
3.5.4. Sanal Bellek Blok Bilgilerinin Saklanması

Boot bilgilerinin saklanması, okunması ve güncellenmesi için flash bellekte iki sektör kullanılmıştır. Bu sektörler kullanım kolaylığı ve bellek dayanım verimliliğini arttırmak amacıyla eeprom emülasyonu yönetmiyle sisteme entegre edilmiştir. Verimli önyükleyici yazılımından ve uygulama yüklenme alanlarından bağımsız olarak çalışmaktadır. Bu nedenle verimli önyükleyici veya uygulamaların bulunduğu flash alanlarındaki eylemler eeprom emülasyonunun yapıldığı flash bellek bloklarını etkilememektedir. İnternal flash bellekte bir bölgenin harici bir flash bellek donanımı gibi davranmasından dolayı eeprom emülasyonu ifadesi kullanılmaktadır.

Eeprom okuma işlevi, her sistem resetlemesinden sonra eeprom emülasyon programı ile boot bilgilerini okur. Sistem resetleme durumları dışında, CLI programı eeprom bilgilerini talep ederse, eeprom okuma işlevi, emülasyon için ayrılmış bölgeden boot bilgilerini okur ve bu bilgileri seri haberleşme portu üzerinden paylaşır. Böylece sistem boot bilgileri, çalışma sırasında herhangi bir zamanda kullanılabilir durumdadır. Boot bilgileri yeni uygulama

yüklendikten sonra veya flash bellek silindikten sonra eeprom yazma fonksiyonları ile güncellenir.

Eeprom emülasyonu, boot bilgilerini okumak ve güncellemek için kullanılmaktadır. Verimli önyükleyici yazılımında eeprom bilgilerinin saklanabilmesi ve bellek dayanım kapasitesinin arttırılması için iki sektör(2 numaralı sektör Page0, 3 numaralı sektör Page1 olarak) kullanılmaktadır. Eeprom emülasyonunda dinamik wear leveling algoritması kullanılmaktadır. Dinamik wear leveling algoritmasının eeprom emülasyonunda kullanımı şu şekilde gerçekleşir: Page0 sayfası dolduğunda, cihaz Page1 sayfasına yazmaya başlar. Page0 sayfası, Page1 sayfasında yedeklenir ve ardından silinir.[1] Page1 sayfası dolduğunda, daha önce silinmiş olan Page0 sayfasında yedeklenir ve ardından Page1 sayfası silinir. Eeprom yazma-silme döngüsü ve sayfalardaki verilerin takas edilmesi bu şekilde devam eder. Wear leveling algoritmasını kullanan iki sayfalı sayfa takas şeması Şekil 3.3'de gösterilmektedir.



Şekil 3.39: İki Sayfalı Sayfa Değiştirme Şeması[1]

Şekil 3.3'de gösterildiği üzere wear leveling algoritması ile kullanılan her sektör için mevcut tüm yazma döngülerinin eşit şekilde kullanılmasını sağlanır.[1]

4. BULGULAR

4.1. VERİMLİ ÖNYÜKLEYİCİ PERFORMANS ANALİZİ

Verimli önyükleyici tasarımında flash bellek toplamda 3 ana bölüme ayrılmıştır. Bölümlerden ilki, diğer bölümlere göre en az değişiklik gösterecek olan önyükleyici yazılımının bulunduğu ilk iki flash bellek sektörlerini temsil eder. İkinci bölüm, flash belleğe yüklenecek uygulamaların sanal bellek bloklarını temsil eder. Üçüncü bölümde, flash bellek başlangıç adresi, en son yüklenen uygulamanın boyutu, sanal bellek bloklarının P/E sayısı ve P/E sayısı geçmiş değerlerinin kaydedildiği boot bilgileri bulunur. Aşağıdaki alt başlıklarda özelleştirilmiş bu 3 ana bölümle ilgili performans bilgilerine yer verilmektedir.

4.1.1. Verimli Önyükleyici Yazılımının Bulunduğu Sektörlerin Performansı

Verimli önyükleyici tasarımı uygulaması için 16 kilobaytlık “Sektör 0” ve “Sektör 1” sektörleri kullanılmaktadır. Önyükleyici yazılımının çok nadir değişeceği veya hiç değişmeyeceği düşünülerek herhangi bir algoritma geliştirilmemiştir. Bu nedenle, önyükleyici yazılımı her güncellendiğinde, yeni önyükleyici yazılımı, flash belleğin sıfır adresinden itibaren programlamaya başlayacaktır. Her güncelleme için, önce “Sektör 0” ve “Sektör 1” sektörleri silinecek ve daha sonra programlanacaktır. Sonuç olarak önyükleyici yazılımının yüklendiği sektörlerin dayanıklılığı mikrodenetleyicinin flash belleğinin dayanıklılığı ile aynıdır ve bu uygulama için minimum değer 10000 P/E döngü sayısıdır. Önyükleyici yazılımının güncelleme sıklığının, kullanıcı uygulama yazılımının güncelleme sıklığına göre çok düşük olacağı varsayılarak, dayanıklılığı artıracak bir algoritma geliştirilmemiştir.

4.1.2. Programlanabilir Sanal Bellek Bloklarının Bulunduğu Sektörlerin Performansı

STM32F407VG mikrodenetleyici flash belleğinin son 7 sektörü olan “Sektör 4,5,6,7,8,9,10,11” sektörleri kullanıcı uygulamalarını yüklemek için kullanılmaktadır. Toplam programlanabilir alan, 16 kilobaytlık 60 sanal bellek bloğuna bölünmüş ve flash bellek boyutu olarak 960 kilobayta denk gelmektedir. Verimli önyükleyici tasarımında

flash bellek dayanıklılığı, yüklenecek uygulama yazılımının boyutuna göre değişiklik göstermektedir.

Klasik önyükleyici tasarımında belli büyüklükteki bir uygulamayı mikrodenetleyicinin flash belleğine kaç kez yükleyebileceğimizi bulmak için sadece dayanıklılık kapasitesini bilmemiz yeterlidir. Çünkü klasik önyükleyici tasarımlarında yeni bir kullanıcı uygulamasının yüklenmeye başlayacağı flash bellekteki yazma başlangıç adresi sabittir ve her uygulama yükleme öncesi silinmeye tabi tutulacaktır. Bu nedenle, her uygulama yükleme aşamasında flash bellek dayanıklılık kapasitesi silme işleminden dolayı azalacaktır. Verimli önyükleyici tasarımında ise durum farklıdır. Verimli önyükleyici tasarımında yüklenecek yeni uygulamanın boyutu, dayanıklılık kapasitesini doğrudan etkilemektedir. Çünkü verimli önyükleyici flash bellek sektörleri, yeni yüklenecek uygulama esnasında bellekte yeterli sayıda boş ve sıralı sanal bellek blokları olduğu sürece silinmeyecektir. Bu nedenle, silme işlemi gerçekleşene kadar dayanıklılık kapasitesi azalmayacaktır. Aşağıda dayanıklılık kapasitesi ile ilgili bazı denklemler gösterilmektedir.

$$PK_{VÖ} = \frac{SektorBoyutu \times DayanıklılıkKapasitesi}{UygulamaBoyutu} \tag{4.1}$$

$$PK_{KÖ} = DayanıklılıkKapasitesi \tag{4.2}$$

$$DAO = \frac{PK_{VÖ}}{PK_{KÖ}} \tag{4.3}$$

Toplam flash bellek sektör boyutu ve flash bellek dayanıklılığının çarpımı toplam bellek kapasitesini belirler. Toplam bellek kapasitesini uygulama boyutuna bölerek mikrodenetleyiciyi kaç kez programlayabileceğimizi gösteren programlama kapasitesi(PK) değerini elde ederiz. PK, verimli önyükleyici($VÖ$) için Denklem(4.1) ve klasik önyükleyici($KÖ$) için Denklem(4.2) kullanılarak hesaplanır. KÖ tasarımlarının bellek

dayanıklılığı üretici firmaların belirlediği bellek dayanıklılığına eşit olduğu için Denklem(4.2) elde edilmiştir. Klasik önyükleyici tasarımına kıyasla verimli önyükleyici tasarımının programlama kapasitesindeki dayanıklılık artış oranı(DAO) Denklem(4.3) ile hesaplanır.

Tablo 4.1'de farklı uygulama boyutlarına göre $PK_{KÖ}$ ve $PK_{VÖ}$ için flash bellek dayanıklılığının değişimleri gösterilmektedir. İlk iki sektör verimli önyükleyici yazılımına ve sonraki 2 sektör eeprom emülasyon yazılımına ayrılmıştır. Yeni uygulama yüklemek için kullanılabilir sektör sayısı Sektör 4 ile Sektör 11 arasında ki sektörleri kapsar. Mevcut sektörlerin doluluk oranlarına göre uygulama boyutu en fazla 960 KB'a kadar ulaşabilir.

Tablo 4.1: Farklı Uygulama Boyutlarına Göre STM32F407VGT6 Flash Bellek Dayanıklılık Verimliliği

Farklı Uygulama Boyutları İçin Programlama Kapasiteleri						
Dayanıklılık Kapasitesi	**Uygun Sektör Numarası**	**Uygun Sektör Boyutu(KB)**	**Uygulama Boyutu(KB)**	**$PK_{KÖ}$**	**$PK_{VÖ}$**	**DAO**
10000	Sektör 4-11	960	8	10000	1200000	120
10000	Sektör 4-11	960	16	10000	600000	60
10000	Sektör 4-11	960	32	10000	300000	30
10000	Sektör 4-11	960	64	10000	150000	15
10000	Sektör 4-11	960	128	10000	75000	7,5
10000	Sektör 4-11	960	960	10000	10000	1

Tablo 4.1'e bakıldığında, kullanıcı uygulamalarının boyutu arttıkça sektörleri silmeden programlayabilme sayısının azaldığı, dolayısıyla bellek dayanıklılığının azaldığı görülmektedir.

Yüklenecek uygulamaların toplam boyutunun toplam sektör boyutu olan 960 kilobayt'tan küçük olduğu durumlarda flash bellek dayanıklılığı artırılabilmektedir. Sanal bellek bloklarının tamamını dolduracak kadar büyük bir uygulama yüklenmesi durumunda(944-960 kilobayt boyutlu uygulamalar), her bir yükleme esnasında flash bellek sektörleri silineceği için flash bellek dayanıklılığı artmayacaktır. Verimli önyükleyici tasarımı klasik önyükleyici tasarımına göre bellek dayanıklılığını azaltmayacak şekilde tasarlanmıştır.

Flash bellek sektörlerinin farklı boyutlarda olduğu durumlarda sektörlerin silinme süreleri

birbirinden farklıdır. STM32F407VGT6 mikrodenetleyicisinin flash sektörlerinin silinme süreleri **Tablo 4.2:**'de gösterilmektedir.

Tablo 4.2: STM32F407VGT6 Flash Bellek Sektör Silinme Süreleri[2]

Koşullar	**Sektörler**	**Boyut(KB)**	**SS(ms)**	**DWWT(us)**
$TA = 0$ to $+40^{\circ}C, VDD = .3V, VPP = 8.5V$	Sector 0	16	230	16
	Sektör 1	16	230	16
	Sektör 2	16	230	16
	Sektör 3	16	230	16
	Sektör 4	64	490	16
	Sektör 5	128	875	16
	Sektör 6	128	875	16
	Sektör 7	128	875	16
	Sektör 8	128	875	16
	Sektör 9	128	875	16
	Sektör 10	128	875	16
	Sektör 11	128	875	16

STM32F407VGT6 mikrodenetleyicisinin double word yazma süresi(DWWT) tüm sektörlerde eşittir.[2] Tablo 4.2'de gösterildiği gibi, silme süresi(SS) sektör boyutuyla doğru orantılıdır. Diğer bir deyişle, sektör boyutu ne kadar büyük olursa, SS o kadar uzun sürer. Önyükleyici ve eeprom emülasyon yazılımı için ayrılan sektörlerin toplam boyutu 64 KB'dır. Kalan tüm sektörler(960 kilobayt) uygulama yükleme alanı olarak ayrılmıştır.

$$WT_{VÖ}(us) = \frac{UygulamaBoyutu(byte) \times DWWT(us)}{4} \tag{4.4}$$

$$WT_{KÖ}(us) = SS(us) + WT_{VÖ}(us) \tag{4.5}$$

Yazma süresi(YS), bir kullanıcı uygulamasının flash belleğe yazılması için geçen süredir. Yazma süresi uygulama boyutu ile flash belleğe veri yazmak için geçen sürenin çarpımından elde edilir.

YS değeri, Denklem(4.4)'de verimli önyükleyici tasarımı için ve Denklem(4.5)'de klasik önyükleyici tasarımları için ayrı ayrı hesaplanmıştır. DWWT değeri verimli önyükleyici ve klasik önyükleyici tasarımlarında eşittir fakat klasik önyükleyici tasarımlarında ek olarak her yükleme öncesi silme işlemi gerekli olduğu için silme işleminde harcanan süre, yazma süresine eklenmektedir. Farklı önyükleyici tasarımları için farklı YS değerlerinin olması tasarımın verimliliği açısından kritik bir önem arz etmektedir. YS değerlerinin uzun olmasının fiziksel ortamda ki karşılığı yeni bir uygulama yüklemek için uzun süre beklemek olarak ifade edilebilir. Bu durum, geliştiriciler açısından komfor alanını bozan bir süreç olduğu gibi endüstriyel üretim ortamlarında iş gücü ve zaman kaybı olarak değerlendirilebilir.

Tablo 4.3, verimli önyükleyici ve klasik önyükleyici tasarımları için farklı boyutlardaki uygulamaların sektörlere bir kez yazılma sürelerini göstermektedir.

Tablo 4.3: Farklı Boyutlardaki Uygulamalar İçin Flash Bellek Yazılma Süreleri

Sektörler	**Sektör Boyutu (KB)**	**SS (ms)**	**DWWT (us)**	**YS(us)**									
				UygulamaBoyutu$_{VÖ}$(KB)					**UygulamaBoyutu$_{KÖ}$(KB)**				
				8	**16**	**32**	**64**	**128**	**8**	**16**	**32**	**64**	**128**
Sektör 4	64	490	16	32768	65536	131072	262144	YB	522768	555536	621072	752144	YB
Sektör 5	128	875	16	32768	65536	131072	262144	524288	907768	940536	1006072	1137144	1399288
Sektör 6	128	875	16	32768	65536	131072	262144	524288	907768	940536	1006072	1137144	1399288
Sektör 7	128	875	16	32768	65536	131072	262144	524288	907768	940536	1006072	1137144	1399288
Sektör 8	128	875	16	32768	65536	131072	262144	524288	907768	940536	1006072	1137144	1399288
Sektör 9	128	875	16	32768	65536	131072	262144	524288	907768	940536	1006072	1137144	1399288
Sektör 10	128	875	16	32768	65536	131072	262144	524288	907768	940536	1006072	1137144	1399288
Sektör 11	128	875	16	32768	65536	131072	262144	524288	907768	940536	1006072	1137144	1399288

Tablo 4.3'te belirtilen "YB" değerleri "Yetersiz Bellek" anlamına gelmektedir. Sektör bazlı yazma süreleri hesaplanırken sektör boyutundan daha büyük bir uygulama yüklenemeyeceği için YB ifadesi kullanılmıştır.

Tablo 4.3'te sektör numaraları, sektörlerin boyutları, sektörlerin silinme süreleri, sektörlere veri yazma süreleri ve farklı tasarımlar için yazma sürelerinin hesaplandığı değerler gösterilmektedir. Tablo 4.3'te farklı sektörlerin farklı boyutlara sahip olduğu durumlarda verimli önyükleyici ve klasik önyükleyici tasarımlarında yazma sürelerinin değişmediği gösterilmektedir. Yazma süreleri her iki tasarım grubu için eşit olsa da, klasik önyükleyici tasarımlarında yazma süresine ek olarak sektör silme süreleri de dahil edilmektedir.Bu yüzden aynı uygulama boyutundaki verileri flash bellek sektörüne yazmak için harcanan sürelerde oldukça büyük farklar oluşmaktadır.

Denklem(4.6)'de yazma süresi verimlilik formulü gösterilmektedir. Denklem(4.8)'ün amacı verimli önyükleyici tasarımı ile klasik önyükleyici tasarımının DAO açısından verim farkını göstermektir.

$$YSV = \frac{YS_{KÖ}}{YS_{VÖ}} \tag{4.6}$$

Tablo 4.4'te farklı boyutlardaki kullanıcı uygulamaları için verimli önyükleyici ve klasik önyükleyici tasarımlarında flash belleğe veri yazma süresi oranları gösterilmektedir.

Tablo 4.4: Farklı Boyutlardaki Uygulamalar İçin Verimli ve Klasik Önyükleyici Flash Belleklerine Yazılma Süresi Oranları

Sektörler	**YSV**				
	Uygulama Boyutlarına(KB) Göre DAO				
	8	**16**	**32**	**64**	**128**
Sektör 4	15.953	8.476	4.738	2.869	YB
Sektör 5	27.702	14.351	7.675	4.337	2.668
Sektör 6	27.702	14.351	7.675	4.337	2.668
Sektör 7	27.702	14.351	7.675	4.337	2.668
Sektör 8	27.702	14.351	7.675	4.337	2.668
Sektör 9	27.702	14.351	7.675	4.337	2.668
Sektör 10	27.702	14.351	7.675	4.337	2.668
Sektör 11	27.702	14.351	7.675	4.337	2.668

Tablo 4.4'te verimli ve klasik önyükleyici tasarımları arasında oluşan yazma süresi oranları Denklem (4.3) kullanılarak elde edilmiştir.

Uygulama boyutunun artması yazma süresi için harcanan süreyi arttırmaktadır. Verimli önyükleyici tasarımlarında uygulama boyutunun küçük olması yazma süresini azaltır ve bellek dayanıklılığını arttırır. Klasik önyükleyici tasarımlarında ise uygulama boyutunun küçük olması sadece yazma süresini azaltır fakat bu azalış verimli önyükleyici tasarımına göre daha düşük bir oranda kalmaktadır. Bu tasarımsal farktan dolayı uygulama boyutunun artması DAO değerinin azalmasına neden olmaktadır.

Ek olarak, Sektör 4'ün boyutu 64 kilobayt ve tablodaki diğer sektörler 128 kilobayt boyuta sahip olduğu için klasik önyükleyici tasarımlarında Sektör 4'e yazma süresi daha kısa ve DAO daha fazladır.

$$SS_{VÖ} = \lceil \frac{ProgramlamaSayısı \times UygulamaBoyutu(byte)}{UygunSektörBoyutu(byte)} \rceil \quad (4.7)$$

$$TSS(ms) = SM \times SS(ms) \times KullanılanSektörSayısı \quad (4.8)$$

$$TYS(ms) = (ProgramlamaSayısı \times YS(ms)) + TSS(ms) \quad (4.9)$$

Denklemlerde belirtilen “Silme Miktarı(SM)”, P/E sayısını ifade etmektedir.

Denklem(4.7)’da toplam bellek gereksinimi, yüklenecek uygulama boyutu ile uygulama yükleme sayısı çarpılarak elde edilir. Sektörün silinme sayısını bulmak için toplam bellek gereksinimi sektör boyutuna bölünür.

Sektörleri silmek için harcanan toplam silme süresinin(TSS) hesaplanması Denklem(4.8)’de gösterilmiştir. Toplam silme süresini bulmak için öncelikle silinecek alanın hesaplanması gerekmektedir. Silinecek alan, kullanılan sektör sayısı ile bu sektörlerin kaç kere silineceğini çarparak elde edilir. Daha sonra silinecek alan ile sektör bazlı silme süreleri çarpılarak TSS değeri hesaplanır.

Uygulama verilerinin flash belleğe yazılması için harcanan toplam yazma süresi(TYS) Denklem(4.9)’de gösterilmiştir. Daha önce programlanmış bir sektörün yeniden programlanabilmesi için önce silinmesi, daha sonra tekrar programlanması gerektiğini önceki başlıklarda belirtilmişti. Sektörün tekrar silindiği koşullardan yola çıkarak toplam yazma süresini hesaplamak için sektörün kaç kere programlanması gerektiği bilinmelidir. Sektörün programlanma sayısı ile her bir programlama için harcanan sürenin çarpımına toplam silinme süresini ekleyerek TYS değeri elde edilmektedir.

Farklı boyutlara sahip kullanıcı uygulamaları için verimli önyükleyici ve klasik önyükleyici tasarımlarının yazma sürelerinin karşılaştırılması Tablo 4.5’te gösterilmiştir.

Tablo 4.5: Farklı Boyutlarda Kullanıcı Uygulamaları İçin Verimli Ve Klasik Önyükleyici Yazma Süresi Verimliliği

Uygun Sektör Sayısı	**Sektör4 için SS** (ms)	**Sektör 5-11 için SS** (ms)	**DWWT** (us)	**Uygun Sektör Boyutu** (KB)	**Uygulama Boyutu** (KB)	**Program Yükleme Sayısı**	**$SM_{VÖ}$**	**$SM_{KÖ}$**	**$TYS_{VÖ}$** (ms)	**Sektör4 için $TYS_{KÖ}$** (ms)	**Sektör 5-11 için $TYS_{KÖ}$** (ms)
Sektör 4-11	490	875	16	960	8	10000	84	10000	883340	5227680	9077680
Sektör 4-11	490	875	16	960	16	10000	167	10000	1760065	5555360	9405360
Sektör 4-11	490	875	16	960	32	10000	334	10000	3520130	6210720	10060720
Sektör 4-11	490	875	16	960	64	10000	667	10000	7033645	7521440	11371440
Sektör 4-11	490	875	16	960	128	10000	1334	10000	12900040	18892880	13992880
Sektör 4-11	490	875	16	960	960	10000	10000	10000	105471600	105471600	YB

Tablo 4.5:'de elde edilen verilere göre silme miktarı verimli önyükleyici tasarımında klasik önyükleyici tasarımlarına göre daha az veya eşittir. Bu durum verimli önyükleyici tasarımının klasik önyükleyici tasarımlarına göre verimlilik açısından herhangi bir dezavantaj durumu oluşturmadığını gösterdiği gibi uygulama boyutunun programlabilir bellek alanının tamamını kaplayacak boyutta olmadığı tüm senaryolarda daha verimli olduğunu göstermektedir. Verimli önyükleyici tasarımlarının silme miktarında daha verimli olduğu tüm senaryolarda toplam yazma süresinde de bir avantaj söz konusudur. Verimli önyükleyici tasarımlarında yüklenecek program boyutu ne kadar küçük olursa verimlilik her alanda aynı miktarda artmaktadır.

Tablo 4.5:'de klasik önyükleyici için iki farklı TYS verisi eklenmiştir. Klasik önyükleyicilerde TYS sadece uygulama boyutuna bağlı değil, aynı zamanda sektör boyutuna da bağlıdır. Bu yüzden farklı boyutlara sahip sektörler için farklı TYS verileri hesaplanmıştır.

4.1.3. Sanal Bellek Blok Bilgilerinin Bulunduğu Sektörlerin Performansı

Verimli önyükleyici tasarımında, sanal bellek blok bilgilerinin saklanması için STM32F07VGT6 mikrodenetleyici flash belleğinin "Sektör 2" ve "Sektör 3" sektörleri kullanılmaktadır. Bu iki sektörde STMicroelectronics üretici firması tarafından sağlanan eeprom emülasyon kütüphanesi[29] kullanılmıştır. Eeprom emülasyon kütüphane dökümanlarında[1] açıklandığı gibi, flash bellekte yapılacak yazma ve silme işlemleri için wear leveling algoritması kullanılmaktadır. Sanal bellek blok bilgisi, sanal bellek bloklarına yazma sayısı geçmişi, sanal bellek bloklarının mevcut yazma durumu, mevcut bellek blok adresi ve en son yüklenen uygulama boyutu bilgisinden oluşur.

Verimli önyükleyici tasarımında 60 adet sanal bellek bloğu bulunmaktadır. Her bir sanal bloğunun mevcut yazma durumu için 1 bayt veri ayrılmıştır. Dolayısıyla sanal blokların mevcut yazma durumu için toplamda 60 bayt yer ayrılmıştır.

Sanal bellek bloklarına yazma sayısının geçmiş değerlerinin tutulması için her bir bloğa ait 2 baytlık alan ayrılmıştır. Her bir bloğun geçmiş yazma sayısı için 2 bayt alan ayrılmasının sebebi, üretici firmanın flash bellek dayanıklılığı için 10000 P/E döngü sayısını garanti etmesidir. 2 bayt verinin alabileceği maksimum değer 65536 olduğu için üretici firmanın garanti ettiği 10000 P/E döngü sayısına ulaşılabilmektedir. Dolayısıyla, sanal bellek bloklarına yazma sayısı geçmişi için bellekte 120 bayt alan ayrılmıştır.

Mevcut bellek blok adresi, belleğe yüklenen en son uygulamanın başlangıç adresidir. Boot mod seçimi esnasında kullanıcı önyükleyici yerine uygulamadan devam etmeyi seçerse, program akışı otomatik olarak mevcut bellek blok adresinden devam edecektir. Dolayısıyla her yeni uygulama yükleme işlemi esnasında mevcut bellek blok adresi kaydedilir. Mevcut bellek blok adresi 4 bayt adres verisidir ve bu yüzden sanal bellek blok bilgilerinin bulunduğu eeprom alanında 4 bayt alan ayrılmıştır.

Verimli önyükleyici tasarımında uygulama boyutu maksimum 960 KB olabilmektedir. 960 KB değeri bayt türünden 983040 bayt değerine karşılık gelmektedir. Bu yüzden en son yüklenen uygulama boyutu bilgisi için sanal bellek blok bilgilerinin bulunduğu eeprom alanında 4 bayt alan ayrılmıştır. Sonuç olarak verimli önyükleyici tasarımında sanal bellek blok bilgisinin bellekte kapladığı toplam alan 188 bayttır.

Tablo 4.6: STM32F407VGT6 Mikrodenetleyicisinin EEPROM Emülasyon Flash Bellek Dayanıklılık Verimliliği

EEPROM Emülasyonu						
Dayanıklılık	**Uygun Sektörler**	**Sektör Boyutları (bytes)**	**Değişken Sayısı (bytes)**	**SM**	**Programlama Sayısı**	**DAO**
10000	Sektör 2-3	32768	188	10000	1732979	173,297

EEPROM emülasyonunun uygulandığı sektörlerin programlanma sayısı Denklem(4.1) kullanılarak elde edilir ve Tablo 4.6'da gösterilmektedir. Tablo 4.6 da dikkat edilmesi gereken en önemli noktalardan biri eeprom emülasyonunda saklanan verilerin boyutu sektör boyutunun tam bölenlerinden olmamasıdır. Bu tür durumlarda son yazma işlemi eeprom

emülasyonundaki sayfanın sonuna sığmadığı için veriler boş olan sayfaya kopyalanarak yazma işlemine devam edilir. Dolayısıyla sektörün her sonuna gelindiğinde sayfa değişim işlemi gerçekleştiği için sektör boyutundan kaydedilecek veri boyutu bir kereliğe mahsus olmak üzere çıkarılır. Daha sonra Denklem(4.1)'de yeni hesaplanan sektör boyutu yerine konulur ve programlama sayısı elde edilir.

Flash belleğin dayanıklılığı, en çok silme döngüsüne maruz kalan sektör tarafından belirlenir(Zincirin zayıf halkasına benzetilebilir). Verimli önyükleyici tasarımı esnasında eeprom emülasyon için ayrılan sektörlerin flash bellek dayanıklılığını etkilememesi amaçlanmıştır. Çünkü verimli önyükleyici tasarımının amacı, geleneksel önyükleyici tasarımlarından daha fazla kullanıcı uygulamasının programlanma kapasitesine ulaşmaktır. Tasarım sürecinde eeprom emülasyon uygulamasının flash bellek dayanıklılığı açısından verimliliği düşürmemesi için iki farklı konu ele alınmıştır. Bu konulardan biri kaydedilecek programın boyutu, diğeri ise sektörün boyutudur.

Yüklenecek uygulamanın boyutunun büyük olması sektörlerin daha kısa sürede dolmasına neden olarak programlama kapasitesinin düşmesine neden olacaktır. Aynı şekilde, çok küçük bir sektör boyutu da benzer bir sonuca yol açacaktır. Bu durumların dışında, eeprom emülasyonu için seçilen sektörlerin çok büyük olması durumunda, eeprom alanının programlanma kapasitesini arttıracak olsa da, kullanıcı uygulamaları için ayrılan sektör boyutunu azaltacaktır. Bu nedenle, verimli önyükleyici tasarımı aşamasında, eeprom emülasyonu için kullanılan sektörlerin boyutu ve sayısı ile eeprom emülasyonu uygulamasının boyutu mümkün olduğunca az tutulmuştur.

4.1.4. Verimli Önyükleyici İle Flash Bellek Bozulmasını Tahmin Etme

Verimli bootloader tasarımının sanal bellek blok bilgilerinin tutulduğu alanda P/E sayısı geçmiş değerleri her güncelleme sonrasında kaydedilmektedir. P/E sayısı geçmiş değerleri sanal bellek bloklarının yazılma ve silinme sayısını tutmaktadır. Üretici firmaların mikrodenetleyicilerin içerisinde bulunan flash bellekler için belirlediği dayanıklılık değeri ile P/E sayısı geçmiş değerleri karşılaştırılarak bellek dayanıklılık ömrü üzerinde tahmin yapmak mümkün hale gelmiştir. P/E sayısı geçmiş değerleri direk olarak dayanıklılık değerine karşılık gelmektedir. Dolayısıyla üretici firmaların bellek dayanıklılık değerine bakılarak, P/E sayısı geçmiş değerinin limit değerlere yaklaşması durumunda kullanıcılara

önceden uyarı vermek mümkündür. Uygulama yazılımına bağlı olarak her sistem başlagıcında veya çalışma zamanında bellek sağlık kontrolü yapılabilmektedir. Böylece olası hasar veya kayıplar önceden tahmin edilebilmektedir.

5. TARTIŞMA VE SONUÇ

Mikrodenetleyicilerdeki flash belleklerin yazma-silme kapasiteleri, flash belleğin dayanıklılığını belirler. Flash belleğe veri yazarken her bir bit, programlama ile 1'den 0 değerine değiştirilebilir, ancak 0'dan 1 değerine değiştirilemez. Flash bellek silme işlemi sektörel veya kütlesel olarak yapılabilir. Klasik önyükleyici tasarımlarında flash bellek dayanıklılığı ve programlama esnasında gerçekleşen flash silme süreleri yüklenecek programın boyutundan bağımsızdır. Ancak, yüklenecek uygulamanın boyutu, optimize edilmiş verimli bir önyükleyici tasarımı ile flash bellek dayanıklılığını ve programlama döngü süresini doğrudan etkiler. Deneysel çalışmalar veya endüstriyel ürünlerin test aşaması gibi mikrodenetleyicilerin kısa aralıklarla programlandığı çalışmalar göz önüne alınarak, flash bellek tutma sürelerinden ziyade, flash bellek dayanıklılığının arttırılması daha ön plana çıkarılmıştır. Özellikle pandemi ile başlayan yarı iletken içeren ürünlerin üretim ve tedarik süreçlerinde yaşanılan aksaklıklar, test ve deneysel süreçlerde kullanılan mikrodenetleyicilerin ömür sürelerinin arttırılması amacı bu araştırmaya ilham kaynağı olmuştur. Mevcut mikrodenetleyici üreticilerinin kullanıcılara sunduğu önyükleyici yazılımı ve örnek uygulamalar flash belleğin sabit bir adresinden başlayacak şekilde yapılandırıldığı için aynı adrese yapılan silme döngüleri flash belleklerin kullanım ömrünü kısaltmaktadır. Wear leveling algoritması ve türevleri üretilerek elde edilen bu çalışma sonucunda, verimli bir önyükleyici tasarımı ile yazma-silme döngülerinin sayısı ve bu işlemler için harcanan süreler optimize edilerek harici bir donanıma ihtiyaç duymadan flash bellek depolama sistemlerinin dayanıklılığı arttırılmıştır. Çalışma sonucunda bellek yazma sürelerinin kısaltılması, gereksiz bellek silme işlemlerinin azaltılması ve bunun neticesinde dayanıklılığın arttırılmasına ek olarak verimli önyükleyici tasarımı sayesinde bellekte dayanıklılığa bağlı oluşabilecek bozulmalar önceden tahmin edilebilir hale gelmiştir.

İlerleyen zamanlarda tasarımın geliştirilmesi için yapılacak çalışmalarda, implementasyon için farklı mikrodenetleyici ailelerini kapsayan ve daha jenerik bir önyükleyici tasarlanabilir. Bare metal sistemler yerine işletim sistemi kullanan sistemler için özel önyükleyici tasarımları üzerinde çalışılabilir. Sanal bellek blok boyutları üzerinde yapılacak optimizasyon çalışmaları ile flash bellek dayanıklılığı daha fazla arttırılabilir. Ek olarak flash

belleğe yeni yüklenecek uygulamalar için yürütülebilir dosyaların imzalanması, kulllanıcıya önem arz eden alanların şifrelenmesi ve olası tehlikeli durumlarda uygulamanın eski sürüme geçirilmesi seçenekleri endüstriyel ihtiyaçları karşılamak amacıyla geliştirilebilir.

KAYNAKLAR

[1]. STMicroelectronics, "An3969, application note, eeprom emulation in stm32f40x/stm32f41x microcontrollers," vol. 2,3, pp. 15–17, 2011.

[2]. STMicroelectronics, "Ds8626, stm32f407vg datasheet," 2021.

[3]. H. Yue-li and X. Bing, "Design of an embedded on-chip debug support module of a mcu," in *Conference on High Density Microsystem Design and Packaging and Component Failure Analysis, 2006. HDP'06.*, pp. 5–8, 2006.

[4]. S. R. Pillay and R. K. Rao, "Circuits and methods for debugging an embedded processor and systems using the same," Sept. 20 2005. US Patent 6,948,098.

[5]. Z. Kazemi, A. Papadimitriou, D. Hely, M. Fazcli, and V. Beroulle, "Hardware security evaluation platform for mcu-based connected devices: Application to healthcare iot," in *2018 IEEE 3rd International Verification and Security Workshop (IVSW)*, pp. 87–92, 2018.

[6]. X. Li, Y. Wang, X. Tang, Y. Hu, D. Li, J. Gan, W. Feng, and L. He, "Design and verification of mcu chip bootloader," in *2020 IEEE 9th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)*, vol. 9, pp. 395–402, 2020.

[7]. N. Wen, W.-m. Yang, J.-h. Peng, and X.-g. WANG, "Design of embedded system bootloader based on mcu," *Microelectronics & Computer*, vol. 35, no. 3, pp. 79–82, 2018.

[8]. Z. Xia, D. S. Kim, N. Jeong, Y.-G. Kim, J.-H. Kim, K.-H. Lee, Y.-K. Park, C. Chung, H. Lee, and J. Han, "Comprehensive modeling of nand flash memory reliability: Endurance and data retention," in *2012 IEEE International Reliability Physics Symposium (IRPS)*, pp. MY.5.1–MY.5.4, 2012.

[9]. L. Shi, K. Wu, M. Zhao, C. J. Xue, D. Liu, and E. H.-M. Sha, "Retention trimming for lifetime improvement of flash memory storage systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 1, pp. 58–71, 2016.

[10]. Y. Cai, Y. Luo, E. F. Haratsch, K. Mai, and O. Mutlu, "Data retention in mlc nand flash memory: Characterization, optimization, and recovery," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, pp. 551–563, 2015.

[11]. Infineon, "An98521, wear leveling application notes, 001-98521 rev.e," 2021. Last accessed 19 Mar 2021.

[12]. S. F. Barrett and D. J. Pack, *Microcontrollers*, pp. 295–322. Boston, MA: Birkhäuser Boston, 2005.

[13]. H. Hidaka. 2018.

[14]. S. T. Sun, "Design and implementation of partial firmware upgrade," 2019.

[15]. Y. Kang, J. Chen, and B. Li, "Generic bootloader architecture based on automatic update mechanism," in *2018 IEEE 3rd International Conference on Signal and Image Processing (ICSIP)*, pp. 586–590, 2018.

[16]. B. J., "Bootloader design for microcontrollers in embedded systems," 2015. Last accessed 09 June 2022.

[17]. R. L. B. R. Guillen O., Nisarga B., "Crypto bootloader secure in field firmware updates for ultra low power mcus," 2015.

[18]. C. Sha and Z.-Y. Lin, "Design optimization and implementation of bootloader in embedded system development," in *2015 International Conference on Computer Science and Applications (CSA)*, pp. 151–156, 2015.

[19]. J. Liao, F. Zhang, L. Li, and G. Xiao, "Adaptive wear-leveling in flash-based memory," *IEEE Computer Architecture Letters*, vol. 14, no. 1, pp. 1–4, 2015.

[20]. J. Li, X. Xu, X. Peng, and J. Liao, "Pattern-based write scheduling and read balance-oriented wear-leveling for solid state drivers," in *2019 35th Symposium on Mass Storage Systems and Technologies (MSST)*, pp. 126–133, 2019.

[21]. X. Ye and Z. Zhai, "Cold-warm-hot block wear-leveling algorithm for a nand flash storage system," in *2017 4th International Conference on Systems and Informatics (ICSAI)*, pp. 762–767, 2017.

[22]. Y.-H. Chang, J.-W. Hsieh, and T.-W. Kuo, "Improving flash wear-leveling by proactively moving static data," *IEEE Transactions on Computers*, vol. 59, no. 1, pp. 53–65, 2010.

[23]. Y.-H. Chang, J.-W. Hsieh, and T.-W. Kuo, "Endurance enhancement of flash-memory storage systems: An efficient static wear leveling design," in *Proceedings of the 44th annual Design Automation Conference*, pp. 212–217, 2007.

[24]. STMicroelectronics, "Rm0090, reference manual, stm32f405/415, stm32f407/417, stm32f427/437 and stm32f429/439 advanced arm based 32-bit mcus," vol. 3, no. 3, pp. 74–75, 2021.

[25]. S. Sheng-Ju, "Implementation of cyclic redundancy check in data communication," in *2015 International Conference on Computational Intelligence and Communication Networks (CICN)*, pp. 529–531, 2015.

[26]. STMicroelectronics, "An4701, application note, proprietary code read-out protection on microcontrollers of the stm32f4 series," pp. 1–9, 2016.

[27]. P. T. Karule and P. R. Selokar, "Firmware upgradation of arm7 through communication link: Via uart over can," in *2016 International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES)*, pp. 80–84, 2016.

[28]. J. R. Márquez L, "Can bus bootloader for the stm32f407vg," 2017.

[29]. S. Mischie and R. Pazsitka, “Designing a msp430 bootloader,” in *2019 International Conference on Applied Electronics (AE)*, pp. 1–4, 2019.