İSTANBUL TEKNİK ÜNİVERSİTESİ
ELEKTRİK-ELEKTRONİK FAKÜLTESİ

CRYSTALS-KYBER POST QUANTUM ALGORİTMASININ RISC-V İŞLEMCİ ÜZERİNDE GERÇEKLENMESİ

LİSANS BİTİRME TASARIM PROJESİ

Ahmet ÇELİK

Fatih YILMAZ

Mehmet Anıl KORKMAZ

Berna Örs Yalçın
Uygundur
08.01.2024

ELEKTRONİK VE HABERLEŞME MÜHENDİSLİĞİ BÖLÜMÜ

OCAK, 2024

İSTANBUL TEKNİK ÜNİVERSİTESİ
ELEKTRİK-ELEKTRONİK FAKÜLTESİ

CRYSTALS-KYBER POST QUANTUM ALGORİTMASININ RISC-V İŞLEMCİ ÜZERİNDE GERÇEKLENMESİ

LİSANS BİTİRME TASARIM PROJESİ

Ahmet ÇELİK
040190031

Fatih YILMAZ
040190046

Mehmet Anıl KORKMAZ
040190035

Proje Danışmanı: Prof. Dr. Berna ÖRS YALÇIN

ELEKTRONİK VE HABERLEŞME MÜHENDİSLİĞİ BÖLÜMÜ

OCAK, 2024

İTÜ, Elektronik ve Haberleşme Mühendisliği Bölümü’nün ilgili Bitirme Tasarım Projesi yönergesine uygun olarak tamamen kendi çalışmamız sonucu hazırladığımız “CRYSTALS-KYBER POST QUANTUM ALGORİTMASININ RISC-V İŞLEMCİ ÜZERİNDE GERÇEKLENMESİ” başlıklı Bitirme Tasarım Projesi’ni sunmaktayız. Bu çalışmayı intihal olmaksızın hazırladığımızı taahhüt eder; intihal olması durumunda bitirme tasarım projesinin başarısız sayılacağını kabul ederiz.

Ahmet ÇELİK ....
040190031

Fatih YILMAZ ....
040190046

Mehmet Anıl KORKMAZ ....
040190035

ÖNSÖZ

Bu bitirme projesi boyunca bizlere rehberlik eden ve destekleriyle yolumuzu aydınlatan sayın Prof.Dr. Sıddıka Berna ÖRS YALÇIN'a en içten teşekkürlerimizi sunarız.

İstanbul Teknik Üniversitesi lisans eğitimi hayatımız boyunca bize katkı sağlayan, bakış açımızı genişleten değerli hocalarımıza ve arkadaşlarımıza teşekkür ederiz.

Son olarak, bizleri her zaman destekleyen ve cesaretlendiren kıymetli ailelerimize sonsuz teşekkürlerimizi sunuyoruz.

Ocak 2024

Ahmet Çelik
Fatih Yılmaz
Mehmet Anıl Korkmaz

İÇİNDEKİLER

Sayfa

KISALTMALAR

ALU	: Arithmetic Logic Unit
BRAM	: Block Random Access Memory
CSR	: Control and Status Register
DIV	: Divider
DSP	: Digital Signal Processor
EX	: Execute Stage
FF	: Flip-Flop
FPGA	: Field-Programmable Gate Array
I/O	: Input/Output
ID	: Instruction Decode Stage
IF	: Instruction Fetch Stage
ISE	: Instruction Set Extension
KeyGen	: Key Generation
LSU	: Load Store Unit
LUT	: Look-Up Table
LUTRAM	: Look-Up Table Random Access Memory
LWE	: Learning With Errors
MULT	: Multiplier
NIST	: National Institute of Standards and Technology
NTT	: Number Theoretic Transform
PQC	: Post-Quantum Cryptography
PLL	: Phase-Locked Loop
PRF	: Pseudo Random Function
PULP	: Parallel Ultra-Low Power
RAM	: Random Access Memory
RISC-V	: Reduced Instruction Set Computing - V
RSA	: Rivest–Shamir–Adleman
XOF	: Extendable Output Function

SEMBOLLER

Θ **:** Theta
ρ **:** Rho
π **:** Pi
χ **:** Chi
ι **:** Iota

ÇİZELGE LİSTESİ

Sayfa

ŞEKİL LİSTESİ

Sayfa

CRYSTALS-KYBER POST QUANTUM ALGORİTMASININ RISC-V İŞLEMCİ ÜZERİNDE GERÇEKLENMESİ

ÖZET

Quantum bilgisayarların olağanüstü hesaplama yetenekleriyle, güncel kriptografik sistemlerin güvenlik temellerini tehdit ederken, bu, kuantum bilgisayar tehditlerine karşı dirençli Post-Kuantum Kriptografi'ye (PQC) doğru bir geçiş gerektirir. Bu sebeple National Institute of Standards and Technology (NIST) tarafından düzenlenen standartlaştırma yarışmaları vardır. Bu alan, gelecekte bilgi güvenliği konusunda önemli bir yer kaplayacağından çalışmalar başlatılmış ve bazı standartlaştırmalar yapılmıştır. Bu makale, özellikle RISC-V Ibex çekirdeği üzerinde yoğunlaşarak, CRYSTALS-Kyber algoritmasını RISC-V çekirdeği kullanarak güçlendirmeye yönelik yenilikçi bir yaklaşımı ele alır.

CRYSTALS-Kyber, kuantum bilgisayar saldırılarına karşı dirençli olarak tasarlanmış bir algoritma olup, kuantum sonrası dünyada güvenli kalacak kriptografik sistemler geliştirmeye yönelik geniş bir girişimin parçasıdır. Açık kaynaklı ve modüler tasarımıyla tanınan RISC-V mimarisi, bu alanda deney yapmak ve yenilikler yapmak için verimli bir zemin sunar. Küçük ve verimli bir 32-bit RISC-V çekirdeği olan Ibex çekirdeğinin seçimi, PQC algoritmalarının uygulanması için kritik olan sadeliği ve esnekliği ile önemlidir.

CRYSTALS-Kyber algoritmasını RISC-V çekirdeğinde hızlandırma stratejisi üç aşamalı olarak varsayılabilir. İlk aşama, detaylı kod profillemeyi içerir. Bu faz, performans darboğazlarını belirlemek ve algoritmanın gerçek dünyadaki uygulamada nasıl davrandığını anlamak için kritiktir. Kodu detaylıca analiz ederek, algoritmanın hız ve verimlilik açısından optimize edilebileceği alanlar belirlenebilir. Detaylı kod profilleme analizi sonucunda en fazla zaman alan fonksiyon donanımda gerçeklenip işlemci mimarisine eklenecektir.

Stratejinin ikinci aşaması, mevcut RISC-V mimarisine özel hızlandırıcı donanımın entegrasyonudur. Bu, CRYSTALS-Kyber algoritmasının belirli fonksiyonlarını hızlandırmak için özel olarak tasarlanmış donanım modüllerinin tasarlanması ve uygulanmasını içerir. Fikir, bazı hesaplama ağırlıklarını yazılımdan donanıma aktararak önemli performans kazanımları elde etmektir. Bu özel donanım entegrasyonu, kapsamlı modifikasyonlar gerektirmeden mevcut mimariyi tamamlayacak şekilde dikkatlice tasarlanmıştır. Kod profilleme sonuçlarına göre en fazla zaman alan fonksiyon KeccakF1600 fonksiyonu olmuş ve donanım olarak gerçeklenip mimariye eklenmiştir.

Stratejinin üçüncü ve son aşaması, sonuçların kapsamlı simülasyonudur. Bu, modifiye edilmiş CRYSTALS-Kyber algoritmasını simüle edilmiş bir RISC-V işlemcide çalıştırarak performans ve verimlilik iyileştirmelerini değerlendirmeyi içerir. Simülasyonlar, kod optimizasyonlarının ve donanım entegrasyonlarının etkinliğini doğrulamak için kritiktir. Ayrıca, algoritmanın farklı koşullar altında nasıl davrandığına dair değerli içgörüler sağlar ve optimal performans için uygulamayı uyarlamamıza yardımcı olur. Çalışmanın sonunda eklenen donanım ile birlikte algoritmanın tamamında %43,26'lık bir hızlandırma sonucu elde edilmiştir.

Bu araştırmanın öne çıkan bir özelliği, yalnızca çekirdek tabanlı uygulama yaklaşımıdır. Ek donanım (Ek mikrokontrolcü çevre birimleri) veya dış hızlandırıcılar kullanmak yerine, tüm CRYSTALS-Kyber algoritması RISC-V çekirdeği üzerinde çalışır. Bu yaklaşımın birkaç avantajı vardır. Genel sistem tasarımını basitleştirir ve donanım ayak izini azaltarak, geniş bir uygulama yelpazesi için daha pratik hale getirir. Ayrıca, performans iyileştirmelerinin yalnızca çekirdek içinde yapılan optimizasyonlar ve geliştirmelerden kaynaklandığını garanti eder.

Sonuç olarak, bu makale, Ibex çekirdeği üzerinde yoğunlaşarak, RISC-V çekirdeği kullanarak CRYSTALS-Kyber PQC algoritmasını hızlandırmaya yönelik yenilikçi ve verimli bir yaklaşım sunar. Detaylı kod profillemesi, özel donanım entegrasyonu ve kapsamlı simülasyonlar yoluyla bu metodoloji, kuantum tehdidine karşı oluşturulmuş CRYSTALS-Kyber algoritmasının performansını (hızını) önemli ölçüde artırır. Bu araştırma, sadece PQC algoritmaları için çekirdek tabanlı uygulamaların uygulanabilirliğini göstermekle kalmaz, aynı zamanda bu çalışma alanında gelecekteki yenilikler için bir öncü olabilir.

IMPLEMENTATION OF CRYSTALS-KYBER POST QUANTUM ALGORITHM ON RISC-V PROCESSOR

SUMMARY

Quantum computers, with their extraordinary computational capabilities, pose a significant threat to the security foundations of current cryptographic systems. This situation necessitates a shift towards Post-Quantum Cryptography (PQC), which is resistant to the threats posed by quantum computing. In response to this need, the National Institute of Standards and Technology (NIST) has organized standardization competitions to foster the development of secure cryptographic protocols for the quantum era. This field is poised to occupy a critical role in the future of information security, prompting the initiation of research and the establishment of some standards.

CRYSTALS-Kyber is an algorithm designed to be resistant to quantum computer attacks, and it is part of the broader initiative to develop cryptographic systems that remain secure in a post-quantum world. The RISC-V architecture, known for its open-source and modular design, offers a fertile ground for experimentation and innovation in this area. The choice of the Ibex core, a small and efficient 32-bit RISC-V core, is significant for its simplicity and flexibility, which are crucial for the implementation of PQC algorithms.

The strategy for accelerating the CRYSTALS-Kyber algorithm on the RISC-V core is multifaceted. The first aspect involves detailed code profiling. This phase is critical for identifying performance bottlenecks and understanding the algorithm's behavior in a real-world implementation. By meticulously analyzing the code, we can pinpoint areas where the algorithm can be optimized for speed and efficiency. The most time consuming function will be realized as hardware to achieve an acceleration.

The second aspect of the strategy is the integration of custom hardware into the existing RISC-V architecture. This involves designing and implementing hardware modules that are specifically tailored to accelerate certain functions of the CRYSTALS-Kyber algorithm. The idea is to offload some of the computational heavy lifting from the software to the hardware, thereby achieving significant performance gains. This custom hardware integration is thoughtfully designed to ensure that it complements the existing architecture without necessitating extensive modifications. According to the profiling results, the hardware that is realised in this study has been KeccakF1600 hardware, which is a part of hash generation.

The third and final aspect is the comprehensive simulation of the results. This involves running the modified CRYSTALS-Kyber algorithm on a simulated RISC-V environment to evaluate its performance and efficiency improvements. The

simulations are critical for validating the effectiveness of the code optimizations and hardware integrations. They also provide valuable insights into how the algorithm behaves under different conditions and help in fine-tuning the implementation for optimal performance. According to the simulation results, the acceleration of the total algorithm has been 43.26% in this study, for CRYSTALS-Kyber PQC algorithm.

A standout feature of this research is its core-only implementation approach. Instead of relying on additional hardware (such as bus interfaces) or external accelerators, the entire CRYSTALS-Kyber algorithm runs on the RISC-V core itself. This approach has several advantages. It simplifies the overall system design and reduces the hardware footprint, making it more practical for a wide range of applications. Additionally, it ensures that the performance improvements are solely due to the optimizations and enhancements made within the core.

In conclusion, this paper presents a novel and efficient approach to accelerating the CRYSTALS-Kyber PQC algorithm using the RISC-V core, particularly focusing on the Ibex core. By employing detailed code profiling, integrating custom hardware, and conducting comprehensive simulations, this methodology significantly enhances the performance and efficiency of the CRYSTALS-Kyber algorithm in a quantum-threatened cryptographic landscape. This research not only demonstrates the viability of core-only implementations for PQC algorithms but also sets a precedent for future innovations in this critical field of study.

1. GİRİŞ

1.1 İlgilenilen Problem ve Amaçlar

CRYSTALS-Kyber, bir Post-Quantum kriptografi algoritmasıdır ve amacı şifrelenmiş verileri hem klasik bilgisayar hem de quantum bilgisayarların ataklarından korumaktır. CRYSTALS-Kyber, var olan kriptografi algoritmalarını belli bir lojikte kullanır ve mesajı bu şekilde birden çok alt algoritma ile şifreler. Bu projenin amacı CRYSTALS-Kyber algoritmasını RISC-V tabanlı bir işlemci üzerinde çalıştırarak analiz etmek ve algoritmayı RISC-V işlemci üzerinde geliştirmektir. Projede temel olarak iki adet problem tanımlanacaktır. Birinci adım CRYSTALS-Kyber algoritmasının RISC-V işlemci üzerinde çalıştırılması, ikinci adım ise algoritman donanım yardımıyla hızlandırılması olacaktır.

Peter Shor tarafından geliştirilen Shor algoritması günümüzdeki klasik şifreleme algoritmaları RSA, Diffie-Helman vb. kolaylıkla kırabilmektedir fakat bu algoritmanın ikili sayı tabanlı bilgisayarlarda gerçeklenmesi mümkün değildir. Kuantum bilgisayarların ortaya çıkması ile birlikte Shor algoritmasının gerçeklenebileceği bir ortam doğmuştur. Kuantum bilgisayarlar günümüzde geldiği noktada hâlen Shor algoritmasını kullanarak çözebilecek hafıza ve işlem gücüne ulaşamamıştır fakat zaman geçtikçe kapasiteleri ve işlem güçleri geliştirilerek arttırılmaktadır. Kuantum bilgisayarların gelişmesi kişisel verilerin gizliliğini gelecek için tehlikeye atmaktadır. Bu yüzden NIST kurumunun düzenlediği yarışma çerçevesinde post-quantum kriptografi algoritmaları geliştirilmektedir. Bu algoritmalardan biri de CRYSTALS-Kyber kriptoloji algoritmasıdır.

Projenin iki amacından ilki CRYSTALS-Kyber Post-Kuantum algoritmasının yazılımsal olarak RISC-V işlemci üzerinde çalıştırılıp algoritmanın içinde temel olarak üretilen üç çıktıya; anahtar çiftlerine (keypair generation), kapsüllenmiş anahtara (encrypted key, Key A) ve geri açılmış anahtara (decrypted key, Key B) çıktılarına erişmektir.

İkinci amacı ise RISC-V işlemci üzerinde gerçeklenen bu algoritmaya özel donanım tasarımı yaparak çalıştırılan algoritmanın hızlandırılması hedeflenmektedir. Geliştirilecek olan donanımların sayısı ve kullanım tipleri (Memory Mapped I/O, Instruction Set Extension vb.) hızlandırma sonuçlarına göre değişkenlik gösterebilir.

Bu problemin seçilmesinin sebebi, NIST tarafından PQC için standartlaştırılmış bir algoritma olması ve aynı zamanda hâlâ güncel olarak hızlandırma için üzerinde çalışılan bir konu olmasıdır. Yazılım ve donanımın birlikte tasarımı, bilgisayar mimarisi, gömülü sistemler gibi konuları beraber içerir.

“CRYSTALS-Kyber” kriptografi algoritmasının referans gerçeklemesi RISC-V için derlenerek Ibex çekirdeği üzerinde gerçeklenmiş, referans gerçeklemenin herhangi bir derleyici optimizasyonu olmadan kod profilleme işlemi gerçekleştirilerek en fazla zaman harcanan fonksiyon bloğu KeccakF1600 tespit edilerek bu fonksiyon bloğuna uygun olarak bir hızlandırıcı donanım tasarlanmıştır. Kod profilleme sonuçları, Tasarlanan donanımın kapladığı alan ve algoritmanın işlem süresi raporlanmıştır. Ayrıca “CRYSTALS-Kyber” algoritması için önceden yapılan yazılım/donanım birlikte tasarım çalışmalarının sonuçları ile elde edilen tasarımın hızlandırma faktörü karşılaştırılmıştır.

1.2 Literatür İncelemesi

Post-Quantum Kriptografi (PQC) CRYSTALS-Kyber algoritması özelinde yapılmış olan ve yayınlanan önceki çalışmalar incelendiğinde ilk olarak saf yazılımda nasıl gerçeklenebilmesine yönelik çalışmalar ile ilerlenmiştir. NIST (National Institute of Standards and Technology) kurumunun hazırladığı standartlarına uygun doküman incelenmiş ve gerçekleme istenilen standartlara uygun olarak devam edilmiştir [1]. Post Quantum Kriptografi CRYSTALS-Kyber Algoritmasının oluşturucuları tarafından yayınlanan C programlama dilinde hazırlanmış algoritmanın koda dökülmüş hali bulunmaktadır [2]. Yazılım olarak gerçeklenmesinde oluşturulan proje ve algoritma yapısı baz alınmıştır. Post Quantum Kriptografi CRYSTALS-Kyber Algoritması içerisinde işlem zamanı olarak oransal olarak hangi algoritma bloğunun etki ettiğine dair önceden yapılmış çalışmalar incelendiğinde yapılmış çalışmalar arasında işlem bloklarından NTT ve Keccak bloklarını hızlandırmak üzerine yoğunlaştığı görülmektedir [3][4][5][6]. PQC (Post-Quantum Cryptography) CRYSTALS-Kyber Algoritmasının hızlandıracak donanımların sınır koşulları olarak güç tüketimi, hızlandırıcının kapladığı alan, hızlandırma oranına göre düşünüldüğünde saf donanım olarak veya Instruction Set Extension(ISE) yapılarak oluşturulduğu görülmüştür [4][5]. Gerçeklemenin yapılacağı RISC-V tabanlı işlemci seçilirken yapılan araştırmada üç farklı aday işlemci incelenmiş olup bunlar: Ibex, Pulpino, Rocketcore'dur. Bu işlemciler arasında uygulanabilirlik olarak içerisinde 3 farklı özelleştirilebilir komut dizini olduğu için Ibex tercih edilmiştir [7].

2. POST-QUANTUM KRİPTOGRAFİ

1995 yılında Peter Shor tarafından ortaya atılan Shor algoritması klasik kriptoloji algoritmalarının temelini oluşturan modüler aritmatik ve logaritma problemlerini kolaylıkla çözebilmektedir [8]. Shor Algoritmasının geliştirildiği dönemde ve sonrasında günümüz kriptografi algoritmalarını çözebilecek donanım ve mimariye sahip olunmadığı için klasik algoritmalar güvenli kabul edilmiştir. Fakat ilk kuantum bilgisayarların ortaya çıkması ile birlikte Shor algoritmasının gerçeklenebileceği bir mimaride donanımın varlığı ortaya çıkmıştır. Günümüz kuantum bilgisayarlarının ulaştığı seviyede RSA [9]ve benzeri günümüz algoritmalarını kırabilecek işlem gücü ve hafızaya ulaşamamasına rağmen bir risk teşkil etmektedir. Bu yüzden kuantum bilgisayarlardan şifrelenmiş verileri korumak amacıyla günümüz ikili sayılı bilgisayar mimarisinde de gerçeklenebilek Amerika Birleşik Devletleri merkezli NIST kurumu tarafından 2012 yılında post-quantum kriptografi algoritmaları özelinde bir yarışma düzenlenmeye başlanmıştır [10]. Yarışmanın temel amacı ikili sayı sistemli günümüz bilgisayarlarında gerçekleştirilen şifreleme işlemlerinin kuantum bilgisayarlara karşı dirençli hale getirmek amaçlanmıştır. Yarışma 2022 yılında 3. Final etabını tamamlamış olup finalist algoritmalar standartlaşma sürecinde ilerlemektedir. CRYSTALS-Kyber Post-Quantum kriptografi algoritması da finalist algoritmalardan birisidir.

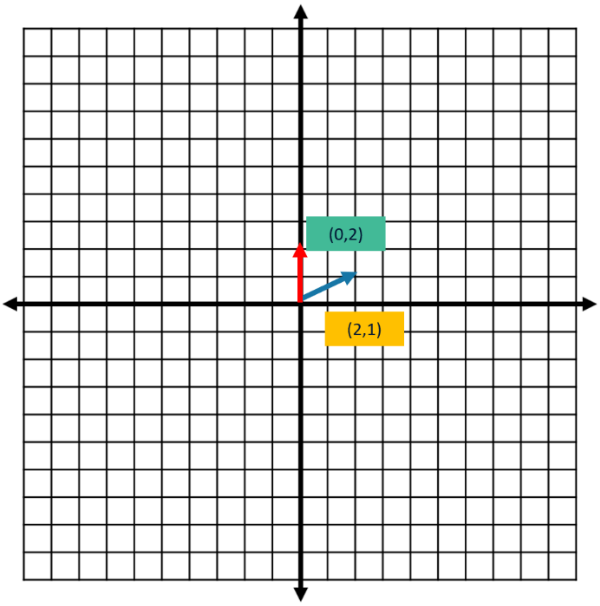
Yarışmanın 3. Etabına kalan 8 algoritmadan CRYSTALS-Kyber dahil olmak üzere 6 tanesi latis tabanlı algoritmalardır [11].

2.1 Latis Tabanlı Kriptografi

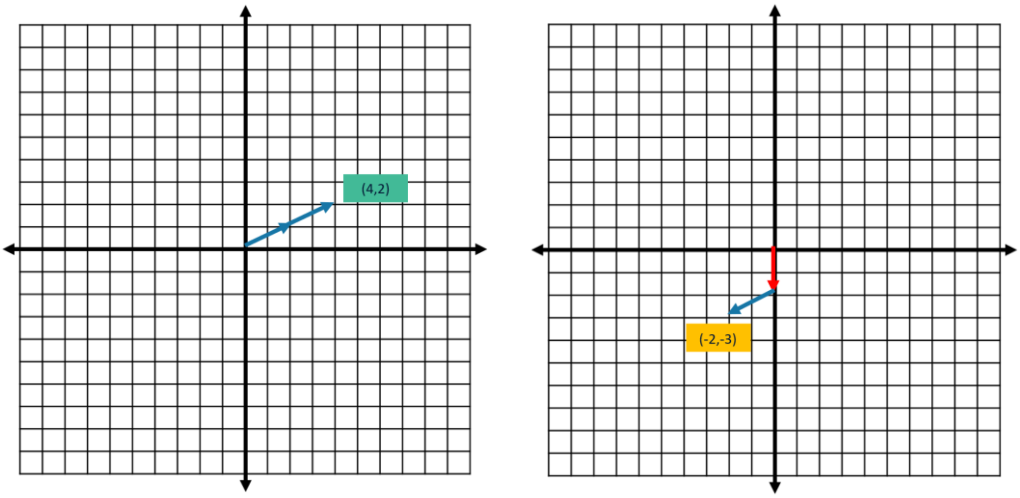
Latis Z^n uzayında tanımlı n adet birbirinden bağımsız a_i vektörünün lineer kombinasyonları olarak tanımlanmaktadır.

$$\{z_1a_1 + z_2a_2 + z_3a_3 \ldots. + z_na_n\} \qquad (2.1)$$

2-boyutlu uzayda Şekil-2.1 ve Şekil-2.2'de gösterildiği gibi $a_1 = (0,2)$ $a_2 = (2,1)$ vektörlerinin lineer kombinasyonları uzaydaki farklı noktaları temsil etmektedir. Herbir nokta bu latisin bir parçasıdır. $a_1$ve a_2 vektörlerinin tüm lineer kombinasyonları latis uzayını tamamlar ve uzay için baz vektörlerdir.

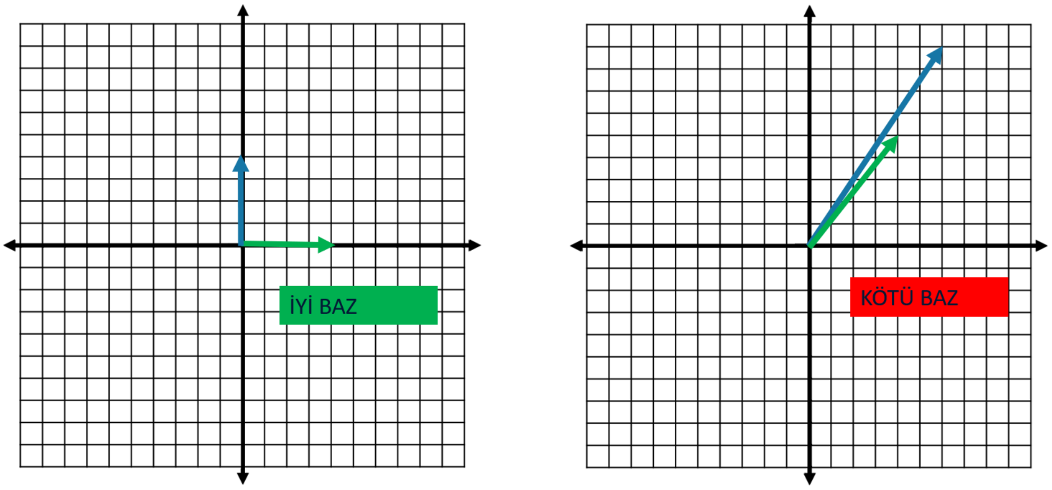


Şekil 2.1 : 2-Boyutlu Uzayda Baz Oluşturan Örnek Vektörler..



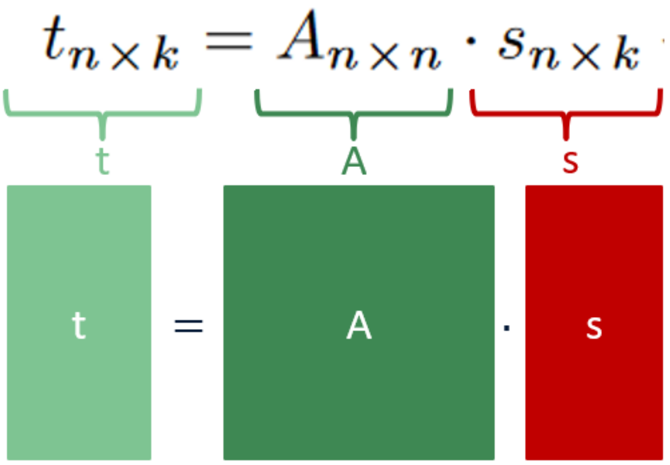
Şekil 2.2 : 2-Boyutlu Uzayda Seçilen Baz Vektörlerin Lineer Kombinasyonu ile Latislerin Oluşturulması.

Bir latis uzayı oluşturan herhangi bir latis baz vektörlerin kombinasyonu ile elde edilebilir. Baz vektörlerin arasında bulunan açı baz vektörlerin bir latis için iyi veya kötü baz teşkil ettiğini göstermektedir. Bir latis uzayı için iyi baz oluşturan vektörler birbirine ortoganal olan vektörlerdir çünkü her bir boyut ayrı bir baz vektörünü temsil etmektedir. İki vektörün arasındaki açının neredeyse sıfır olması bu iki vektörün neredeyse birbirine paralel olduğunu gösterir ve bunlar kötü bazları temsil eder. İyi ve kötü baz vektörü kavramının çıkış noktası “Shortest Point” probleminden gelmektedir. Latis uzayında belirtilen bir noktanın iyi baz vektörleri ile temsil edilebilmesi kolaydır fakat birbiri ile neredeyse aynı olan iki vektör ile bu aramanın yapılması işlemleri komplike hâle getirmektedir. İki boyutlu uzayda bu problemin çözümü basit olarak gözükse de çok boyutlu uzayda (Örneğin 200 boyutlu bir uzayda) bu problemin çözümü kuantum bilgisayarlar ve ikili sayılı bilgisayarlar için de çözümü yıllar alan problemlerdir.



Şekil 2.3 : 2-Boyutlu Uzayda İyi Baz ve Kötü Baz Vektörlerin Görüntülerinin Karşılaştırılması.

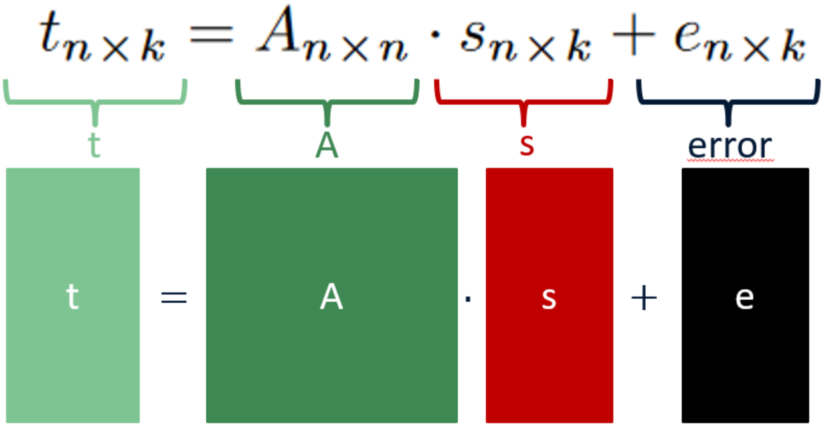
CRYSTALS-Kyber'in de içinde bulunduğu latis tabanlı kriptografi algoritmaları bu problemin çözümünün zorluğuna dayanmaktadır. Bu yönteme özel olarak "Learning with Errors" adı verilmiştir.



Şekil 2.4 : Şifreleme İşlemeninin Sadece Matris Çarpımı ile Gerçeklenmesi.

Şekil-2.4'te gösterilen s matrisi şifrelenecek mesajı A matrisi şifrelemeyi yapmaktadır. Çıkış olarak t matrisi elde edilmektedir. Fakat s matrisi $A^{-1}t$ matrisi ile kolaylıkla elde edilebilir.

Eğer As matrisine çok küçük değerlerde bir hata (error) teriminin eklendiği düşünüldüğünde bu problemin çözümü komplike bir hâle gelicektir. “Learning with errors” yönteminin ve onunla birlikte latis bazlı kriptografi algoritmalarının çıkış noktası burası olmuştur.



Şekil 2.5 : “Learning with Error” Yönteminin Uygulanması.

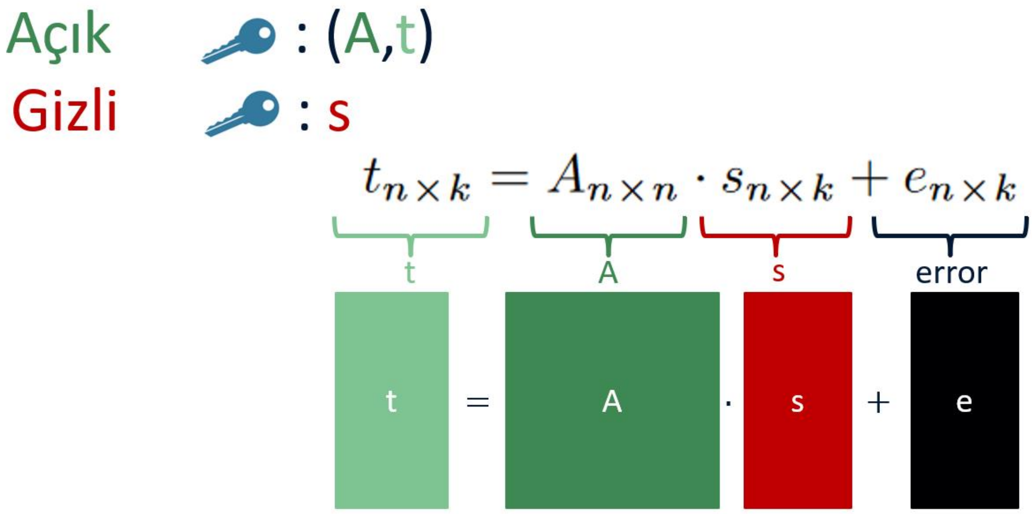
Şifreleme yapan gönderici elinde iyi bazları tutacak olup hata terimi ile şifrelediği mesajı kötü bazlar ile birlikte göndererek verilerini güvenli bir şekilde alıcı tarafa ulaştırır. Bu şekilde asimetrik bir kriptografi sistemi oluşturulmuş olur. Verilen örnekler CRYSTALS-Kyber algoritması özelinde açıklanmıştır.

2.2 CRYSTALS-KYBER Algoritması

CRYSTALS-Kyber algoritması asimetrik bir kriptografi algoritmasıdır. Asimetrik kriptografi algoritmalarında alıcı sadece gönderici tarafından gönderilen açık anahtar (public key) ile mesajını şifreleyip göndericiye gönderebilir ve bu mesajı sadece ilk gönderici elinde bulunan (gizli anahtar) ile açarak mesajı elde edebilir.

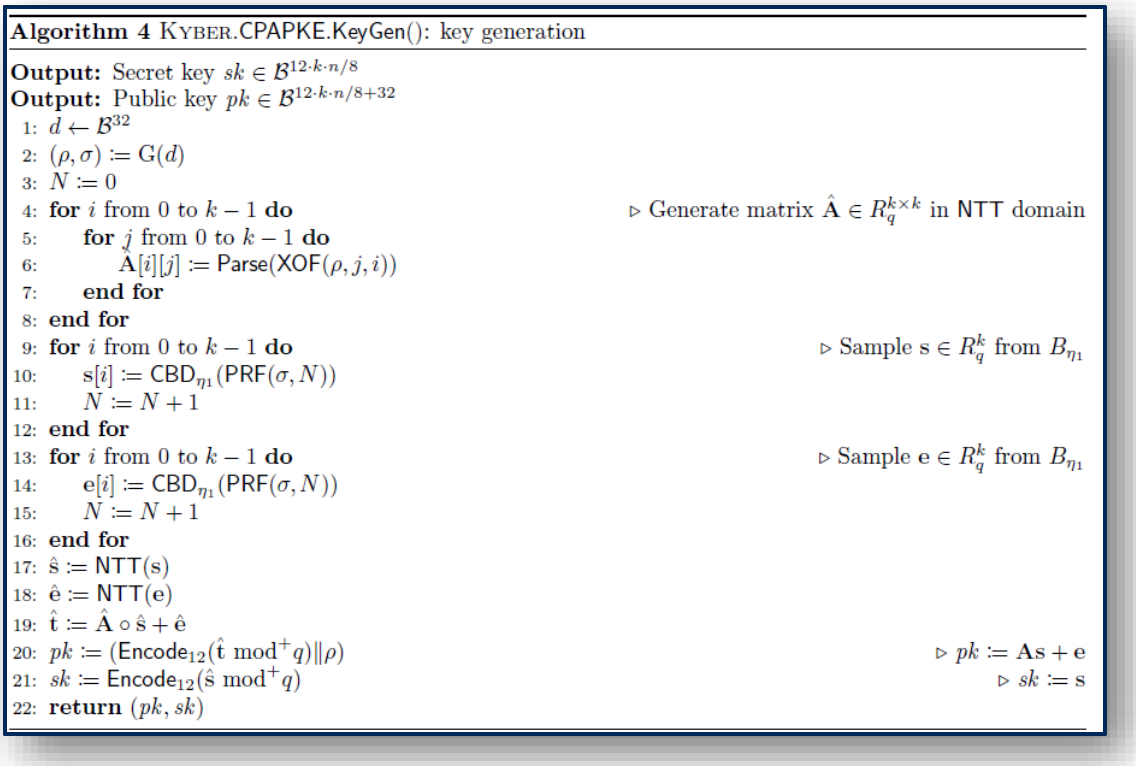
2.2.1 Anahtar üretme (Key generation)

Klasik bir örnek üzerinden gidilecek olursa birbirleri arasında veri alış verişi yapmak isteyen Alice ve Bob olsun. Alice ve Bob arasında şifreli bir veri aktarımı sağlamak için ilk olarak aralarında açık anahtarlarını karşılıklı olarak iletmelidirler. Açık anahtardan önce de gizli anahtarın üretilmesi gerekmektedir. Gizli anahtarlar her ikisi için de karakteristiktir ve sadece kendileri tarafından bilinmektedir.



Şekil 2.6 : Anahtar Üretme Fonksiyonu Matris Gösterimi.

Elde edilen gizli anahtar A matrisi ile çarpılarak ve hata teriminin eklenmesi ile t matrisi elde edilir. A ve t matrisi açık anahtarı oluşturur ve Kötü bazı temsil eder. Kötü bazları kullanarak bir şifreleme işlemi gerçekleştirilebilir fakat LWE dolayısıyla çözülmesi imkansıza yakındır.



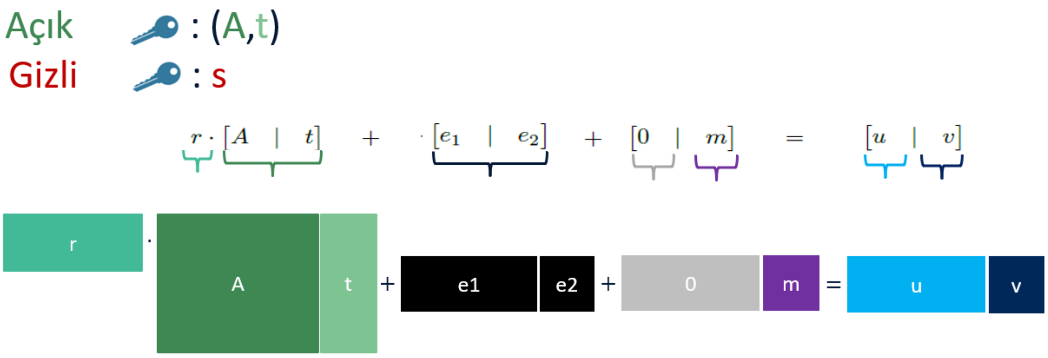
```
Algorithm 4 KYBER.CPAPKE.KeyGen(): key generation
Output: Secret key $sk \in \mathcal{B}^{12 \cdot k \cdot n/8}$
Output: Public key $pk \in \mathcal{B}^{12 \cdot k \cdot n/8+32}$
$d \leftarrow \mathcal{B}^{32}$
$(\rho, \sigma) := \mathrm{G}(d)$
$N := 0$
for $i$ from 0 to $k-1$ do                          ▷ Generate matrix $\hat{\mathbf{A}} \in R_q^{k \times k}$ in NTT domain
    for $j$ from 0 to $k-1$ do
        $\hat{\mathbf{A}}[i][j] := \mathsf{Parse}(\mathsf{XOF}(\rho, j, i))$
    end for
end for
for $i$ from 0 to $k-1$ do                          ▷ Sample $\mathbf{s} \in R_q^k$ from $B_{\eta_1}$
    $\mathbf{s}[i] := \mathsf{CBD}_{\eta_1}(\mathsf{PRF}(\sigma, N))$
    $N := N+1$
end for
for $i$ from 0 to $k-1$ do                          ▷ Sample $\mathbf{e} \in R_q^k$ from $B_{\eta_1}$
    $\mathbf{e}[i] := \mathsf{CBD}_{\eta_1}(\mathsf{PRF}(\sigma, N))$
    $N := N+1$
end for
$\hat{\mathbf{s}} := \mathsf{NTT}(\mathbf{s})$
$\hat{\mathbf{e}} := \mathsf{NTT}(\mathbf{e})$
$\hat{\mathbf{t}} := \hat{\mathbf{A}} \circ \hat{\mathbf{s}} + \hat{\mathbf{e}}$
$pk := (\mathsf{Encode}_{12}(\hat{\mathbf{t}} \bmod^{+} q) \| \rho)$          ▷ $pk := \mathbf{As} + \mathbf{e}$
$sk := \mathsf{Encode}_{12}(\hat{\mathbf{s}} \bmod^{+} q)$                   ▷ $sk := \mathbf{s}$
return $(pk, sk)$
```

Şekil 2.7 : Anahtar Üretme Fonksiyon Bloğu Referans Algoritması.[12]

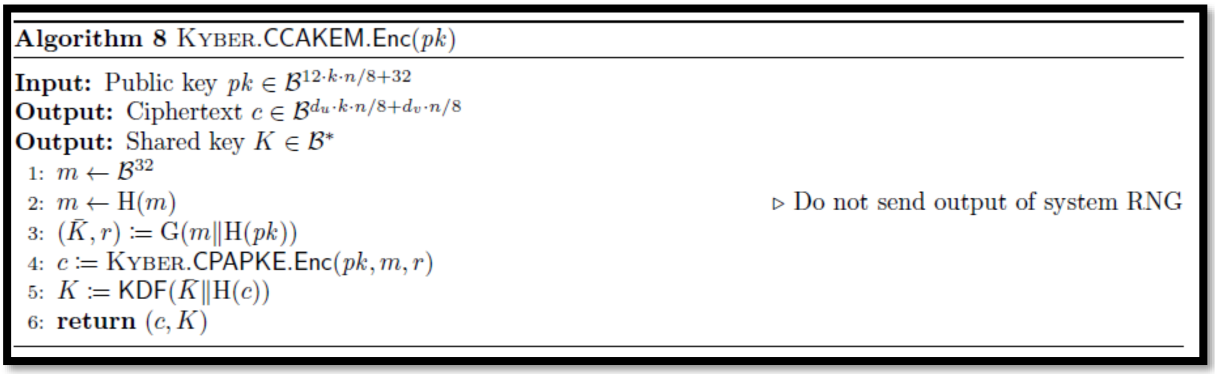
Anahtar üretme fonksiyonunun (KeyGen) akışı Şekil-2.7'de gösterilmiştir. Rastgele sayılardan oluşan A matrisi standart hash fonksiyonlarından olan SHAKE-128 (XOF) fonksiyonu kullanılarak elde edilmişitr. Gizli anahtar (s) ve hata terimi (e) hash fonksiyonu SHAKE-256 (PRF) ile elde edilmişitr. Tüm matrisler bir polinomu temsil ettiği ve polinomlarda aritmetik işlemler ile birlikte yapılan modüler işlemleri kolaylaştıran NTT dönüşümü kullanılmıştır[13]. İşlemler öncesi bütün matrisler NTT uzayına aktarılmış ve işlemler tamamlandıktan sonra tekrardan normal uzaya döndürülmüştür.

2.2.2 Şifreleme (Encryption)

Alice tarafından gönderilen açık anahtarı kullanarak Bob göndermek istediği mesajı şifreler. Şekil-2.8'de belirtilen r matrisi rastgele üretilmiş bir matristir, "random token" olarak adlandırılır. Şifre çözme (Decryption) tarafında elimine edilir. m matrisi taşınmak istenen mesajı temsil etmektedir. e_1 ve e_2 hata terimlerini temsil etmektedir, hata terimeleri A,t matrisini elinde bulunduran 3. kişi tarafından çözülememesi için eklenmiştir. Şifreleme işleminin tamamlanmasının ardından Bob şifrelenmiş mesajı Alice’e u ve v matrisleri olarak gönderir.

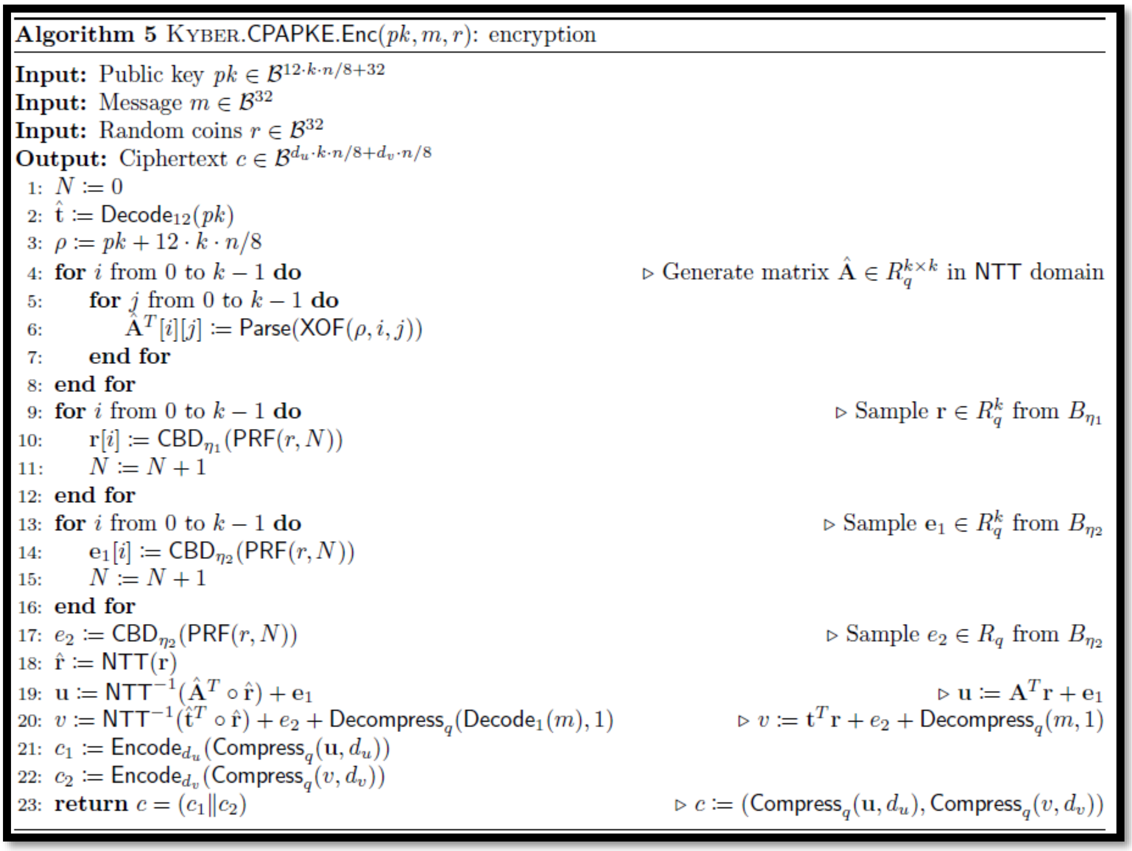


Şekil 2.8 : Şifreleme Algoritmasının Matris Gösterimi.



```
Algorithm 8 KYBER.CCAKEM.Enc($pk$)
Input: Public key $pk \in \mathcal{B}^{12\cdot k\cdot n/8+32}$
Output: Ciphertext $c \in \mathcal{B}^{d_u\cdot k\cdot n/8+d_v\cdot n/8}$
Output: Shared key $K \in \mathcal{B}^*$
$m \leftarrow \mathcal{B}^{32}$
$m \leftarrow \mathrm{H}(m)$                                   ▷ Do not send output of system RNG
$(\bar{K}, r) := \mathrm{G}(m \| \mathrm{H}(pk))$
$c := \mathrm{KYBER.CPAPKE.Enc}(pk, m, r)$
$K := \mathrm{KDF}(\bar{K} \| \mathrm{H}(c))$
return $(c, K)$
```

Şekil 2.9 : Şifreleme Fonksiyon Bloğunun Referans Algoritması.[12]



```
Algorithm 5 KYBER.CPAPKE.Enc($pk, m, r$): encryption
Input: Public key $pk \in \mathcal{B}^{12 \cdot k \cdot n/8+32}$
Input: Message $m \in \mathcal{B}^{32}$
Input: Random coins $r \in \mathcal{B}^{32}$
Output: Ciphertext $c \in \mathcal{B}^{d_u \cdot k \cdot n/8 + d_v \cdot n/8}$
$N := 0$
$\hat{\mathbf{t}} := \mathsf{Decode}_{12}(pk)$
$\rho := pk + 12 \cdot k \cdot n/8$
for $i$ from 0 to $k-1$ do                          ▷ Generate matrix $\hat{\mathbf{A}} \in R_q^{k \times k}$ in NTT domain
    for $j$ from 0 to $k-1$ do
        $\hat{\mathbf{A}}^T[i][j] := \mathsf{Parse}(\mathsf{XOF}(\rho, i, j))$
    end for
end for
for $i$ from 0 to $k-1$ do                          ▷ Sample $\mathbf{r} \in R_q^k$ from $B_{\eta_1}$
    $\mathbf{r}[i] := \mathsf{CBD}_{\eta_1}(\mathsf{PRF}(r, N))$
    $N := N + 1$
end for
for $i$ from 0 to $k-1$ do                          ▷ Sample $\mathbf{e}_1 \in R_q^k$ from $B_{\eta_2}$
    $\mathbf{e}_1[i] := \mathsf{CBD}_{\eta_2}(\mathsf{PRF}(r, N))$
    $N := N + 1$
end for
$e_2 := \mathsf{CBD}_{\eta_2}(\mathsf{PRF}(r, N))$          ▷ Sample $e_2 \in R_q$ from $B_{\eta_2}$
$\hat{\mathbf{r}} := \mathsf{NTT}(\mathbf{r})$
$\mathbf{u} := \mathsf{NTT}^{-1}(\hat{\mathbf{A}}^T \circ \hat{\mathbf{r}}) + \mathbf{e}_1$          ▷ $\mathbf{u} := \mathbf{A}^T\mathbf{r} + \mathbf{e}_1$
$v := \mathsf{NTT}^{-1}(\hat{\mathbf{t}}^T \circ \hat{\mathbf{r}}) + e_2 + \mathsf{Decompress}_q(\mathsf{Decode}_1(m), 1)$          ▷ $v := \mathbf{t}^T\mathbf{r} + e_2 + \mathsf{Decompress}_q(m, 1)$
$c_1 := \mathsf{Encode}_{d_u}(\mathsf{Compress}_q(\mathbf{u}, d_u))$
$c_2 := \mathsf{Encode}_{d_v}(\mathsf{Compress}_q(v, d_v))$
return $c = (c_1 \| c_2)$          ▷ $c := (\mathsf{Compress}_q(\mathbf{u}, d_u), \mathsf{Compress}_q(v, d_v))$
```

Şekil 2.10 : Şifreleme Algoritmasının Fonksiyon Bloğunun Referans Algoritması.[12]

CRYSTALS-Kyber algoritmasının şifreleme fonksiyon bloğunda A matrisini elde edebilmek için SHAKE-128 (XOF); r, e_1, e_2 matrisleri SHAKE-256 (PRF) güvenli hash fonksiyonları kullanılmıştır.

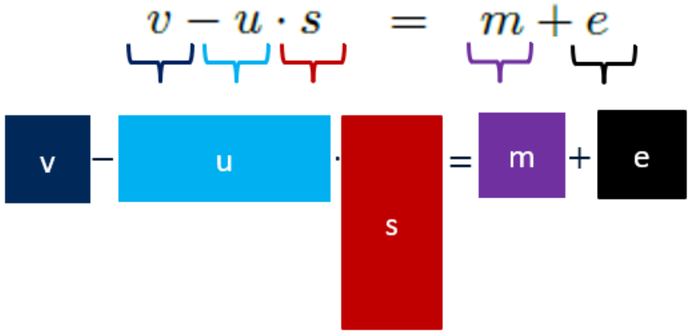
2.2.3 Şifre çözme (Decryption)

Şifrelenmiş mesajı Alice'e ait olan açık anahtar ile yapan Bob, şifrelenmiş mesaj Alice'e ulaştıktan sonra şifreyi çözebilmek için şifre çözme (decryption) fonksiyon bloğunu çağırır.

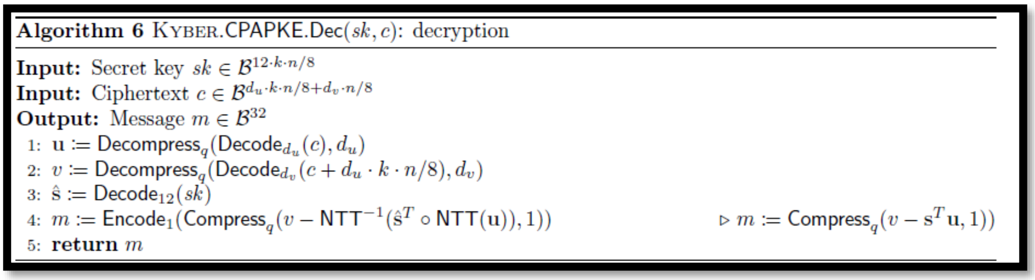
$$\begin{aligned} v &= r \cdot t + e_2 + m \\ u &= r \cdot A + e_1 \\ A \cdot s + e &= t \end{aligned} \qquad \begin{aligned} v - u \cdot s &= r \cdot t + e_2 + m - (r \cdot A + e_1) \cdot s \\ &= e \cdot e_1 + e_2 + m - e_1 \cdot s \\ &= m + e \cdot e_1 + e_2 - e_1 \cdot s \end{aligned}$$

Şekil 2.11 : Şifre Çözme Algoritması r Matrisinin Elimine Edilmesinin Cebirsel Gösterimi.

Şifreleme işleminden elde edilen u ve v matrisleri ve gizli anahtar (s) kullanılarak, Şekil-2.11'de belirtildiği gibi mesaj elde edilir. Hata terimleri modülo operosyonları ile elimine edilir ve mesaj elde edilir.

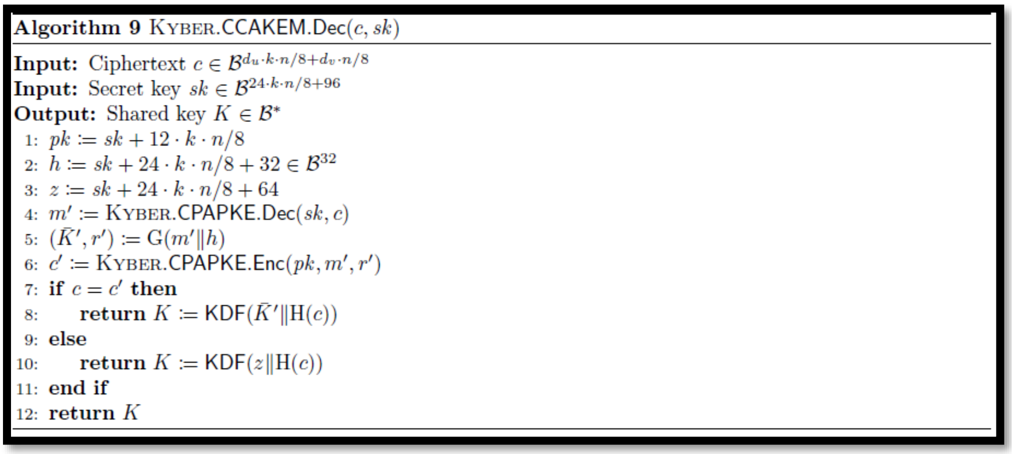


Şekil 2.12 : Şifre Çözme Algoritmasının Matris Gösterimi.



```
Algorithm 6 KYBER.CPAPKE.Dec($sk, c$): decryption
Input: Secret key $sk \in \mathcal{B}^{12 \cdot k \cdot n/8}$
Input: Ciphertext $c \in \mathcal{B}^{d_u \cdot k \cdot n/8 + d_v \cdot n/8}$
Output: Message $m \in \mathcal{B}^{32}$
  $\mathbf{u} := \mathsf{Decompress}_q(\mathsf{Decode}_{d_u}(c), d_u)$
  $v := \mathsf{Decompress}_q(\mathsf{Decode}_{d_v}(c + d_u \cdot k \cdot n/8), d_v)$
  $\hat{\mathbf{s}} := \mathsf{Decode}_{12}(sk)$
  $m := \mathsf{Encode}_1(\mathsf{Compress}_q(v - \mathsf{NTT}^{-1}(\hat{\mathbf{s}}^T \circ \mathsf{NTT}(\mathbf{u})), 1))$    ▷ $m := \mathsf{Compress}_q(v - \mathbf{s}^T\mathbf{u}, 1))$
  return $m$
```

Şekil 2.13 : Şifre Çözme Fonksiyon Bloğunun Referans Algoritması.[12]



```
Algorithm 9 KYBER.CCAKEM.Dec($c, sk$)
Input: Ciphertext $c \in \mathcal{B}^{d_u \cdot k \cdot n/8 + d_v \cdot n/8}$
Input: Secret key $sk \in \mathcal{B}^{24 \cdot k \cdot n/8 + 96}$
Output: Shared key $K \in \mathcal{B}^*$
$pk := sk + 12 \cdot k \cdot n/8$
$h := sk + 24 \cdot k \cdot n/8 + 32 \in \mathcal{B}^{32}$
$z := sk + 24 \cdot k \cdot n/8 + 64$
$m' :=$ KYBER.CPAPKE.Dec($sk, c$)
$(\bar{K}', r') := \mathrm{G}(m' \| h)$
$c' :=$ KYBER.CPAPKE.Enc($pk, m', r'$)
if $c = c'$ then
    return $K :=$ KDF($\bar{K}' \| \mathrm{H}(c)$)
else
    return $K :=$ KDF($z \| \mathrm{H}(c)$)
end if
return $K$
```

Şekil 2.14 : Şifre Çözme Fonksiyonu Bloğunun Referans Algoritması.[12]

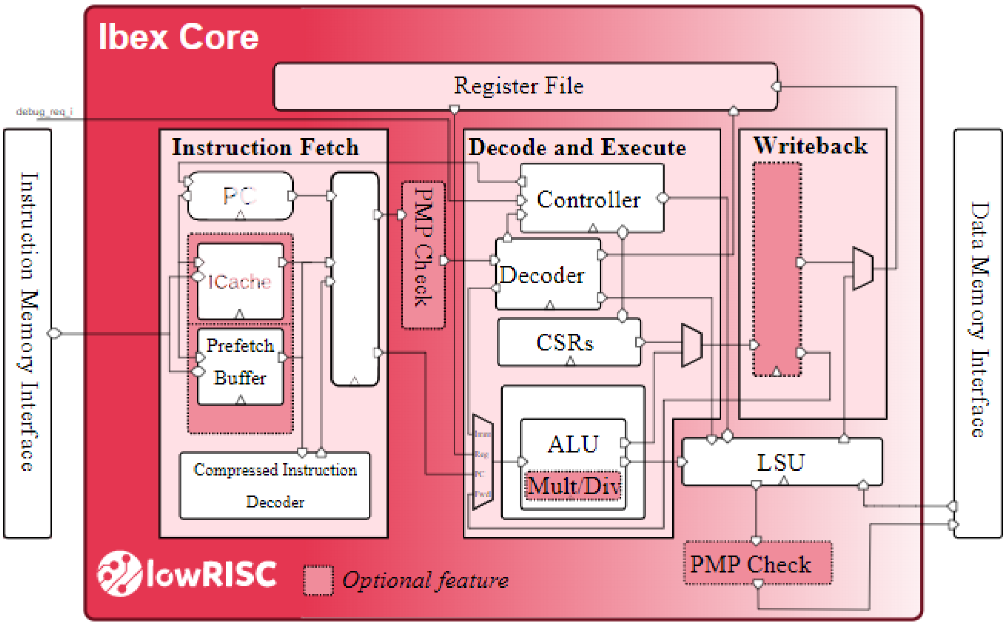
Şifre çözme (decryption) fonkisyon bloğunda Algoritma-5 (Şekil 2.10) tekrardan çağrılır.

3. CRYSTALS-KYBER'İN YAZILIMSAL GERÇEKLENMESİ

3.1 Kullanılacak Olan İşlemcinin Belirlenmesi ve Çalıştırılması

Proje için işlemci seçiminde mimari olarak RISC-V kullanılmasına karar verilmiştir. RISC-V, açık kaynaklı bir talimat seti mimarisi (Instruction Set Architecture, ISA) olarak tanımlanabilir. 2010 yılında Kaliforniya Üniversitesi, Berkeley'de geliştirilen bu mimari, özgürce kullanılabilen ve değiştirilebilen bir tasarıma sahiptir. RISC-V, "Reduced Instruction Set Computer" (RISC) prensiplerine dayanır ve basit ama güçlü bir tasarım sunar. Bu mimari, akademik ortamlarda geniş bir kullanım alanı bulmuştur

Projede, literatür incelemesi kısmında da belirtildiği üzere, Ibex RISC-V tabanlı işlemcisi kullanılmak üzere belirlenmiştir [7] ve bir demo koduyla işlemci Vivado simülasyon ortamında gerçeklenmiştir.



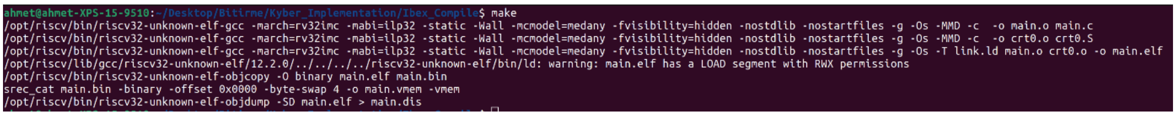
Şekil 3.1: Ibex Çekirdeği.[7]

Ibex RISC-V Vivado simülasyon ortamında Nexys A7-100T FPGA kartı seçilerek [14] hazırlandıktan sonra işlemcinin çalışır olduğunu test etmek için bir deneme kodu derlenmiş ve simülasyon ortamında işlemcide çalıştırılmıştır. Şekil 3.1'de verilen kod ile basit bir string XOR metodu ile şifrelenmiş, geri açılmış ve orijinali ile karşılaştırılmıştır. Eğer bu işlemler doğru bir şekilde yapıldıysa tanımlanan LED adresine 040190046, yanlış yapıldıysa 040190031 yazılacaktır.

```c
int main(void)
{

   volatile unsigned int *LED_ADDR = (volatile unsigned int *) 0x0000c010;
   unsigned char isWrong = 0;

   char key = 'x';
   char str[] = "Crystals-KYBER";
   char original[] = "Crystals-KYBER";

   // Encrypt
   for(int i=0; str[i] != '\0'; ++i)
   {
      str[i] = str[i] ^ key;
   }

   // Decrypt
   for(int i=0; str[i] != '\0'; ++i)
   {
      str[i] ^= key;
   }

   // Check
   for(int i=0; str[i] != '\0'; ++i)
   {
      if(str[i] != original[i])
      {
         isWrong = 1;
         break;
      }
   }

   if(!isWrong)
      *LED_ADDR = 40190046;

   else
      *LED_ADDR = 40190031;


  return 0;
}
```

Şekil 3.2: İşlemci Üzerinde Denenen Kod.

Yazılan kod, RISC-V IMC buyruk seti ile Kyber referans gerçeklemesini birebir kullanıp [2] derlenerek işlemci üzerinde çalışabilir hexadecimal dosya oluşturulmuştur. Çalıştırılan makefile dosyasının terminal üzerindeki çıktısı Şekil 3.3'te verilmiştir.



```
ahmet@ahmet-XPS-15-9510:~/Desktop/Bitirme/Kyber_Implementation/Ibex_Compile$ make
/opt/riscv/bin/riscv32-unknown-elf-gcc -march=rv32imc -mabi=ilp32 -static -Wall -mcmodel=medany -fvisibility=hidden -nostdlib -nostartfiles -g -Os -MMD -c  -o main.o main.c
/opt/riscv/bin/riscv32-unknown-elf-gcc -march=rv32imc -mabi=ilp32 -static -Wall -mcmodel=medany -fvisibility=hidden -nostdlib -nostartfiles -g -Os -MMD -c  -o crt0.o crt0.S
/opt/riscv/bin/riscv32-unknown-elf-gcc -march=rv32imc -mabi=ilp32 -static -Wall -mcmodel=medany -fvisibility=hidden -nostdlib -nostartfiles -g -Os -T link.ld main.o crt0.o -o main.elf
/opt/riscv/lib/gcc/riscv32-unknown-elf/12.2.0/../../../../riscv32-unknown-elf/bin/ld: warning: main.elf has a LOAD segment with RWX permissions
/opt/riscv/bin/riscv32-unknown-elf-objcopy -O binary main.elf main.bin
srec_cat main.bin -binary -offset 0x0000 -byte-swap 4 -o main.vmem -vmem
/opt/riscv/bin/riscv32-unknown-elf-objdump -SD main.elf > main.dis
```

Şekil 3.3: Kodun Derlenmesi.

Kodun derlenmesi sonucu elde edilen hexadecimal dosya Xilinx Vivado [15] simülasyon ortamında kullanılarak kodun işlemci üzerindeki simülasyonu yapılmış ve Şekil 3.4'te verilmiştir.

Name	Value
IO_CLK	1
IO_RST_N	1
LED1[31:0]	40190046
clk_sys	1
rst_sys_n	1
instr_req	0
instr_gnt	0
instr_rvalid	1
instr_addr[31:0]	0000010c
instr_rdata[31:0]	XXXXXXXX
data_req	1
data_gnt	1
data_rvalid	1
data_we	1
data_be[3:0]	f
data_addr[31:0]	0000c010
data_wdata[31:0]	0265405e
data_rdata[31:0]	XXXXXXXX
mem_addr[31:0]	0000c010

Şekil 3.4: Simülasyon Sonucu..

Simülasyon sonucunda da görüldüğü üzere işlemci doğru bir şekilde çalışmış ve sonuç olarak LED adresine 040190046 (Doğru çalıştığı durumda beklenen değer) yazılmıştır.

3.2 CRYSTALS-Kyber Algoritmasının Linux'ta Çalıştırılması

Referans olarak kullanılan [2] CRYSTALS-Kyber Algoritması Linux işletim sistemi kullanan bir bilgisayar üzerinde çalıştırılacak ve referans çıktıların alınması işlemi gerçekleştirilecektir. Bu işlem için referans gerçeklemede verilen kodlardan test_vectors512 kodu kullanılmıştır. Referans kod github sayfasından indirildikten sonra verilen makefile içerisinde belli değişiklikler yapılarak test_vectors512 kodunun çıktıları alınmıştır. Fakat bu işlemin Ibex çekirdeği üzerinde de gerçekleştirilmesi için kod üzerinde bazı değişiklikler yapılmalıdır ve sadeleştirilmelere gidilmelidir.

Örneğin I/O işlemleri ile ilgili olan printf fonksiyonları kod içerisinde kullanılmamalı, hafızada ekstradan yer kaplamasının yanında fonksiyonalitesinin de olmayacağı için kod içerisinden çıkarılmalıdır.

Ek olarak, referans gerçeklemede sadece test_vectors512 üzerinden gidileceği için kod içerisinde verilmiş olan ekstra parametreler de hafızada yer kaplayacaktır. Örneğin, kod içerisinde 512 yerine 1024 ile de işlem yapılmasına izin verilir ve bunun için de ekstradan fonksiyon tanımlamaları yapılmaktadır. Kullanılmayan hiçbir fonksiyonun ve hiçbir parametrenin derlemesine izin vermemek amacıyla referans gerçeklemedeki C kodu kullanılarak farklı bir dosyada sadece Ibex işlemci üzerinde test_vectors512 kodunun çalıştırılması amacıyla özel bir dosya daha oluşturulmuştur. Bu dosyanın çıktıları ile orijinal test_vectors512 kodunun çıktılarının aynı olması beklenmektedir.

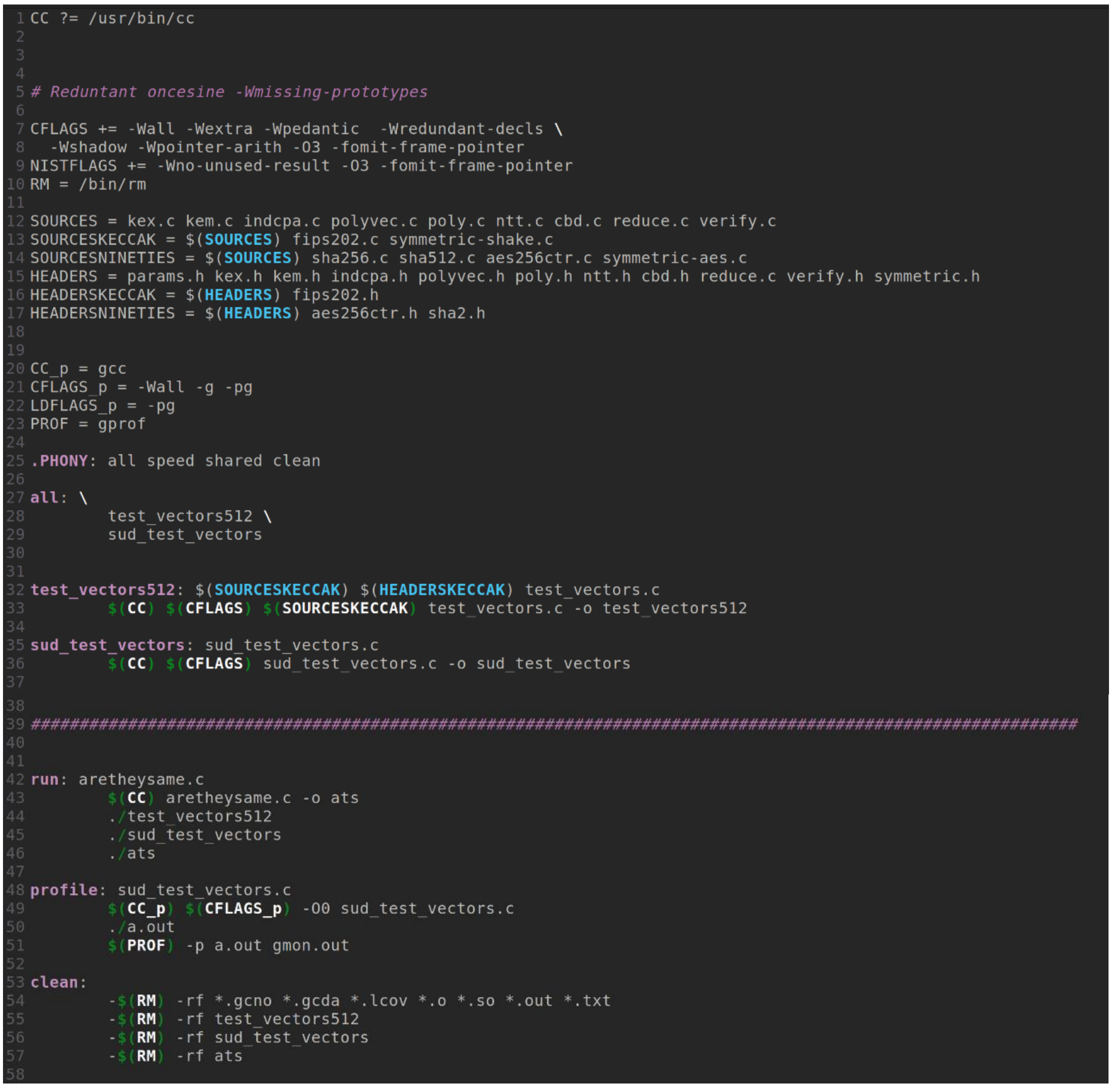
Bundan dolayı bu işlemin yazılımsal doğruluğunu test etmek amacıyla bir makefile dosyası oluşturulmuştur. Bu makefile dosyası öncelikle orijinal test_vectors512 dosyasını çalıştırdıktan sonra Ibex için sadeleştirilmiş olan sud_test_vectors512 dosyasını çalıştırır. Her iki .c dosyası da kendi çıktılarını birer .txt dosyasına kaydederler. Daha sonra ats.c (Are They Same) kodu da bu dosyaların aynı olup olmadığını kontrol eder. Eğer dosyalar aynı ise, yani sadeleştirme işlemi başarıyla yapılıp CRYSTALS-Kyber algoritmasının çalışmasından herhangi bir şey kaybettirmeden kod RISC-V işlemci için uygun hale getirilebildiyse ekrana “The files are the same” yazar. Eğer dosyaların 1 biti bile birbirinden farklıysa “The files are different” ekrana yazdırılacaktır.

Şekil 3.5’te çalıştırılan algoritma, Şekil 3.6’da dosyaların çalıştırılması için gereken makefile verilmiştir.

```c
// Key-pair generation
crypto_kem_keypair(pk, sk);

fprintf(fp, "Key-pair Generation Function Output:\n");
fprintf(fp, "Secret Key:\n");
for(int o=0; o<CRYPTO_SECRETKEYBYTES; o++)
  fprintf(fp, "%x ", sk[o]);

fprintf(fp, "\n\nPublic Key:\n");
for(int o=0; o<CRYPTO_PUBLICKEYBYTES; o++)
  fprintf(fp, "%x ", pk[o]);
fprintf(fp,"\n\n\n\n\n\n");

// // Encapsulation
crypto_kem_enc(ct, key_b, pk);

fprintf(fp, "Encapsulation Function Output:\n");
fprintf(fp,"Ciphertext: \n");
for(int o=0;o<CRYPTO_CIPHERTEXTBYTES;++o)
  fprintf(fp,"%x ",ct[o]);
fprintf(fp,"\nShared Secret B: \n");
for(int o=0;o<CRYPTO_BYTES;++o)
  fprintf(fp, "%x ",key_b[o]);
fprintf(fp,"\n");

fprintf(fp,"\n\n\n\n\n\n");

// // Decapsulation
crypto_kem_dec(key_a, ct, sk);

fprintf(fp, "Decapsulation Function Output:\n");
fprintf(fp, "Shared Secret A: \n");
for(int o=0;o<CRYPTO_BYTES;++o)
  fprintf(fp, "%x ",key_a[o]);
fprintf(fp, "\n\n");

for(int o=0;o<CRYPTO_BYTES;++o) {
  if(key_a[o] != key_b[o]) {
    isWrong = 1;
  }
}
if(!isWrong)
{
  fprintf(fp,"\nEN-DEC is OK!\n");
  printf("\nEN-DEC is OK!\n");
}
```

Şekil 3.5: CRYSTALS-Kyber Algoritması.

Şekil 3.5’te görüldüğü üzere algoritma her bir alt aşamasının (Anahtar çifti oluşturma, kapsülleme ve kapsül açma) ardından çıktıları kendi dosyasına yazar.



```
CC ?= /usr/bin/cc



# Reduntant oncesine -Wmissing-prototypes

CFLAGS += -Wall -Wextra -Wpedantic  -Wredundant-decls \
  -Wshadow -Wpointer-arith -O3 -fomit-frame-pointer
NISTFLAGS += -Wno-unused-result -O3 -fomit-frame-pointer
RM = /bin/rm

SOURCES = kex.c kem.c indcpa.c polyvec.c poly.c ntt.c cbd.c reduce.c verify.c
SOURCESKECCAK = $(SOURCES) fips202.c symmetric-shake.c
SOURCESNINETIES = $(SOURCES) sha256.c sha512.c aes256ctr.c symmetric-aes.c
HEADERS = params.h kex.h kem.h indcpa.h polyvec.h poly.h ntt.h cbd.h reduce.c verify.h symmetric.h
HEADERSKECCAK = $(HEADERS) fips202.h
HEADERSNINETIES = $(HEADERS) aes256ctr.h sha2.h


CC_p = gcc
CFLAGS_p = -Wall -g -pg
LDFLAGS_p = -pg
PROF = gprof

.PHONY: all speed shared clean

all: \
        test_vectors512 \
        sud_test_vectors


test_vectors512: $(SOURCESKECCAK) $(HEADERSKECCAK) test_vectors.c
        $(CC) $(CFLAGS) $(SOURCESKECCAK) test_vectors.c -o test_vectors512

sud_test_vectors: sud_test_vectors.c
        $(CC) $(CFLAGS) sud_test_vectors.c -o sud_test_vectors


############################################################################################################


run: aretheysame.c
        $(CC) aretheysame.c -o ats
        ./test_vectors512
        ./sud_test_vectors
        ./ats

profile: sud_test_vectors.c
        $(CC_p) $(CFLAGS_p) -O0 sud_test_vectors.c
        ./a.out
        $(PROF) -p a.out gmon.out

clean:
        -$(RM) -rf *.gcno *.gcda *.lcov *.o *.so *.out *.txt
        -$(RM) -rf test_vectors512
        -$(RM) -rf sud_test_vectors
        -$(RM) -rf ats

```

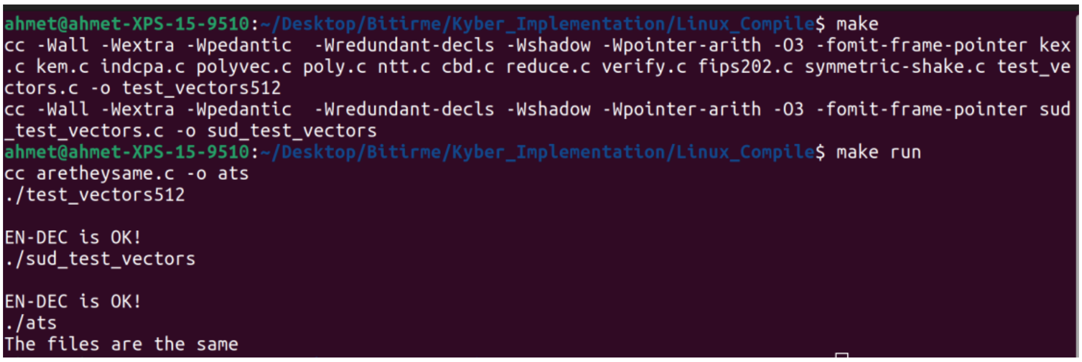
Şekil 3.6: Kullanılan makefile Dosyası.

Şekil 3.6’da görüldüğü üzere makefile ile kullanılmak üzere 4 asıl operasyon sunulmuştur:

- *make* komutu ile orijinal dosya olan test_vectors512 dosyası ve sud_test_vectors dosyaları derlenir.
- *make run* komutu ile bu dosyaların çıktıları aretheysame.c dosyası ile kontrol edilir.
- Raporun devamında yapılacak olan kod profilleme işlemi için *make profile* komutunu kullanıma sunar.

- Çıktıların silinmesi ve yeni bir test işlemine olanak sağlaması amacıyla *make clean* komutunu kullanıma sunar.

Şekil 3.7’de verilen terminal çıktısı ile görülebilir ki sud_test_vectors dosyası doğru bir şekilde çalışmış ve sadeleştirme işleminin doğruluğu kanıtlanmıştır. Böylece CRYSTALS-Kyber algoritması RISC-V mimarili Ibex işlemci üzerinde çalışıtırılmak üzere hazır hale getirilmiştir.



```
ahmet@ahmet-XPS-15-9510:~/Desktop/Bitirme/Kyber_Implementation/Linux_Compile$ make
cc -Wall -Wextra -Wpedantic  -Wredundant-decls -Wshadow -Wpointer-arith -O3 -fomit-frame-pointer kex
.c kem.c indcpa.c polyvec.c poly.c ntt.c cbd.c reduce.c verify.c fips202.c symmetric-shake.c test_ve
ctors.c -o test_vectors512
cc -Wall -Wextra -Wpedantic  -Wredundant-decls -Wshadow -Wpointer-arith -O3 -fomit-frame-pointer sud
_test_vectors.c -o sud_test_vectors
ahmet@ahmet-XPS-15-9510:~/Desktop/Bitirme/Kyber_Implementation/Linux_Compile$ make run
cc aretheysame.c -o ats
./test_vectors512

EN-DEC is OK!
./sud_test_vectors

EN-DEC is OK!
./ats
The files are the same
```

Şekil 3.7: Hazırlanan Dosyanın Doğru Çalıştığını Gösteren Terminal Çıktısı..

3.3 CRYSTALS-Kyber Algoritmasının RISC-V İşlemcide Çalıştırılması

Hazırlanmış ve Linux’ta test edilmiş olan sud_test_vectors dosyasının RISC-V tabanlı işlemci olan Ibex için derlenmesiyle bu işlem gerçekleştirilecektir. Daha önceden işlemcinin çalışırlığını test etmek için kullanılan makefile dosyası, bu kez sud_test_vectors dosyasını girdi olarak alıp CRYSTALS-Kyber algoritması için hexadecimal hafıza dosyasını üretecektir. Bu işlem gerçekleştirilmiş ve elde edilen main.vmem dosyası Vivado’da simüle edilmek üzere hafızaya konmuştur. Algoritmanın çalışırlığı yine benzer şekilde kontrol edilmiştir. Eğer algoritma anahtarları ürettikten sonra doğru şekilde enkapsülleyip geri açabilirse LED adresine 40190031 yazacaktır. Bu işlem doğru şekilde yapılamazsa 40190046 LED adresinde gözükecektir (Şekil 3.8). Vivado’da yaklaşık 3 saatlik simülasyonun ardından elde edilen simülasyon çıktısı Şekil 3.9’da verilmiştir.

```
for(i=0;i<NTESTS;i++) {
    // Key-pair generation
    crypto_kem_keypair(pk, sk);

    // // Encapsulation
    crypto_kem_enc(ct, key_b, pk);

    // // Decapsulation
    crypto_kem_dec(key_a, ct, sk);

    for(int o=0;o<CRYPTO_BYTES;++o) {
      if(key_a[o] != key_b[o]) {
        isWrong = 1;
      }
    }

    if(!isWrong)
    {
      *LED_ADDR = 40190031;
    }

    else
    {
      *LED_ADDR = 40190046;
    }
  }
```

Şekil 3.8: Ibex Üzerinde Çalıştırılan Algoritma.

Name	Value
IO_CLK	0
IO_RST_N	0
LED1[31:0]	0
clk_sys	0
rst_sys_n	0
instr_req	0
instr_gnt	0
instr_rvalid	0
instr_addr[31:0	00000080
instr_rdata[31:0	XXXXXXXX
data_req	0
data_gnt	0
data_rvalid	0
data_we	0
data_be[3:0]	f
data_addr[31:0	00000000
data_wdata[31:	00000000
data_rdata[31:0	XXXXXXXX
mem_addr[31:0	00000000
mem_req	0
mem_write	0
mem_be[3:0]	0

Şekil 3.9: Algoritmanın Ibex Üzerinde Çalıştırılmasıyla Elde Edilen Simülasyon Sonucu.

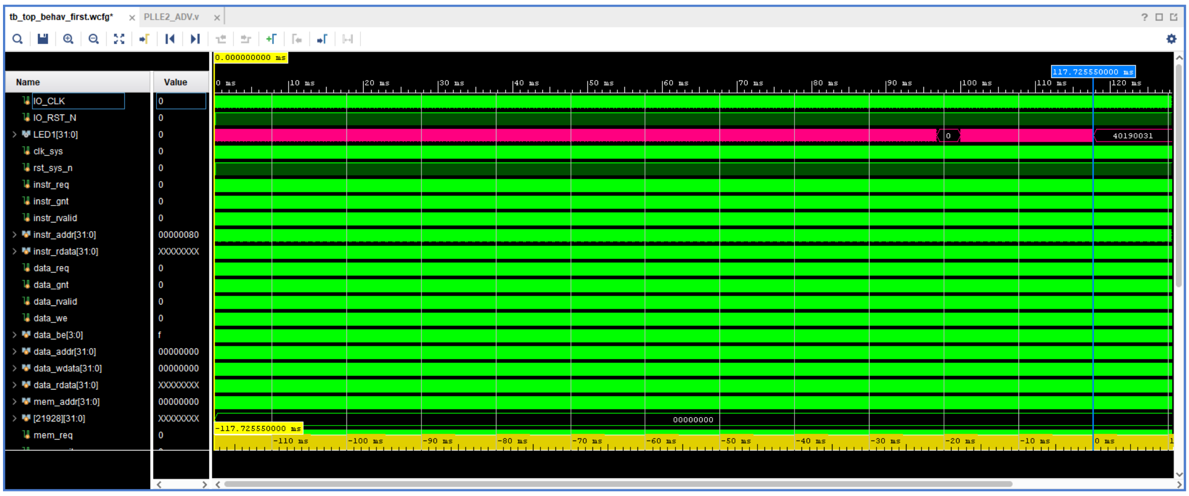
Simülasyon sonucunda görüldüğü üzere LED adresi üzerinde 40190031 sayısı elde edilmiş ve algoritmanın doğru bir şekilde Ibex işlemci üzerinde çalıştırılabildiği sonucuna varılmıştır.

Bu test yapılırken Ibex çekirdeği 20 ns’lik periyoda sahip saat işareti ile çalıştırılmıştır. Bitlerin karşılaştırılması işlemi dahil 339,425590 ms süren simülasyon sonucunda tüm işlemin 16.971.279,5 adet saat işareti kadar sürdüğü sonucuna varılmıştır.

Algoritma içerisinde üç temel fonksiyondan ilki olan crypto_kem_keypair() fonksiyonu 4.702.047,5 adet saat işareti kadar zaman almaktadır ve sonucu Şekil 3.10’da verilmiştir.

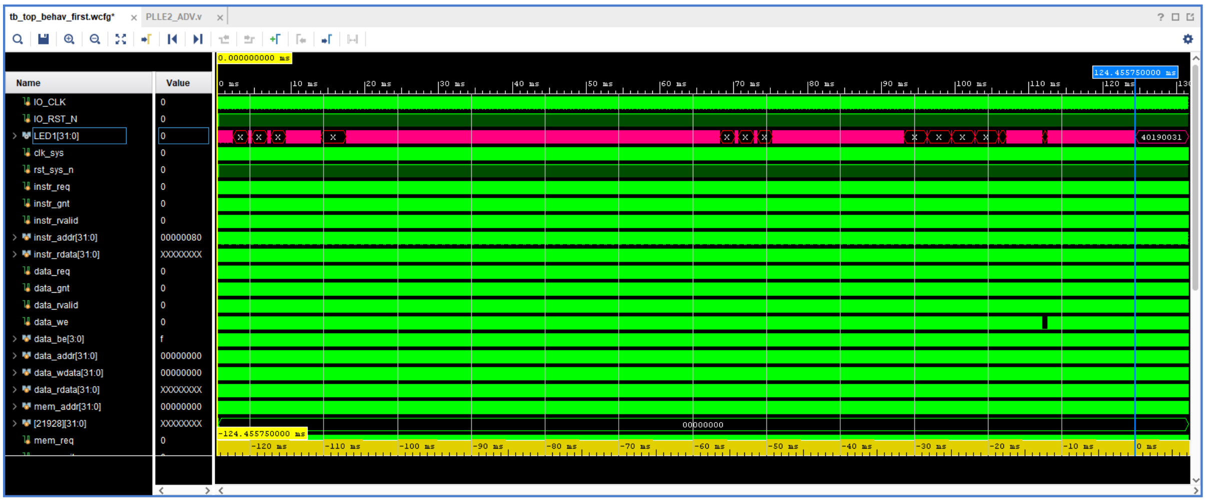
Şekil 3.10: crypto_kem_keypair() Fonksiyonunun Aldığı Süre

Algoritma içerisinde üç temel fonksiyondan ikincisi olan crypto_kem_enc() fonksiyonu 5.886.277,5 adet saat işareti kadar zaman almaktadır ve sonucu Şekil 3.11’de verilmiştir.



Şekil 3.11: crypto_kem_enc() Fonksiyonunun Aldığı Süre.

Algoritma içerisinde üç temel fonksiyondan üçüncüsü olan crypto_kem_dec() fonksiyonu 6.222.787,5 adet saat işareti kadar zaman almaktadır ve sonucu Şekil 3.12'de verilmiştir.



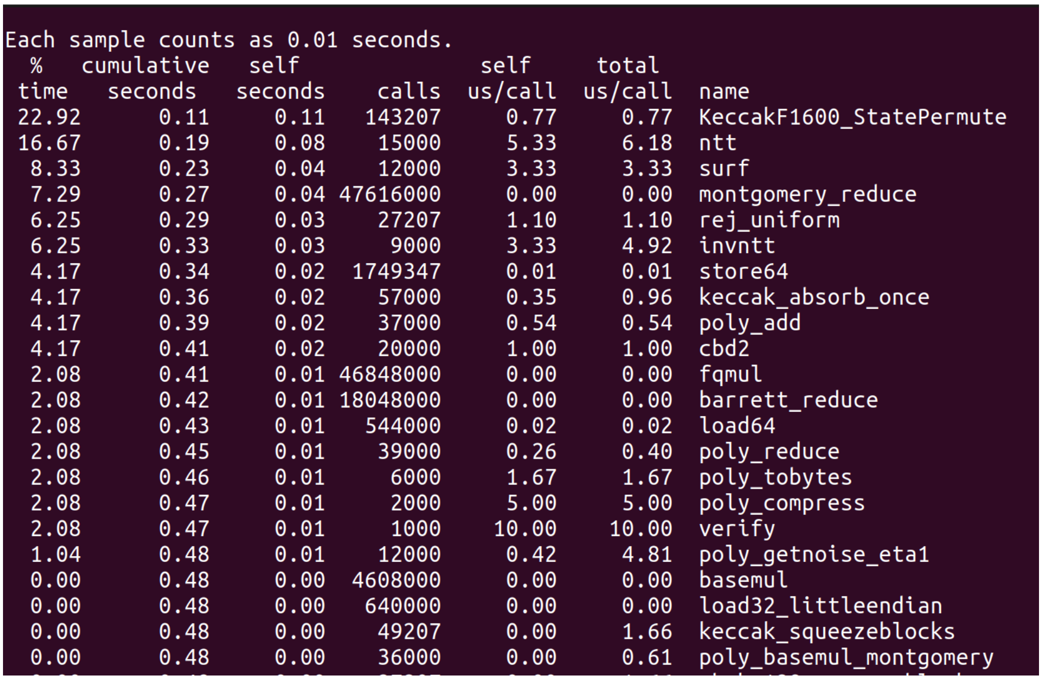
Name	Value
IO_CLK	0
IO_RST_N	0
LED1[31:0]	0
clk_sys	0
rst_sys_n	0
instr_req	0
instr_gnt	0
instr_rvalid	0
instr_addr[31:0]	00000080
instr_rdata[31:0]	XXXXXXXX
data_req	0
data_gnt	0
data_rvalid	0
data_we	0
data_be[3:0]	f
data_addr[31:0]	00000000
data_wdata[31:0]	00000000
data_rdata[31:0]	XXXXXXXX
mem_addr[31:0]	00000000
[21928][31:0]	XXXXXXXX
mem_req	0

Şekil 3.12: crypto_kem_dec() Fonksiyonun Aldığı Süre.

Böylece algoritma hem bütün olarak hem de üç ana alt fonksiyon olarak Ibex işlemci üzerinde çalıştırılmış ve analizleri yapılmıştır.

3.4 CRYSTALS-Kyber Algoritması için Kod Profilleme Yapılması

Ibex işlemci üzerinde çalıştırılan CRYSTALS-Kyber algoritması için hızlandırıcı donanım tasarımı yapılacaktır. Algoritma içerisinde gerçekleştirilen işlemlerden hangisinin donanımının tasarlanması en fazla faydayı getirecekse o işlemi yapan fonksiyonun donanımının tasarlanması en mantıklı iş olacaktır. Bu fonksiyona karar verebilmek içinse hangi fonksiyonun en çok çağrıldığının ve/veya en çok zaman aldığının bulunması gereklidir. Bu işlem kod profilleme ile yapılır. Kod profilleme işlemi için GNU gprof aracı kullanılmıştır. gprof, halihazırda RISC-V desteğine sahip değildir fakat yine de profilleme amacı için mantıklı sonuçlar vermektedir [16]. 3.2 bölümünde paylaşılan makefile dosyasının içerisinde tanımlanan komut ile kod profilleme işlemi yapılabilir. Profilleme sonucu Şekil 3.13'te verilmiştir.

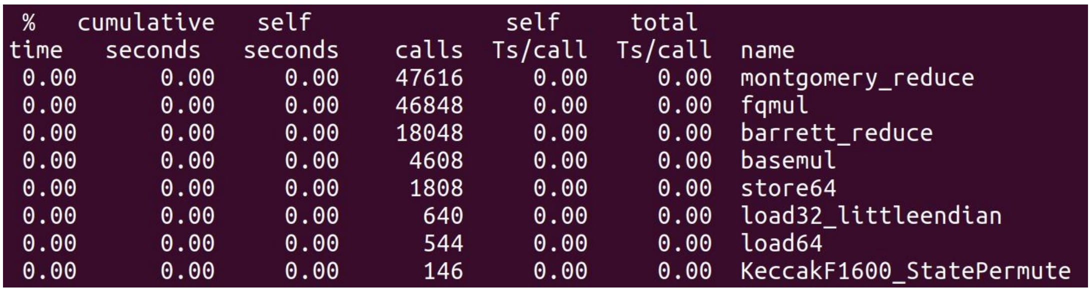


```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  us/call  us/call  name
 22.92      0.11     0.11   143207     0.77     0.77  KeccakF1600_StatePermute
 16.67      0.19     0.08    15000     5.33     6.18  ntt
  8.33      0.23     0.04    12000     3.33     3.33  surf
  7.29      0.27     0.04 47616000     0.00     0.00  montgomery_reduce
  6.25      0.29     0.03    27207     1.10     1.10  rej_uniform
  6.25      0.33     0.03     9000     3.33     4.92  invntt
  4.17      0.34     0.02  1749347     0.01     0.01  store64
  4.17      0.36     0.02    57000     0.35     0.96  keccak_absorb_once
  4.17      0.39     0.02    37000     0.54     0.54  poly_add
  4.17      0.41     0.02    20000     1.00     1.00  cbd2
  2.08      0.41     0.01 46848000     0.00     0.00  fqmul
  2.08      0.42     0.01 18048000     0.00     0.00  barrett_reduce
  2.08      0.43     0.01   544000     0.02     0.02  load64
  2.08      0.45     0.01    39000     0.26     0.40  poly_reduce
  2.08      0.46     0.01     6000     1.67     1.67  poly_tobytes
  2.08      0.47     0.01     2000     5.00     5.00  poly_compress
  2.08      0.47     0.01     1000    10.00    10.00  verify
  1.04      0.48     0.01    12000     0.42     4.81  poly_getnoise_eta1
  0.00      0.48     0.00  4608000     0.00     0.00  basemul
  0.00      0.48     0.00   640000     0.00     0.00  load32_littleendian
  0.00      0.48     0.00    49207     0.00     1.66  keccak_squeezeblocks
  0.00      0.48     0.00    36000     0.00     0.61  poly_basemul_montgomery
```

Şekil 3.13: Kod Profilleme İşleminin Sonucunda Elde Edilen Çıktı.

CRYSTALS-Kyber algoritması kendi içerisinde rastgelelik de barındırdığı için bu işlemi bir kez yapmak yanıltıcı sonuçlar doğurabilir [17]. Bu sebeple algoritma döngü içerisinde 1000 defa çalıştırılarak profilleme sonucu Şekil 3.13'teki gibi elde edilmiştir. Bu sonuca göre KeccakF1600_StatePermute fonksiyonu algoritma içerisinde en çok zaman harcanan fonksiyondur.

Fonksiyonların çağrılma sayısının net olarak saptanabilmesi için bu kez algoritma döngü içerisinde değil bir kez çalıştırılarak tekrar kod profilleme işlemi yapılmıştır. Bu işlemin çıktısı ise Şekil 3.14'te verilmiştir.

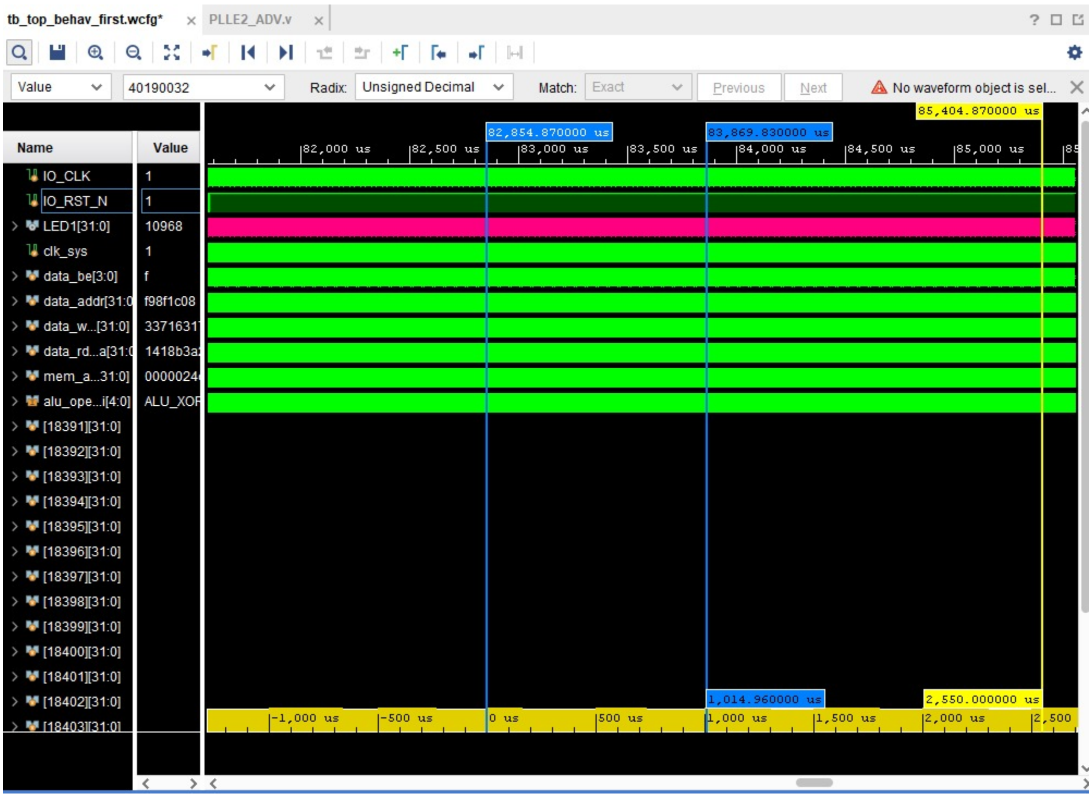


```
 %   cumulative   self              self     total
time   seconds   seconds    calls  Ts/call  Ts/call  name
 0.00      0.00     0.00    47616     0.00     0.00  montgomery_reduce
 0.00      0.00     0.00    46848     0.00     0.00  fqmul
 0.00      0.00     0.00    18048     0.00     0.00  barrett_reduce
 0.00      0.00     0.00     4608     0.00     0.00  basemul
 0.00      0.00     0.00     1808     0.00     0.00  store64
 0.00      0.00     0.00      640     0.00     0.00  load32_littleendian
 0.00      0.00     0.00      544     0.00     0.00  load64
 0.00      0.00     0.00      146     0.00     0.00  KeccakF1600_StatePermute
```

Şekil 3.14: Tek Seferlik Çalıştırma ile Kod Profilleme İşleminin Çıktısı.

Bu sonuca göre KeccakF1600_StatePermute işlemi algoritma içerisinde en çok zaman alan ve test_vectors512 içerisinde 146 defa çağrılan bir fonksiyondur.

Bu işlemin Ibex çekirdeği üzerinde ne kadar zaman aldığını görmek için bunu saptayan simülasyon tekrar yapılmıştır ve sonuc Şekil 3.15 ile verilmiştir.



Value: 40190032

Radix: Unsigned Decimal

Match: Exact

Name	Value
IO_CLK	1
IO_RST_N	1
LED1[31:0]	10968
clk_sys	1
data_be[3:0]	f
data_addr[31:0	f98f1c08
data_w...[31:0]	3371631
data_rd...a[31:0	1418b3a
mem_a...31:0]	0000024
alu_ope...i[4:0]	ALU_XOF
[18391][31:0]	
[18392][31:0]	
[18393][31:0]	
[18394][31:0]	
[18395][31:0]	
[18396][31:0]	
[18397][31:0]	
[18398][31:0]	
[18399][31:0]	
[18400][31:0]	
[18401][31:0]	
[18402][31:0]	
[18403][31:0]	

Şekil 3.15: KeccakF1600_StatePermute Fonksiyonunun Ibex Çekirdeği Üzerinde Aldığı Süre.

Simülasyon sonucuna göre KeccakF1600_StatePermute fonksiyonu (Şekil 3.15’te iki mavi çizgi arası) Ibex çekirdeği üzerinde 50.748 saat işareti kadar zaman almaktadır, hesaplama yine 20 ns saat periyodu üzerinden yapılmıştır. Bu işlem bir anahtar çifti üretme, enkapsülleme ve dekapsülleme işlemi için 146 kez yapıldığında 146 x 50,748 = 7.409.208 adet saat işareti kadar zaman alacaktır.

Bu saat işareti sayısı bir işlemin yaklaşık 7.409.208 / 16.811.112,5 ≅ %44’üne denk gelmektedir. Dolayısıyla bu fonksiyonda donanım yardımıyla yapılabilecek bir hızlandırma, algoritmanın tamamının önemli bir ölçüde hızlandırılması anlamına gelecektir.

4. KECCAKF1600 FONKSİYONU

SHA-3 (Güvenli Karma Algoritması 3) standardının temelini oluşturan keccak, Amerika Birleşik Devletleri'ndeki Ulusal Standartlar ve Teknoloji Enstitüsü (NIST) tarafından düzenlenen açık bir yarışma sırasında geliştirilmiş bir kriptografik karma (hash) fonksiyonudur. 2012 NIST yarışmasının galibi olan Keccak algoritması, SHA-2 karma fonksiyonları ailesinin yerine geçerek yeni SHA-3 standardı olarak seçilmiştir.

Kriptografide temel araçlar olan karma fonksiyonlar, parola karma, dijital imzalar ve mesaj bütünlüğü kontrolleri gibi birçok güvenlik uygulamasında kullanılır. Bir kriptografik karma fonksiyonuna girdi verildiğinde, genellikle "mesaj" olarak adlandırılan bu girdi, girdiye özgü sabit uzunlukta bir bayt dizisi olan bir özet üretir. Bu işlemin tersine çevrilmesi, yani karmadan orijinal mesajı elde etmek ya da aynı karmayı oluşturan iki farklı mesaj bulmak hesaplama açısından imkansız olmalıdır. Bu çıktının rastgele görünmesi beklenir.

4.1 Keccak Nedir?

Keccak, "sünger yapısı" adı verilen özgün bir metot kullanmasıyla öne çıkar. Bu teknik, SHA-1 ve SHA-2 gibi önceki karma algoritmalarında kullanılan Merkle-Damgård yapısından farklıdır. Giriş verileri sünger yapısı tarafından doğal haliyle emilir, işlenir ve sonrasında karma çıkarılır. Bu yöntem, güvenliği artırır ve karmaların çıktı uzunluğuna önemli ölçüde esneklik kazandırır.

Keccak'ın sünger yapısı, giriş verilerini kapsamlı bir şekilde karıştıran bir permütasyon fonksiyonuna dayanır. Bu fonksiyonun geri bildirim döngüsünde tekrar tekrar uygulanması yüksek güvenlik sağlar. Keccak tasarımı, özellikle donanım uygulamalarında sadeliği ve verimliliğiyle dikkat çekicidir.

4.2 KeccakF1600 Fonksiyonu

Keccak karma fonksiyonları ailesinin en önemli üyesi KeccakF1600 fonksiyonudur. Bu, 1600 bitlik bir bit dizisi üzerinde çalışan bir permütasyon fonksiyonudur. Keccak dilinde bu dizilere "durum" denir. KeccakF1600 fonksiyonu bu durumu dönüştürerek gelen verilerin iyi karıştırılmasını sağlar.

KeccakF1600 fonksiyonunun gerçekleştirdiği işlemler, bir dizi turdan oluşur. Her turda şu adımlar yer alır:

- **Theta (θ)**: Karıştırma işleminde, durumdaki her bit, kendi sütunu ve yanındaki iki sütundaki bitlerin bir fonksiyonu ile XOR edilir.
- **Rho (ρ)**: Durumdaki her bir şerit, belirli bir ofsetle döndürülerek Rho (ρ) adı verilen bir bit döndürme adımını içerir.
- **Pi (π)**: Durumun şerit yapılandırması, Pi (π) adı verilen permütasyon adımıyla yeniden düzenlenir.
- **Chi (χ)**: Her bit, aynı sıradaki iki diğer bitin bir fonksiyonu kullanılarak bir XOR işlemine tabi tutulur.
- **Iota (ι)**: Iota (ι) adı verilen enjeksiyon adımında, bir tur sabiti duruma XOR edilir.

Algoritmanın C kodu Ek A'da verilmiştir.

4.3 Keccak Read Unit

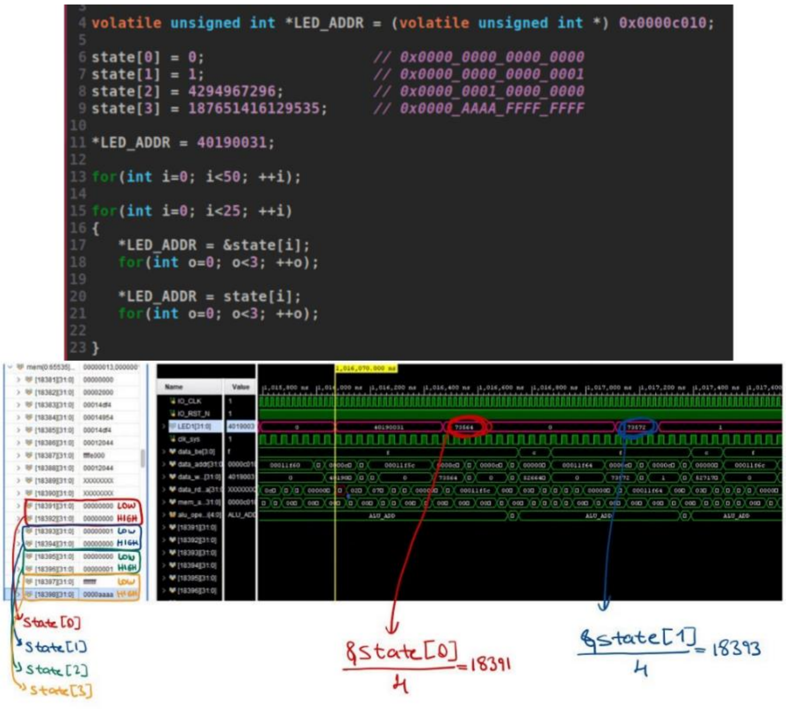
Bu modül sayesinde KeccakF1600 donanımı RAM Bloğuna direkt erişim özelliği kazanmaktadır. Keccak fonksiyonu girişinde 25x64 bitlik data alır. Bu data hafızaya State Pointerından itibaren 50x32 bitlik bir blok halinde yazılır. IBEX bu datayı (State Pointerı+0x00) ile (State Pointerı+0x3F) adresleri arasına yazar.

1600 bit boyutuna sahip Data girişi ve çıkışına sahip bu modül "keccak_done" ve "keccak_en" sinyalleri ile kontrol edilir. "keccak_en"sinyali geldiğinde, IBEX'ten gelen başlangıç adresinden itibaren gelen 1600 bitlik datayı "keccak_input" adlı çıkıştan gönderir. Benzer şekilde "keccak_done" sinyali geldiğinde, yine aynı adrese "after" girişinden gelen datayı yazar. Bu yöntem sayesinde RAM'de belirli bir bölgeyi Keccak modülünün kullanması için rezerve edilir ve Keccak modülüne RAM'e direkt erişim özelliği sağlanır.

4.4 Tasarlanan Donanım: KeccakF1600_StatePermute

Tasarlanan donanımda kullanılan kodda KeccakF1600_StatePermute fonksiyonu 64 bitlik 25 boyutlu bir dizi aldığı ve projede kullanılan işlemci mimarisi 32 bitlik olduğu için bitler alt32-üst32 mi üst32-alt32 şeklinde mi ayrıldığından emin olunmalıdır.

Örneğin 0xAAAA_AAA_BBBB_BBBB şeklinde 64-bitlik bir sayının hafızada 0xAAAA_AAAA'dan sonra 0xBBBB_BBBB'nin gelmesi ve 0xBBBB_BBBB'den sonra 0xAAAA_AAAA'nın gelmesi donanımın nasıl çalışması gerektiğini değiştirir. Bunu denemek için Şekil 4.1'de verilen kod ve simülasyon kullanılmıştır.

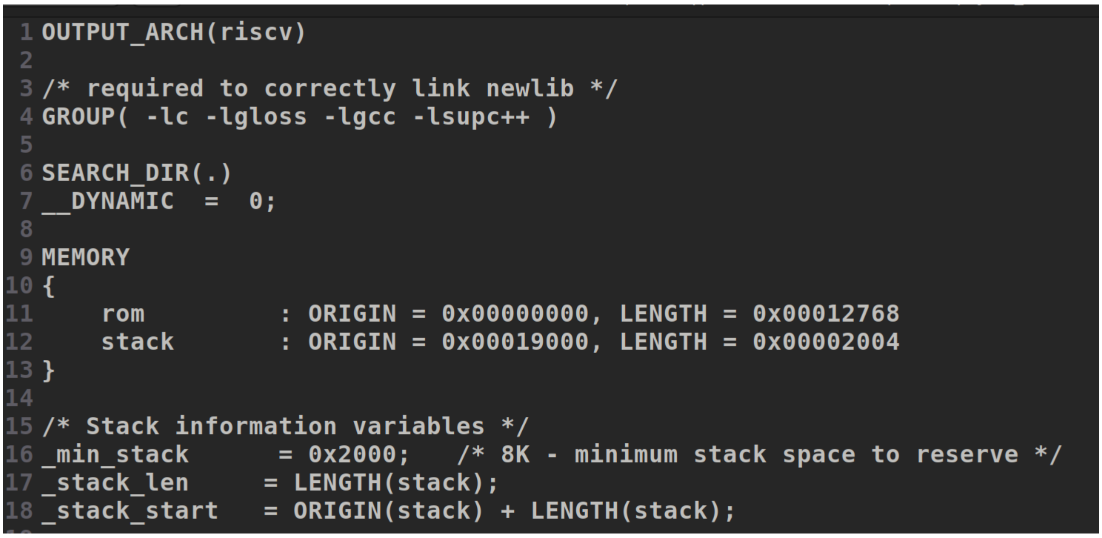


```
volatile unsigned int *LED_ADDR = (volatile unsigned int *) 0x0000c010;

state[0] = 0;                   // 0x0000_0000_0000_0000
state[1] = 1;                   // 0x0000_0000_0000_0001
state[2] = 4294967296;          // 0x0000_0001_0000_0000
state[3] = 187651416129535;     // 0x0000_AAAA_FFFF_FFFF

*LED_ADDR = 40190031;

for(int i=0; i<50; ++i);

for(int i=0; i<25; ++i)
{
   *LED_ADDR = &state[i];
   for(int o=0; o<3; ++o);

   *LED_ADDR = state[i];
   for(int o=0; o<3; ++o);

}
```

Şekil 4.1: Ibex Çekirdeğinin Little Endian ya da Big Endian Yapıya Sahip Olduğunun Belirlenmesi.

Simülasyon sonucuna bakarak önce en anlamsız (least significant) daha sonra en anlamlı (most significant) bitlerin hafızaya kaydedildiği görülmüştür.

Tasarlanan donanım ile haberleşebilmek için ayrılan adresler 0x14000-0x14010 olmuştur. Bundan dolayı bu adreslerin arasının kod tarafından buyruk (instruction) veya veri (data) hafızaları tarafından kullanılmaması gerekmektedir. Bu kullanımı engellemek için compiler için kullanılan link.ld dosyası aşağıdaki gibi modifiye edilmiştir.

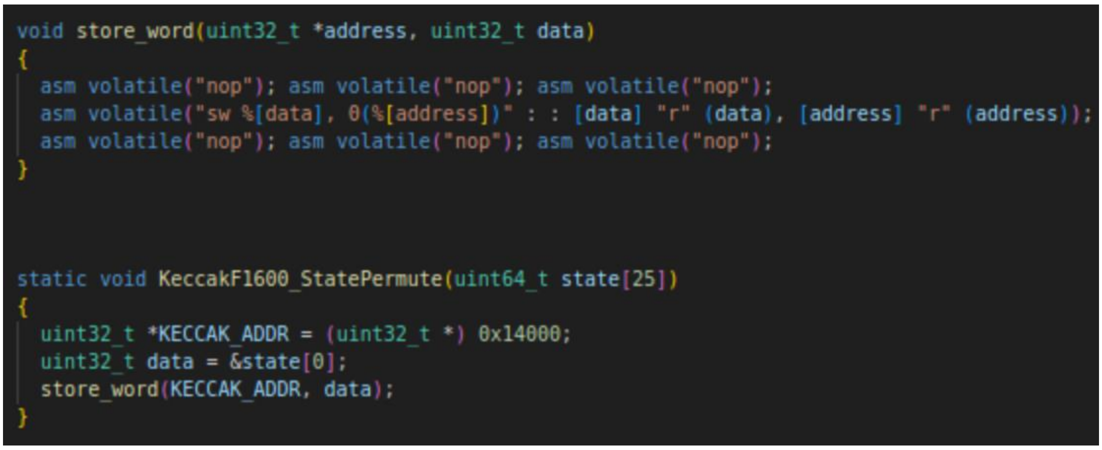


```
OUTPUT_ARCH(riscv)

/* required to correctly link newlib */
GROUP( -lc -lgloss -lgcc -lsupc++ )

SEARCH_DIR(.)
__DYNAMIC  =  0;

MEMORY
{
    rom         : ORIGIN = 0x00000000, LENGTH = 0x00012768
    stack       : ORIGIN = 0x00019000, LENGTH = 0x00002004
}

/* Stack information variables */
_min_stack      = 0x2000;   /* 8K - minimum stack space to reserve */
_stack_len     = LENGTH(stack);
_stack_start   = ORIGIN(stack) + LENGTH(stack);
```

Şekil 4.2: İşlemci Hafızasında KeccakF1600 Donanımının Kullanabileceği Hafızanın Ayrılması.

Bu modifikasyon sonucunda 0x00000-0x12768 arası roma ayrılırken 0x19000'den başlamak üzere 0x2004 uzunluklu bir stack düzenlenmiştir. Böylece 0x14000-0x14010 adresleri kod tarafından kullanılmayacaktır.

Bütün bu modikifasyonlar ve donanımın işlemci mimarisine entegrasonu sonucunda bu donanımı uyaracak olan yazılımsal kod aşağıdaki gibi düzenlenmiştir.



```
void store_word(uint32_t *address, uint32_t data)
{
  asm volatile("nop"); asm volatile("nop"); asm volatile("nop");
  asm volatile("sw %[data], 0(%[address])" : : [data] "r" (data), [address] "r" (address));
  asm volatile("nop"); asm volatile("nop"); asm volatile("nop");
}



static void KeccakF1600_StatePermute(uint64_t state[25])
{
  uint32_t *KECCAK_ADDR = (uint32_t *) 0x14000;
  uint32_t data = &state[0];
  store_word(KECCAK_ADDR, data);
}
```

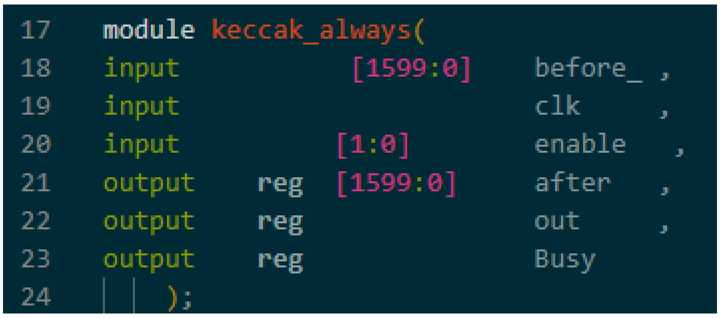
Şekil 4.3: KeccakF1600_StatePermute Fonksiyonunun Ibex Çekirdeği Üzerinde Aldığı Süre

Bu kod ile birlikte Ek’te verilen uzun KeccakF1600_StatePermute fonksiyonu sadece 3 satırlık bir fonksiyona dönüştürülmüştür. Bu fonksiyonun içerisinde çağrılan store_word fonksiyonu ise KECCAK_ADDR ile belirlenen Keccak donanımının adresine state pointerını yazar. Bu pointerın değerinin Keccak donanımının bölgesine yazılmasıyla birlikte işlemci buyruk alma işlemini beklemeye alır. Keccak Read Unit vasıtasıyla hazırlanan 1600-bitlik girdi Keccak donanımına verilir. 25 saat işaretlik bir zamandan sonra Keccak karma (hash) işlemi sona erer ve veriler, hafızadan geldikleri bölgeye geri yazılır. Verilerin geri yazılmasından sonra işlemci tekrardan buyruk çekmeye devam ederek kaldığı yerden çalışmaya devam eder. Böylece yazılım ile tanımlanan uzun, fazla yer kaplayan ve çok uzun zaman alan Keccak fonksiyonu donanım ile aynı fonksiyonalite ile çalışarak daha kısa bir sürede işlevini yerine getirmiş olur.

KeccakF1600_StatePermute fonksiyonu CRYSTALS-Kyber algoritmasında olduğu gibi birçok farklı uygulamada da kullanılmaktadır. Bu sebeple donanım tasarımında izlenen yol ilk önce daha önce bu donanım için tasarlanmış olan donanımları incelenmesi ve tasarım metodlarının keşfedilmesi olmuştur.

Yaklaşım hem yeni bir KeccakF1600_StatePermute donanımı oluşturmak hem de araştırmalar sonucu elde edilen donanım bloklarının Vivado geliştirme ortamında test edilerek karşılaştırmalarının yapılması olarak belirlenmştir.

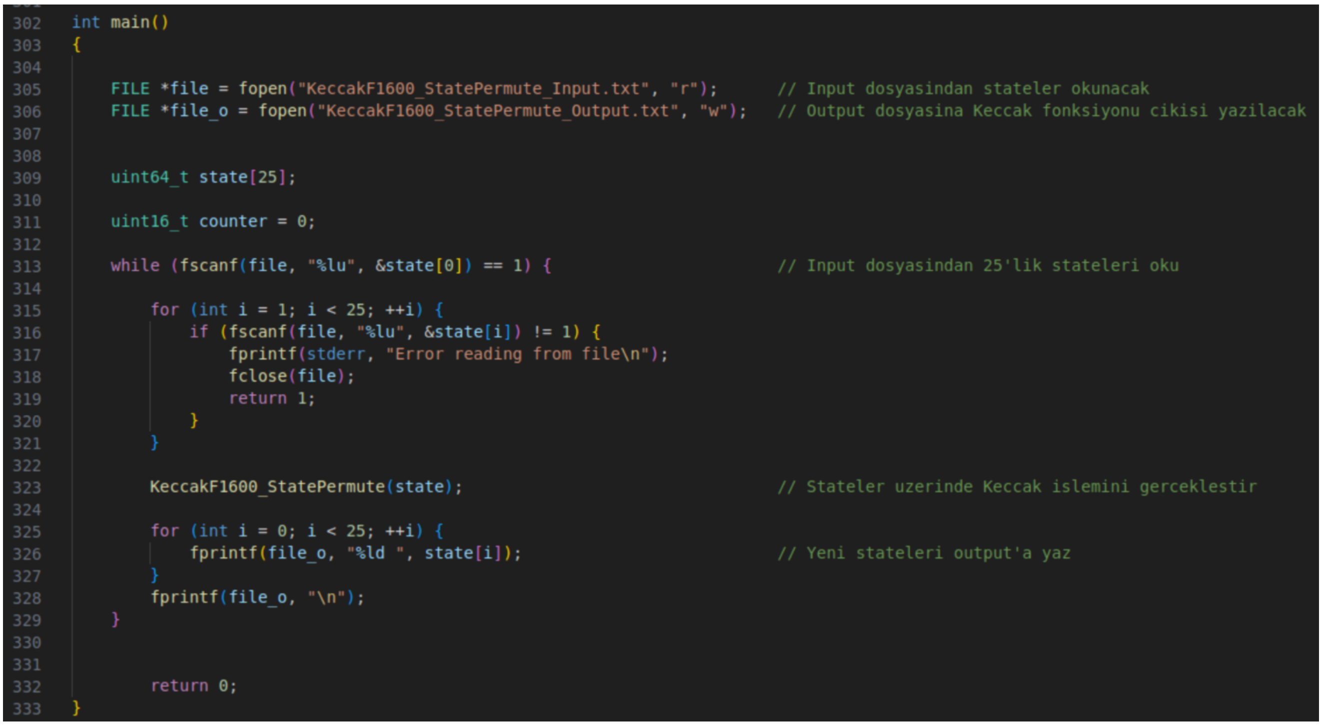
Oluşturulacak olan donanım Ibex RISC-V işlemci çekirdeğine entegre edileceği için, yeni tasarlanacak KeccakF1600_StatePermute donanımında öncelik olarak çekirdeğe doğru şekilde entegre olması için gerekli koşullar sağlanmıştır. Giriş çıkışlar düzenlenmiş ve donanım tasarımı KeccakF1600_StatePermute fonksiyonunun C dilinde yazılmış algoritması referans alınarak davranışsal olarak doğru çıktıyı vermesi amaçlanarak tasarıma başlanmıştır.



```
module keccak_always(
input                [1599:0]    before_ ,
input                            clk     ,
input               [1:0]        enable   ,
output      reg     [1599:0]     after   ,
output      reg                  out     ,
output      reg                  Busy
        );
```

Şekil 4.4: KeccakF1600 donanımı için belirlenen giriş ve çıkışlar.

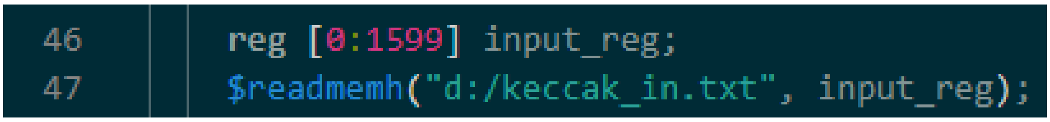
Şekil 4.4’te yazılımsal olarak bu fonksiyon ayrıştırılarak belirli bir giriş verisi oluşturulmuş ve bu veriye karşı KeccakF1600_StatePermute fonksiyonunun çıktısı kaydedilmiştir. Öncelikli amaç donanımda aynı giriş verisine karşın yazılımdan elde edilenle aynı çıkış verisini elde etmek olarak belirlenmiştir. Bu işlem sonrası hem yapılan donanım bloğunun RISC-V işlemci çekirdeğine entegrasyon aşamasına geçilebilecek hem de tasarlanan KeccakF1600_StatePermute donanımının iyileştirilmesine devam edilebilecektir.



```
int main()
{

    FILE *file = fopen("KeccakF1600_StatePermute_Input.txt", "r");      // Input dosyasindan stateler okunacak
    FILE *file_o = fopen("KeccakF1600_StatePermute_Output.txt", "w");   // Output dosyasina Keccak fonksiyonu cikisi yazilacak


    uint64_t state[25];

    uint16_t counter = 0;

    while (fscanf(file, "%lu", &state[0]) == 1) {                         // Input dosyasindan 25'lik stateleri oku

        for (int i = 1; i < 25; ++i) {
            if (fscanf(file, "%lu", &state[i]) != 1) {
                fprintf(stderr, "Error reading from file\n");
                fclose(file);
                return 1;
            }
        }

        KeccakF1600_StatePermute(state);                                  // Stateler uzerinde Keccak islemini gerceklestir

        for (int i = 0; i < 25; ++i) {
            fprintf(file_o, "%ld ", state[i]);                            // Yeni stateleri output'a yaz
        }
        fprintf(file_o, "\n");
    }


        return 0;
}
```

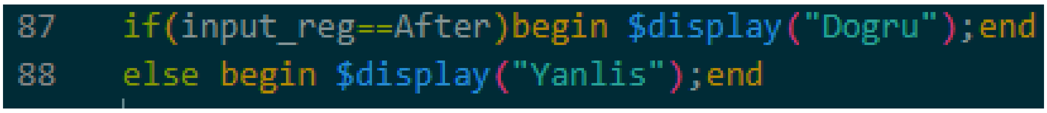
Şekil 4.5: Keccak Giriş/Çıkışının Yazılım ile Elde Edilmesi.

Donanım için oluşturulmuş test dosyasında, yazılımdan elde edilen giriş donanım bloğuna verilerek çıktı karşılaştırılması yapılarak kontrolü sağlanmıştır.



```
reg [0:1599] input_reg;
$readmemh("d:/keccak_in.txt", input_reg);
```

Şekil 4.6: Test Dosyasına yazılımsal olarak elde edilen verinin yüklenmesi.



```
if(input_reg==After)begin $display("Dogru");end
else begin $display("Yanlis");end
```

Şekil 4.7: Donanım Sonucu oluşan çıktının kontrol edilmesi.

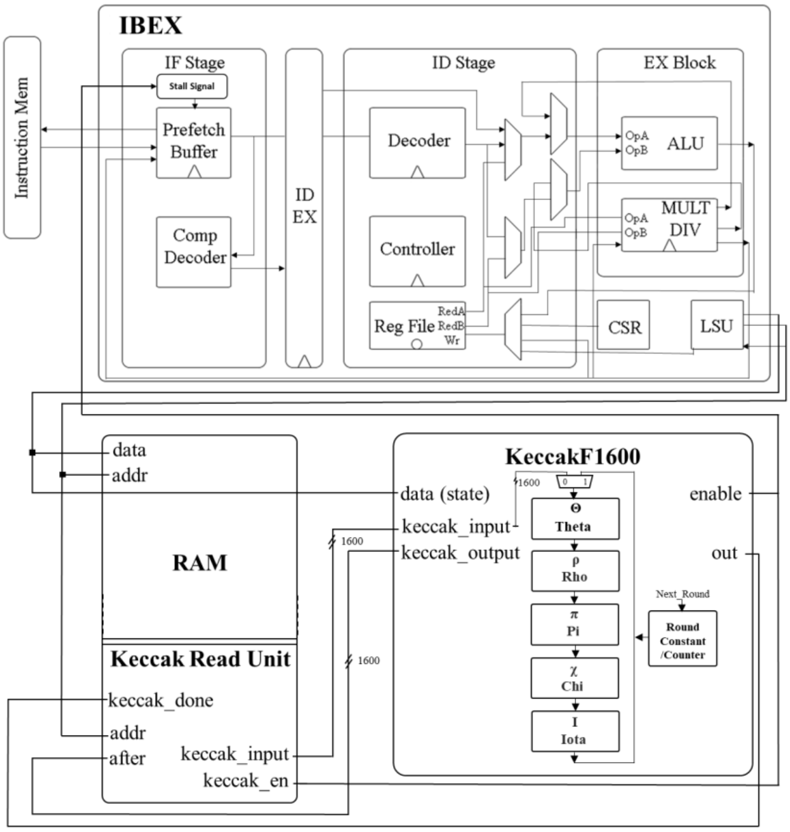
C ile yazılmış olan ardışık işlenen algoritma, durumlara bölünerek her durumda değişmesi gereken kayıtçılar(register) ayırt edilerek tanımlamaları yapılmıştır. Bu şekilde öncelikle 10 durumlu bir donanım elde edilmiştir.

KeccakF1600_StatePermute fonksiyonunun yazılımında olduğu gibi her round için hesaplanmış sabitler donanımda da kaydedilmiştir. Yeterli round için durumlar gerçekleştiğinde donanım çıktı sinyali 1 yapılarak fonksiyonun tamamladığı belirtilmiştir.

Öte yandan KeccakF1600_StatePermute fonksiyonu için altarnatif donanımların araştırılması sonucu, Keccak algoritmasının geliştiricisi olan TeamKeccak[6] grubnunun web sitelerinde referans gerçeklemesi bulunmuştur. Daha sonrasında farklı Keccak hızlandırıcı tasarımları incelendiğinde de temel olarak TeamKeccak grubuna ait referans gerçekleme kullanılarak farklı biçimlerde sistemlere entegre edildiği görülmüştür[4][18]. Vivado ortamında gerçeklenerek aynı test verisine karşı doğru çıktı alınmıştır. Hız parametresi ve entegrasyon için kullanışlılık açısından göz önünde bulundurularak karşılaştırılmıştır.

4.5 KeccakF1600 Donanımının RISC-V İşlemciye Entegrasyonu

Donanım entegrasyonu için KeccakF1600 donanımında belirlenen giriş ve çıkışlar IBEX işlemci çekirdeği ile uygun olarak eşleştirilmiştir. KeccakF1600 donanımının yanında Keccak Okuma Bloğu tasarlanmış ve Keccak için ayrılan Hafıza bloğu ile entegre edilmiştir. Böylece tüm donanım mimarisi oluşturulmuştur.



Şekil 4.8: RISC-V İşlemci ve KeccakF1600 Donanım Mimarisi.

Şekil 4.8’de oluşturulan donanım mimarisi görselleştirilmiştir ve bu mimari göz önünde bulundurularak entegrasyon tasarımı gerçeklenmiştir.

Keccak donanımı ile işlemci arasında herhangi bir Bus bağlantısı yoktur. İşlemci keccak donanımına veriyi aktarmak istediğinde öncelikle donanıma gidecek veri RAM belleğindeki belirlenmiş adreslere yazılır yazma işlemi bittikten sonra yine RAM belleğinde bulunan bir adresten Keccak donanımına”enable” sinyali verilir Bu sırada işlemciye stall sinyali gönderilerek işlemcinin başka işlere devam etmeden program sayacında sabit kalması sağlanır.

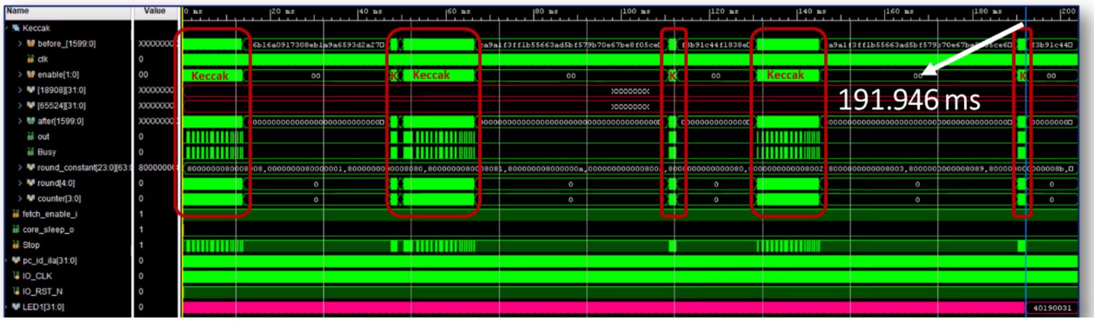
Keccak donanımının “enable” sinyali ile instruction geldikten sonra fetch aşamasında çekirdek durdurularak Keccak donanımının çalıştırılması sağlanmıştır. Bu aşamadan sonra ibex çekirdeğinde bir sonraki instructiona geçiş engellenmiş ve işletim durdurulmuştur.

Keccak donanımı aktif olduktan sonra, hafızada keccak için oluşturulmuş giriş verisine direkt erişmiştir. “Keccak_input” verisi keccak içerisinde bulunan Theta, Rho, Pi, Chi, Iota aşamalarından her round için geçerek çıktı üretmiş ve bu çıktı Keccak Okuma Ünitesi ile tekrar keccak için ayırılan hafıza bloğuna yazılmıştır. Enable sinyalinin güncellenmesi ile ibex çekirdeği işletimi bir sonraki instructiona geçirilmiştir. Bu yöntemle yazılımsal profilleme sonucu CRYSTALS-Kyber algoritması için en çok hızlandırılmaya müsait KeccakF1600 fonksiyonu ile IBEX RISC-V işlemci çekirdeği entegrasyonu sağlanmıştır.

5. SONUÇLAR

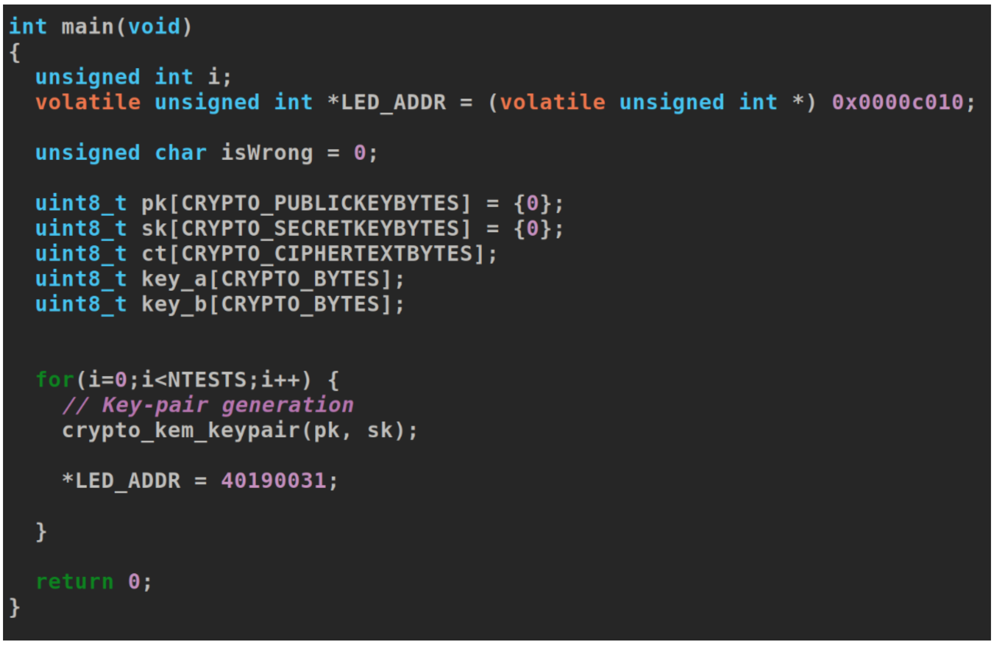
5.1 Algoritmanın Tamamı

KeccakF1600 fonksiyonu için yapılan donanım mimariye eklendikten sonra yine Xilinx Vivado ile simülasyon yapılmıştır. Simülasyon sonuçlarına göre daha önce 339,426 ms süren ve 50 MHz saat işareti frekansında 16.971.279 saat işareti süren işlem, donanım eklendikten sonra aynı saat frekansında 191,946 ms sürerek 9.538. 150 saat işareti sürecek kadar zaman almıştır. Bu sonuç 1,76'lık bir hızlandırma faktörüne denk gelmektedir.



Şekil 5.1: Algoritmanın Tamamının Simülasyon Ortamında Aldığı Süre.

5.2 Anahtar Çifti Üretimi

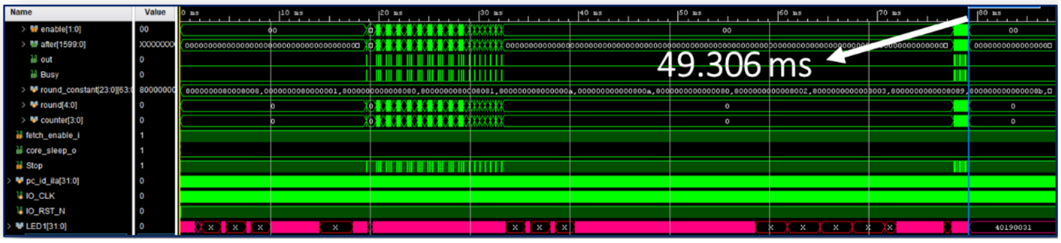


```
int main(void)
{
  unsigned int i;
  volatile unsigned int *LED_ADDR = (volatile unsigned int *) 0x0000c010;

  unsigned char isWrong = 0;

  uint8_t pk[CRYPTO_PUBLICKEYBYTES] = {0};
  uint8_t sk[CRYPTO_SECRETKEYBYTES] = {0};
  uint8_t ct[CRYPTO_CIPHERTEXTBYTES];
  uint8_t key_a[CRYPTO_BYTES];
  uint8_t key_b[CRYPTO_BYTES];


  for(i=0;i<NTESTS;i++) {
    // Key-pair generation
    crypto_kem_keypair(pk, sk);

    *LED_ADDR = 40190031;

  }

  return 0;
}
```

Şekil 5.2: Anahtar Üretme Fonksiyonunun Ne Kadar Sürdüğünü Gösteren Tetiklenme Kodu.

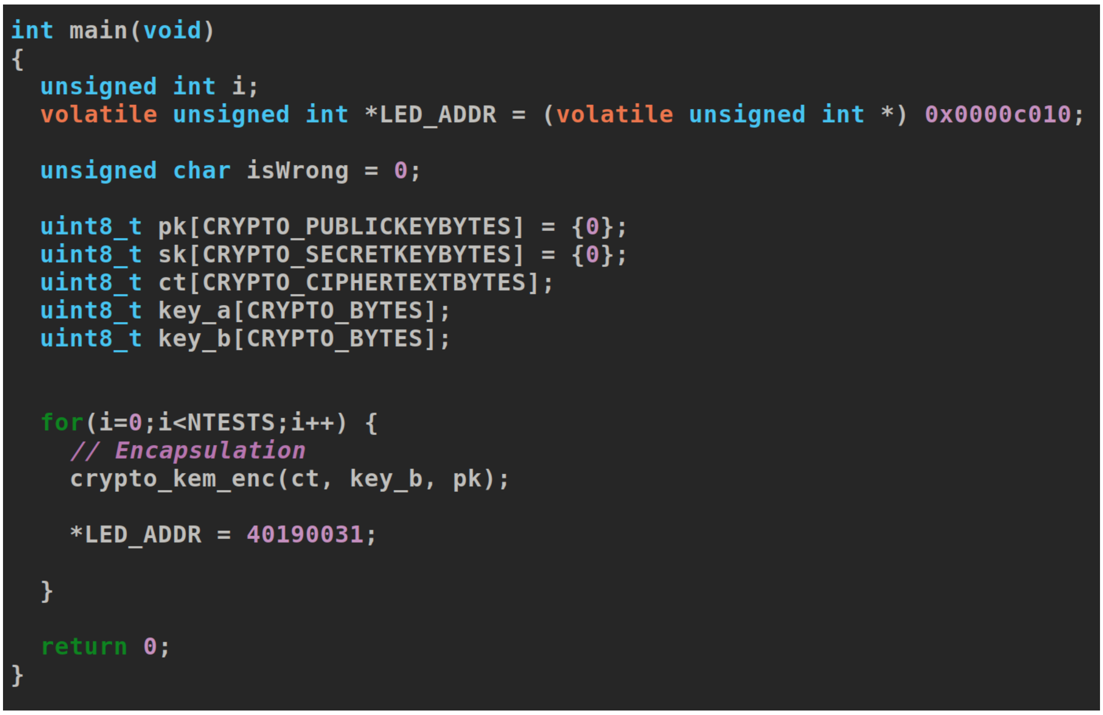
Anahtar üretme (keypair generation) süresinin ne kadar sürdüğünü anlamak için yukarıdaki kod parçası kullanılmıştır. Bu kısımda kodun tamamı doğru veriler ile çalışmadığı için doğru bir sonuç beklenmez, burada amaç sadece bu fonksiyonun ne kadar sürdüğünü anlamaktır.

Anahtar üretimi (keypair generation) kısmı kod içerisinde tek başına çalıştırılarak sadece bu işlemin ne kadar sürdüğünün simülasyonu yapıldığında 1,91'lik bir hızlandırma faktörü ile sürenin 4.702.047 saat işaretinden 2.465.323 saat işaretine düştüğü gözlemlenmiştir.



Şekil 5.3: Anahtar Üretme Fonksiyonunun Simülasyon Ortamında Aldığı Süre.

5.3 Kapsülleme

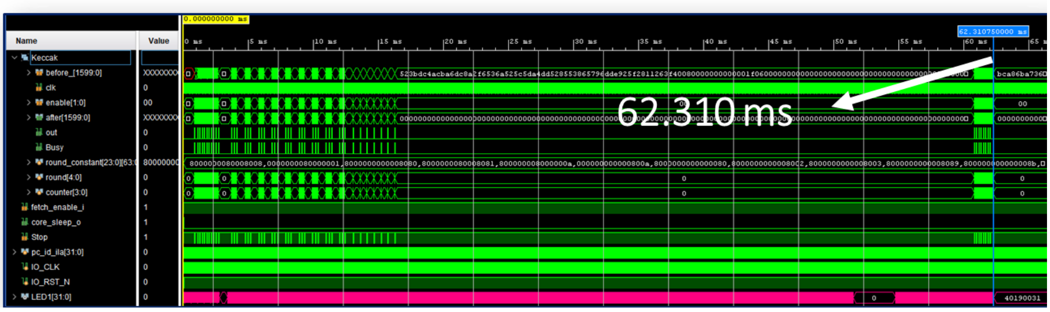


```
int main(void)
{
  unsigned int i;
  volatile unsigned int *LED_ADDR = (volatile unsigned int *) 0x0000c010;

  unsigned char isWrong = 0;

  uint8_t pk[CRYPTO_PUBLICKEYBYTES] = {0};
  uint8_t sk[CRYPTO_SECRETKEYBYTES] = {0};
  uint8_t ct[CRYPTO_CIPHERTEXTBYTES];
  uint8_t key_a[CRYPTO_BYTES];
  uint8_t key_b[CRYPTO_BYTES];


  for(i=0;i<NTESTS;i++) {
    // Encapsulation
    crypto_kem_enc(ct, key_b, pk);

    *LED_ADDR = 40190031;

  }

  return 0;
}
```

Şekil 5.4: Şifreleme Fonksiyonunun Ne Kadar Sürdüğünü Gösteren Tetiklenme Kodu.

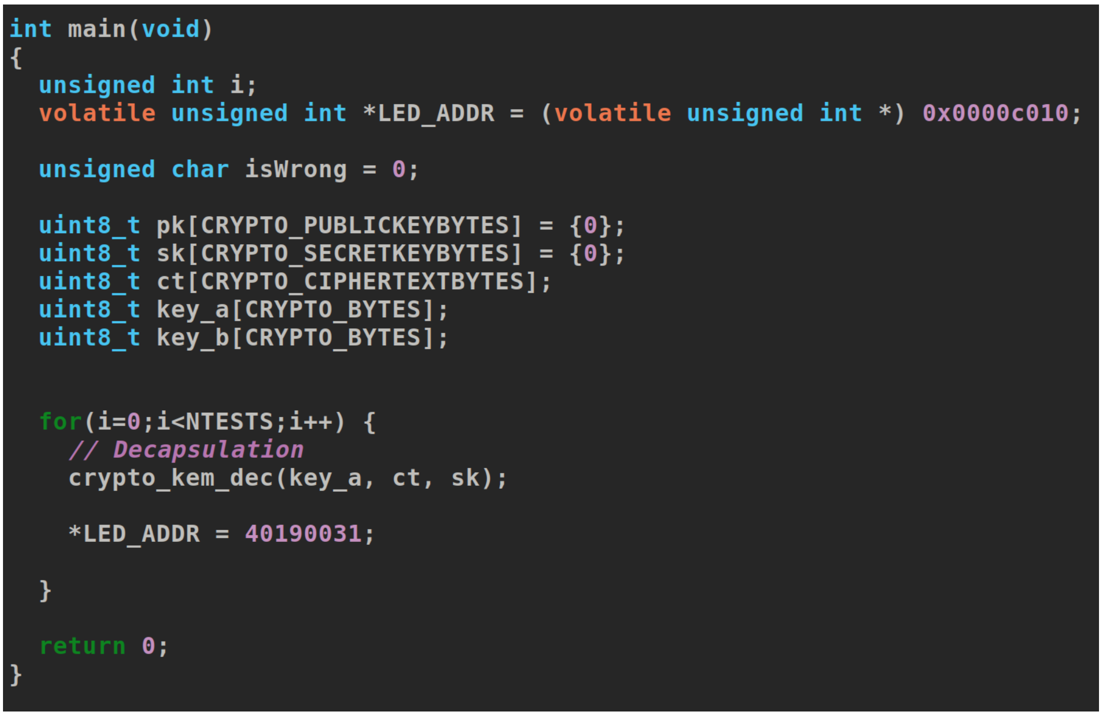
Veri şifreleme (enkapsülleme, encapsulation) süresinin ne kadar sürdüğünü anlamak için yukarıdaki kod parçası kullanılmıştır. Bu kısımda kodun tamamı doğru veriler ile çalışmadığı için doğru bir sonuç beklenmez, burada amaç sadece bu fonksiyonun ne kadar sürdüğünü anlamaktır.

Enkapsülleme (encapsulation) kısmı kod içerisinde tek başına çalıştırılarak sadece bu işlemin ne kadar sürdüğünün simülasyonu yapıldığında 1,89'luk bir hızlandırma faktörü ile sürenin sürenin 5.886.277 saat işaretinden 3.115.537 saat işaretine düştüğü gözlemlenmiştir.



Şekil 5.5: Şifreleme Fonksiyonunun Simülasyon Ortamında Aldığı Süre.

5.4 Dekapsülleme

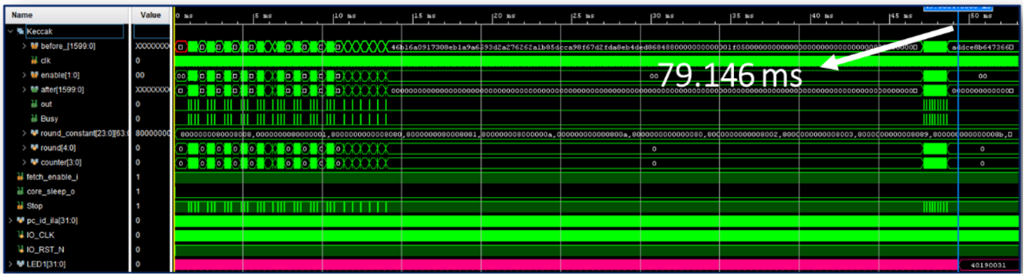


```
int main(void)
{
  unsigned int i;
  volatile unsigned int *LED_ADDR = (volatile unsigned int *) 0x0000c010;

  unsigned char isWrong = 0;

  uint8_t pk[CRYPTO_PUBLICKEYBYTES] = {0};
  uint8_t sk[CRYPTO_SECRETKEYBYTES] = {0};
  uint8_t ct[CRYPTO_CIPHERTEXTBYTES];
  uint8_t key_a[CRYPTO_BYTES];
  uint8_t key_b[CRYPTO_BYTES];


  for(i=0;i<NTESTS;i++) {
    // Decapsulation
    crypto_kem_dec(key_a, ct, sk);

    *LED_ADDR = 40190031;

  }

  return 0;
}
```

Şekil 5.6: Şifre Çözme Fonksiyonunun Ne Kadar Sürdüğünü Gösteren Tetiklenme Kodu.

Veri açma (dekapsülleme, decapsulation) süresinin ne kadar sürdüğünü anlamak için yukarıdaki kod parçası kullanılmıştır. Bu kısımda kodun tamamı doğru veriler ile çalışmadığı için doğru bir sonuç beklenmez, burada amaç sadece bu fonksiyonun ne kadar sürdüğünü anlamaktır.

Dekapsülleme (decapsulation) kısmı kod içerisinde tek başına çalıştırılarak sadece bu işlemin ne kadar sürdüğünün simülasyonu yapıldığında 1,57'lik bir hızlandırma faktörü ile sürenin sürenin 6.222.787 saat işaretinden 3.957.289 saat işaretine düştüğü gözlemlenmiştir.



Şekil 5.7: Şifre Çözme Fonksiyonunun Simülasyon Ortamında Aldığı Süre.

Ibex üzerinde, saf yazılım ve tasarlanan KeccakF1600 donanımı ile birlikte yazılım implementasyonunun fonksiyon özellerinde gerçeklenme süreleri karşılaştırması ve hızlandırma değerleri Çizelge 6.1'de belirtilmiştir.

Çizelge 6.1 : Saf Yazılım ve Donanım/Yazılım Gerçeklemesi Süresi Karşılaştırması.

Fonksiyon	Saf-Yazılım (Saat döngüsü)	Donanım/Yazılım Ortak Tasarım (Saat Döngüsü)	Hızlandırma Oranı (%)
Anahtar Üretme	4.702.047,5	2.465.323,5	47,57
Şifreleme	5.886.277,5	3.115.537,5	47,07
Şifre Çözme	6.222.787.5	3.957.289,5	36,41

Tasarlanan KeccakF1600 donanımının kullandığı alan ve Ibex çekirdeğinin kullandığı alan Çizelge 6.2'de yer verilmiştir. FPGA kartı olarak NEXYS-A7-100T[14] modeli tercih edilmiştir.

Çizelge 6.2 : KeccakF1600 ve Ibex Çekirdeğinin Kullandığı Donanım Alanı Karşılaştırması.

	Kaynak	Kaynak Kullanımı	FPGA Üzerinde Bulan Kaynak Sayısı	Kaynak Kullanımı (%)
KECCAKF1600	LUT	6933	63400	10,94
	FF	3543	126800	2,79
Ibex	LUT	5945	63400	9,41
	LUTRAM	545	19000	2,87
	FF	5657	126800	4,46
	BRAM	67	135	49,63
	DSP	1	240	0,42
	IO	34	210	16,19
	PLL	1	6	16,67

Çizelge 6.3 : CRYSTALS-Kyber Algoritması Özelinde Tasarlanan Donanımların Hızlandırma Değeri Karşılaştırması

Dizayn	Hızlandırıcı Donanım	Mimari	Hızlandırma Faktörü
Fritzman, Sigl ve Sep´ulveda [4] - 2020	KeccakF1600 ve NTT	RISC-V (PULPino)	9,6
Nannipieri vd. [19] - 2021	NTT	RISC-V (PULPino)	1,84
Masera, Martina ve Dolmeta [18] - 2023	KeccakF1600	RISC-V (PULPissimo)	2,47
Bizim Tasarımımız	KeccakF1600	RISC-V (Ibex)	1,76

Fritzman, Sigl ve Sep´ulveda çalışmasında CRYSTALS-Kyber algoritmasını RISC-V tabanlı PULPino mikrokontrolcüsünde gerçeklemiş olup yazılımı uygun bir biçimde optimize etmişlerdir. Daha sonrasında aritmetik lojik birimine (ALU) KeccakF1600 ve kod profilleme sonucu 3. En fazla zaman kaplayan “Montgomery Reduce” fonksiyonunu gerçekleştirebilecek üç farklı donanım eklemişlerdir. Bu donanımlar komut dizini eklentisi (ISE) ile eklenmiştir. Ayrıca veri transferi arabirimine (BUS) bağlı bir NTT hızlandırcısı içermektedir bu şekilde 9,6 oranında bir hızlandırma faktörü elde edilmiştir [4]. Nannipieri vd. Çalışmasında NTT işlemini gerçekleyecek bir donanım tasarlayarak veri transferi arabirimi (BUS) eklenmiştir. Yazılım gerçeklemesini de optimize ederek 1,84 hızlandırma faktörü elde etmişlerdir [19].

Mesera, Martina ve Dolmetanın çalışmasında RISC-V tabanlı PULPissimo mikrokontrolcüsünün veri transferi arabirimine (BUS) bağlı bir KECCAKF1600 donanımı ekleyerek ve yazılım gerçeklemesinde optimizasyon yaparak 2,47 hızlandırma faktörü elde etmiştir [18]. Bizim tasarımımızda ise RISC-V çekirdeğine bağlı bir veri transferi arabirimi (BUS) içermemekle birlikte KECCAKF1600 donanımı direct olarak hafıza ve Ibex'e bağlıdır. Yazılım gerçeklemesinde de herhangi bir optimizasyona gidilmemiştir.

6. GERÇEKÇI KISITLAR, SONUÇLAR VE ÖNERİLER

6.1 Çalışmanın Uygulama Alanı

Bu çalışmanın uygulama alanı, CRYSTALS-Kyber PQC algoritmasını kullanarak haberleşen bir sistemin veri alış verişinde şifreleme yapılması sırasında kullanılması olarak belirlenmiştir. FPGA üzerinde çalışan bir RISC-V işlemciye eklenmiş KeccakF1600_StatePermute işlevini gerçekleştiren bir donanımla birlikte anahtar oluşturma, şifreleme ve şifre açma işlemleri daha hızlı yapılarak veri alış verişinin yavaşlamasının bir miktar önüne geçilmeye çalışılmıştır.

6.2 Gerçekçi Tasarım Kısıtları

Tasarımda hem yazılımda hem de donanımda açık kaynak kodlar kullanıldığı için kullanılan kodların modifikasyonu için bir sınır yoktur. Fakat donanımdan gelebilecek olan kritik yol (critical path) sınırı donanımın performansını ve hızını kısıtlamaktadır, tasarlanan donanım işlemcinin kritik yolundan daha uzun bir yola sahip olmamalıdır. Aksi takdirde işlemcinin çalışma frekansı düşürülmek zorunda kalınır ve bu da istenmeyen bir şeydir. Diğer yandan kullanılan işlemcinin ve tasarlanan donanımın alan kısıtı vardır. FPGA üzerinde kullanılan alan efektif bir şekilde kullanılmalıdır, tasarlanan sistemin ihtiyacı olan alan, kullanımdaki FPGA'i geçmemiştir

6.2.1 Maliyet

Tasarlanan sistem Nexys A7-100T'de çalışması hedeflenerek tasarlanmıştır. Bunun dışında Xilinx Vivado yazılımı ile simülasyonlar yapılmıştır, fakat Xilinx'in öğrenci lisansı kullanılmıştır.

6.2.2 Standartlar

Donanım tasarımları için IEEE'nin System Verilog ve VHDL standartları kullanılmıştır. Aynı şekilde C programlama dili için C99 standartı referans alınmıştır.

6.2.3 Sosyal, çevresel ve ekonomik etki

Çalışılan konu güvenlik sistemleri ile alakalı olduğu için PQC algoritmalarının standartlaşmaları ve yaygın kullanımı sonrası milyar dolarlık sektörlerde kullanılabilir. Bu proje, CRYSTALS-Kyber algoritmasının hızlandırılmasıyla birlikte başka araştırmalara da konu olarak daha büyük bir akademik-sosyal etki yaratabilir. Çevresel faktör olarak da kullanılan güç minimumda tutulmaya çalışılarak çevreye kötü bir etki bırakmamaya çalışılmıştır.

6.2.4 Sağlık ve güvenlik riskleri

Kullanılan ortamlar yazılımsal ortamlar olduğu için projede herhangi bir sağlık ve güvenlik riski belirlenmemiştir.

6.3 Sonuçlar

Sonuç olarak bu projede öncelikle CRYSTALS-Kyber algoritmasının kod profilleme işlemi ile darboğaza yol açan fonksiyonu KeccakF1600_StatePermute olarak belirlenmiştir. Daha sonra bu fonksiyonun donanım gerçeklemesi yapıldıktan sonra açık kaynak RISC-V mimarili işlemci olan Ibex Core'a entegrasyonu gerçekleştirilmiştir. Yapılan simülasyonlar sonucunda anahtar üretme, şifreleme ve şifre açma adımları için sırasıyla 1,91, 1,89 ve 1,57'lik hızlandırma çarpanları ile algoritma hızlandırılmıştır. Bu hızlandırma karşısında eldeki mevcut alandan fazladan yalnızca 6933 LUT kullanılmıştır. Sonuçta genel olarak algoritmada 1,76'lık bir hızlandırma çarpanı elde edilmiştir. Simülsayonlar aracalığı ile belirlenen bu çarpan, kullanılan alana göre iyi seviyededir.

6.4 Geleceğe Yönelik Öneriler

Eklenen bu donanım Instruction Set Extension (ISE) ile değil donanım adresleme (hardware addressing) metodu ile eklenmiştir. Bu metodun kullanılması yerine ISE ile mimarinin geliştirilmesi, işlemciye dışarıdan müdehaleyi daha az hale getireceği için daha tercih edilebilir bir metot olabilir. Bu metotun kullanılmasının devam edilmesi halinde sırasız [Out of Order (OoO)] mimaride çalışan işlemciler için bir modifikasyon gerekecektir, örneğin donanımın aktive edilmesi görevi OoO mimarilerde Yeniden Düzenleme Birimine [Re-order Buffer (ROB)] verilebilir. Ek olarak, CRYSTALS-Kyber algoritmasının en çok kullanılan diğer donanımları da mimariye eklenerek daha hızlı fonksiyonalitesi olan bir algoritma elde edilebilir. Bu fonksiyonlar, çalışma içerisinde kod profilleme kısmında belirtilmiştir. Buradaki boyut düşürme (reduction) algoritmalarından birinin ISE ile mimariye eklenmesi iyi bir tercih olabilir.

KAYNAKLAR

[1] **CRYSTALS Organization,** https://pq-crystals.org/kyber/resources.shtml

[2] **Url-1** < https://github.com/pq-crystals/kyber >, erişim tarihi 31.05.2023

[3] **M. Bisheh-Niasar, R. Azarderakhsh and M. Mozaffari-Kermani**, "High-Speed NTT-based Polynomial Multiplication Accelerator for Post-Quantum Cryptography," 2021 IEEE 28th Symposium on Computer Arithmetic (ARITH), Lyngby, Denmark, 2021, pp. 94-101, doi: 10.1109/ARITH51176.2021.00028.

[4] **Fritzmann, T., Sigl, G., & Sepúlveda, J**. (2020). RISQ-V: Tightly Coupled RISCV Accelerators for Post-Quantum Cryptography. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2020(4), 239–280. https://doi.org/10.13154/tches.v2020.i4.239-280

[5] **M. Bisheh-Niasar, R. Azarderakhsh and M. Mozaffari-Kermani**, "Instruction-Set Accelerated Implementation of CRYSTALS-Kyber," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 68, no. 11, pp 4648-4659, Nov. 2021, doi: 10.1109/TCSI.2021.3106639.

[6] **Url-2** < https://keccak.team/keccak.html >, erişim tarihi 31.05.2023

[7] **Url-3:** https://ibex-core.readthedocs.io/en/latest/, erişim tarihi 31.05.2023

[8] **Shor, P.W.** (1999). Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer, SIAM Review, 41(2), 303–332, https://doi.org/10.1137/S0036144598347011,

[9] **Xin Zhou and Xiaofei Tang,** "Research and implementation of RSA algorithm for encryption and decryption," Proceedings of 2011 6th International Forum on Strategic Technology, Harbin, Heilongjiang, 2011, pp. 1118-1121, doi: 10.1109/IFOST.2011.6021216.

[10] **Alagic, G. , Apon, D. , Cooper, D. , Dang, Q. , Dang, T. , Kelsey, J. , Lichtinger, J. , Liu, Y. , Miller, C. , Moody, D. , Peralta, R. , Perlner, R. , Robinson, A. and Smith-Tone, D.** (2022), Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process, NIST Interagency/Internal Report (NISTIR), National Institute of Standards and Technology, Gaithersburg, MD, [online], https://doi.org/10.6028/NIST.IR.8413-upd.

[11] ***Post-quantum cryptography - lattice-based cryptography.*** *(2023). Retrieved January 5, 2024, Mevcut:* https://www.redhat.com/en/blog/post-quantum-cryptography-lattice-based-cryptography

[12] **R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, ve D. Stehlé,** "CRYSTALS-Kyber Algorithm Specifications and Supporting Documentation, Versiyon 3.0," [Online]. Mevcut: http://www.example.com. [Erişildi: 7 Ocak 2024].

[13] **Liang, Z., & Zhao, Y. (2022).** Number Theoretic Transform and Its Applicationsin Latticebased Cryptosystems: A Survey. doi:https://doi.org/10.48550/arXiv.2211.13546.

[14] **Digilent,** "Nexys A7™ FPGA Board Referans Kılavuzu," t.y. [10 Temmuz 2019 tarihinde güncellenmiştir].

[15] **Xilinx,** "Vivado Design Suite Kullanıcı Kılavuzu," 2012 [Kasım 2023 tarihinde güncellenmiştir].

[16] **O. Altınay ve S.B. Ö. Yalçın**, "”Instruction Extension of RV32I and GCC BackEnd for Ascon Lightweight Cryptography Algorithm," *2021 IEEE International Conference on Omni-Layer Intelligent Systems (COINS)*, Barselona, İspanya, 2021, ss. 1-6, doi: 10.1109/COINS51742.2021.9524190.

[17] **D. J. Bernstein, T. Lange ve P. Schwabe**, "The security impact of a new cryptographic library," *Proceedings of Progress in Cryptology – LATINCRYPT 2012*, A. Hevia ve G. Neven, Ed., c. 7533, LNCS, ss. 159–176, Springer, 2012. [Çevrimiçi]. Erişim adresi: http://cryptojedi.org/papers/coolnacl.

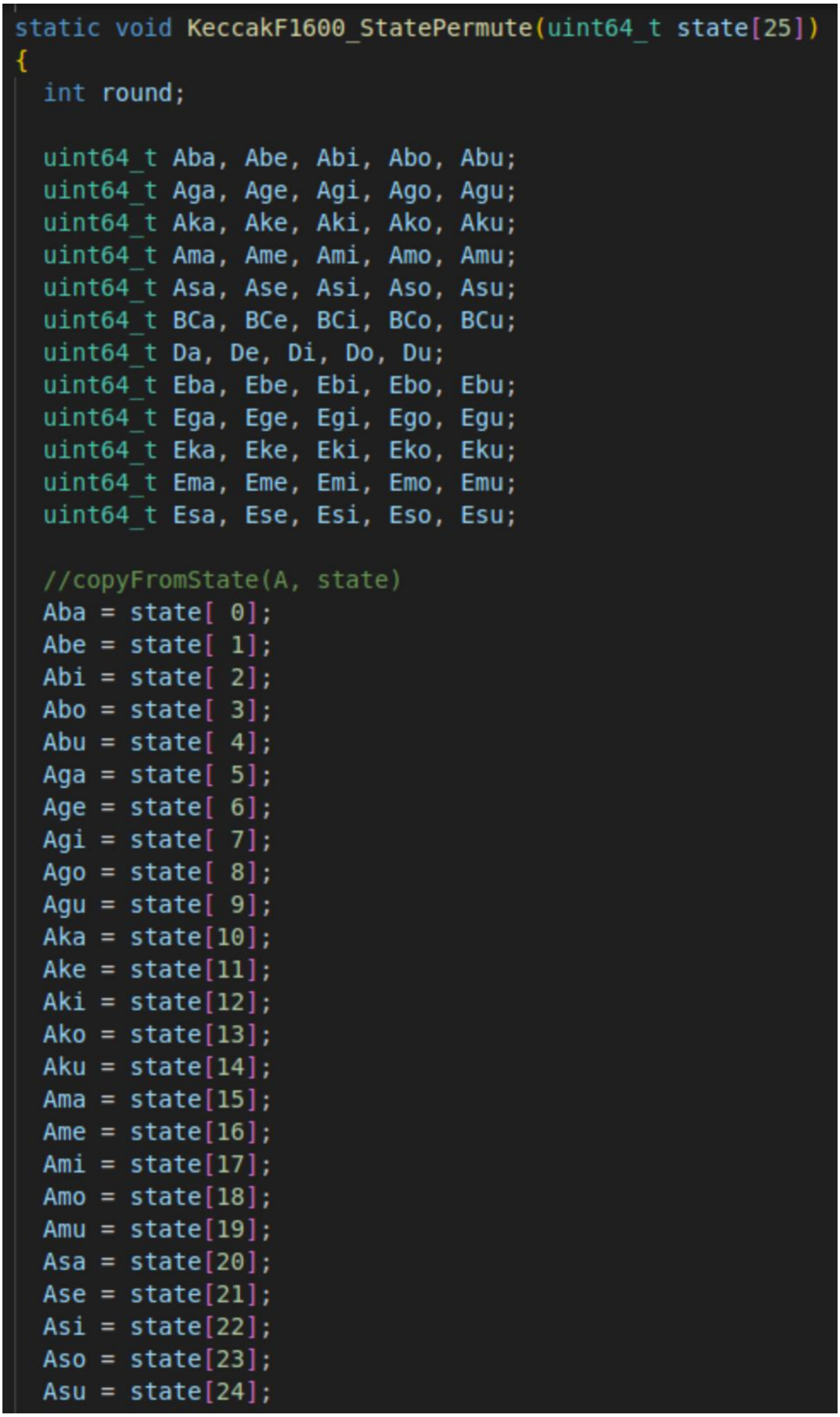
[18] **G. Masera, M. Martina,**” Integration and optimization of a RISC-V based Keccak accelerator,” Yüksek Lisans Tezi, Corso di laurea magistrale in Ingegne-ria Elettronica (Elektronik Mühendisliği), Politecnico di Torino, Torino, 2023. [Çevrimiçi]. Erişim adresi: https://webthesis.biblio.polito.it/26725/

[19] **P. Nannipieri, S. Di Matteo, L. Zulberti, F. Albicocchi, S. Saponara ve L. Fanucci,** ”A RISC-V Post Quantum Cryptography Instruction Set Extension for Number Theoretic Transform to Speed-Up CRYSTALS Algorithms,” *IEEE Access*, vol. 9, pp. 150798-150808, 2021, doi: 10.1109/ACCESS.2021.3126208

EKLER

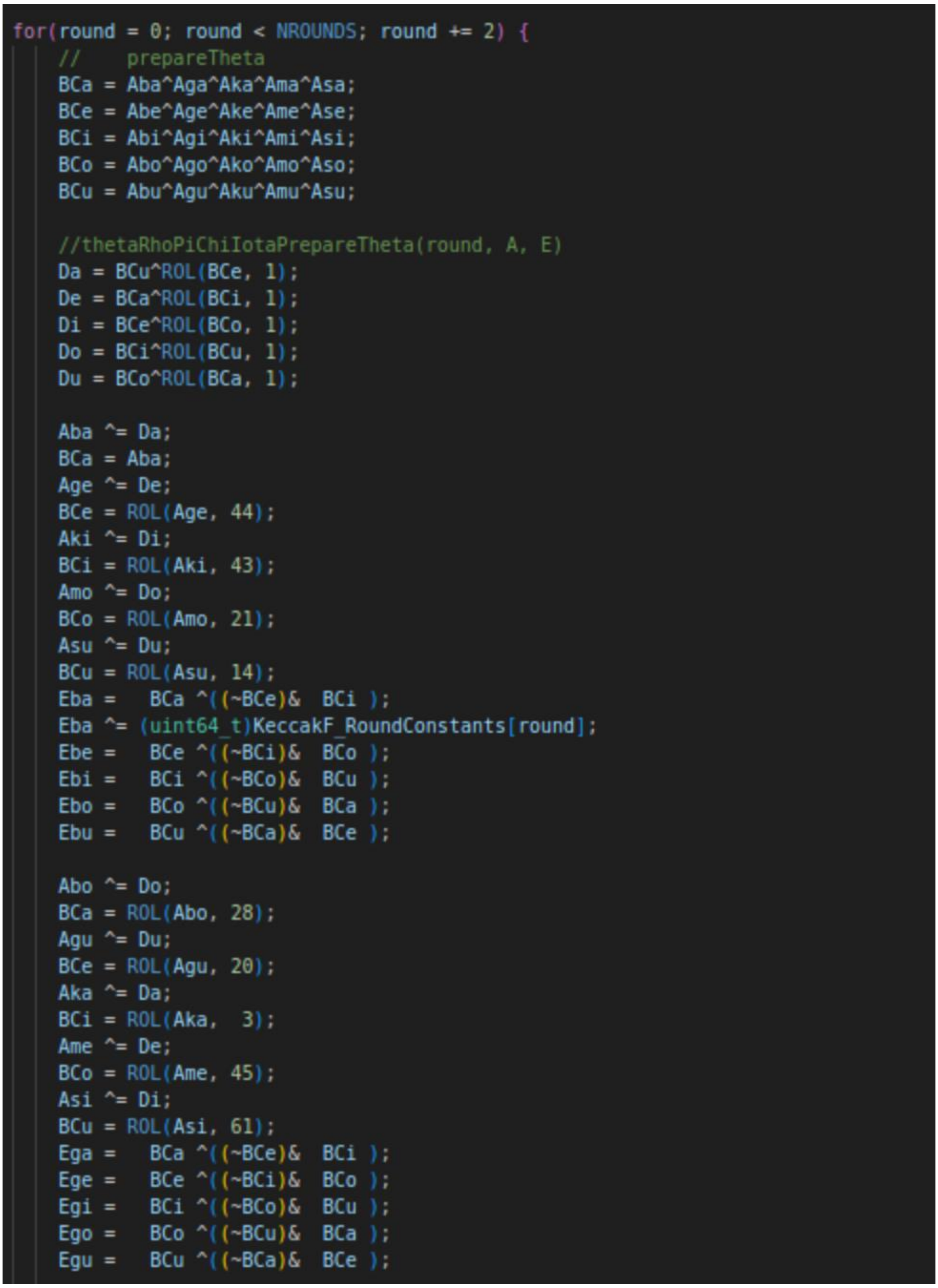
EK A: Keccak F1600 Kodu

EK A



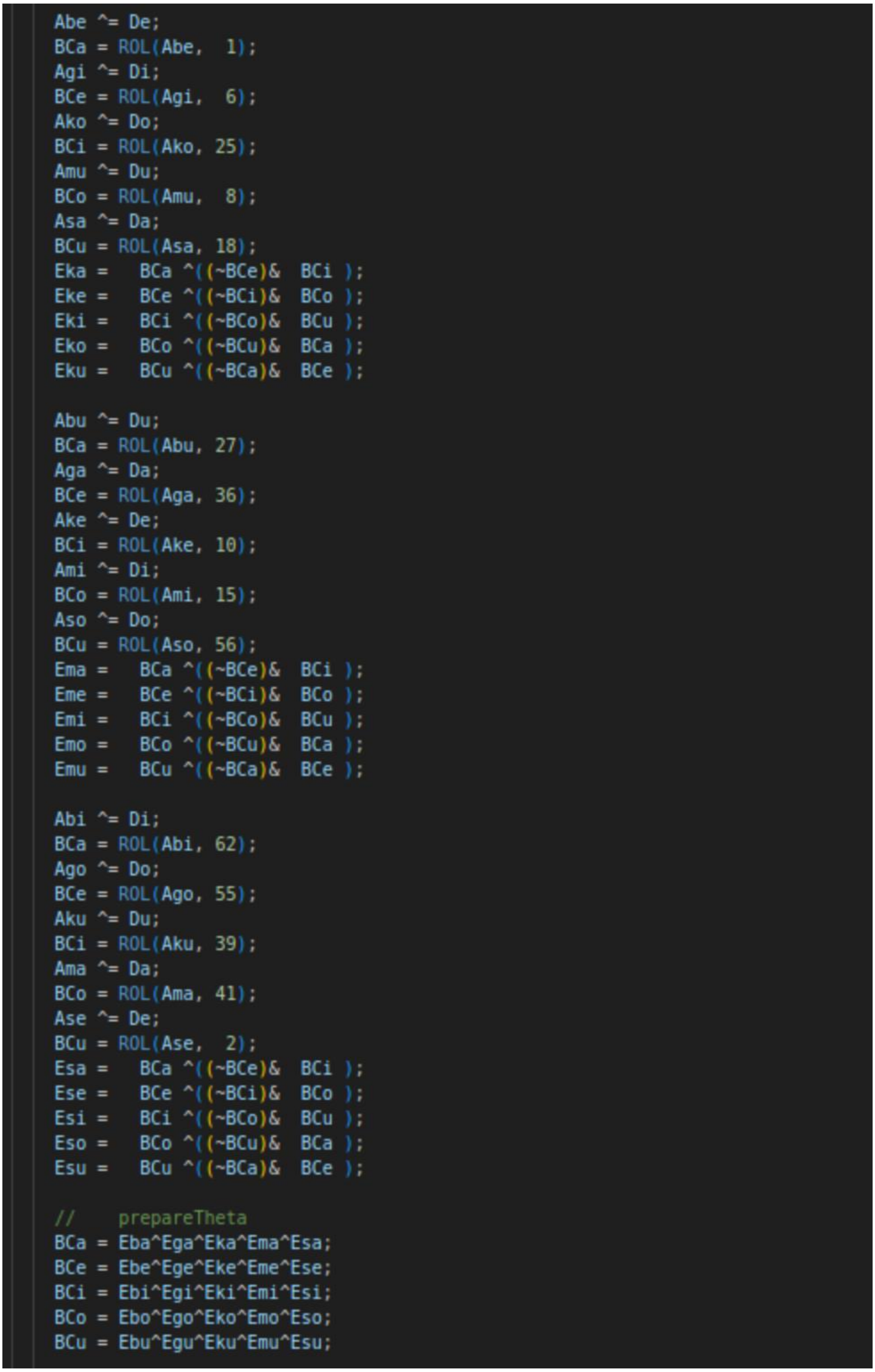
```
static void KeccakF1600_StatePermute(uint64_t state[25])
{
  int round;

  uint64_t Aba, Abe, Abi, Abo, Abu;
  uint64_t Aga, Age, Agi, Ago, Agu;
  uint64_t Aka, Ake, Aki, Ako, Aku;
  uint64_t Ama, Ame, Ami, Amo, Amu;
  uint64_t Asa, Ase, Asi, Aso, Asu;
  uint64_t BCa, BCe, BCi, BCo, BCu;
  uint64_t Da, De, Di, Do, Du;
  uint64_t Eba, Ebe, Ebi, Ebo, Ebu;
  uint64_t Ega, Ege, Egi, Ego, Egu;
  uint64_t Eka, Eke, Eki, Eko, Eku;
  uint64_t Ema, Eme, Emi, Emo, Emu;
  uint64_t Esa, Ese, Esi, Eso, Esu;

  //copyFromState(A, state)
  Aba = state[ 0];
  Abe = state[ 1];
  Abi = state[ 2];
  Abo = state[ 3];
  Abu = state[ 4];
  Aga = state[ 5];
  Age = state[ 6];
  Agi = state[ 7];
  Ago = state[ 8];
  Agu = state[ 9];
  Aka = state[10];
  Ake = state[11];
  Aki = state[12];
  Ako = state[13];
  Aku = state[14];
  Ama = state[15];
  Ame = state[16];
  Ami = state[17];
  Amo = state[18];
  Amu = state[19];
  Asa = state[20];
  Ase = state[21];
  Asi = state[22];
  Aso = state[23];
  Asu = state[24];
```

Şekil A.1 : Keccak Kod



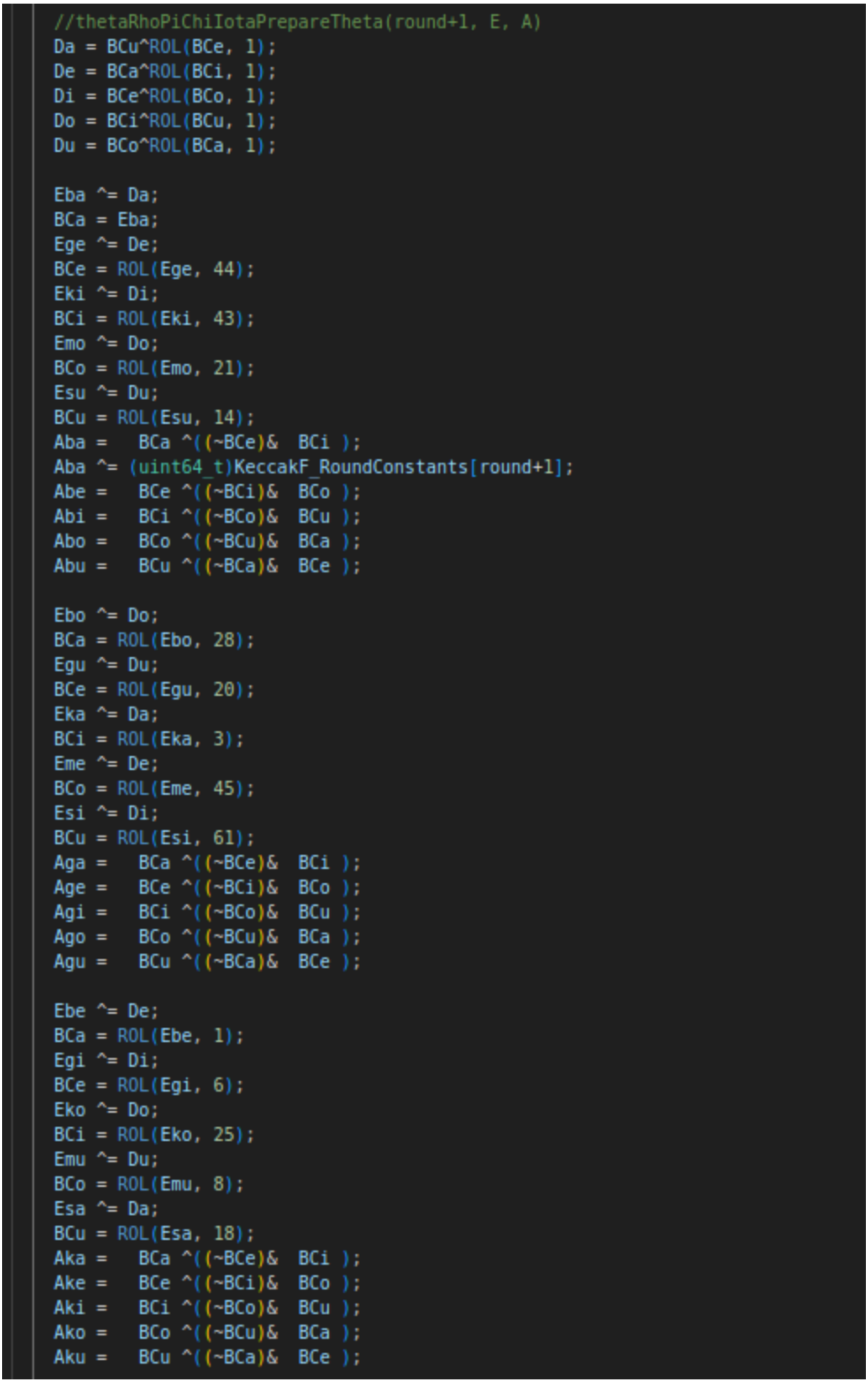
```
for(round = 0; round < NROUNDS; round += 2) {
    //    prepareTheta
    BCa = Aba^Aga^Aka^Ama^Asa;
    BCe = Abe^Age^Ake^Ame^Ase;
    BCi = Abi^Agi^Aki^Ami^Asi;
    BCo = Abo^Ago^Ako^Amo^Aso;
    BCu = Abu^Agu^Aku^Amu^Asu;

    //thetaRhoPiChiIotaPrepareTheta(round, A, E)
    Da = BCu^ROL(BCe, 1);
    De = BCa^ROL(BCi, 1);
    Di = BCe^ROL(BCo, 1);
    Do = BCi^ROL(BCu, 1);
    Du = BCo^ROL(BCa, 1);

    Aba ^= Da;
    BCa = Aba;
    Age ^= De;
    BCe = ROL(Age, 44);
    Aki ^= Di;
    BCi = ROL(Aki, 43);
    Amo ^= Do;
    BCo = ROL(Amo, 21);
    Asu ^= Du;
    BCu = ROL(Asu, 14);
    Eba =   BCa ^((~BCe)&  BCi );
    Eba ^= (uint64_t)KeccakF_RoundConstants[round];
    Ebe =   BCe ^((~BCi)&  BCo );
    Ebi =   BCi ^((~BCo)&  BCu );
    Ebo =   BCo ^((~BCu)&  BCa );
    Ebu =   BCu ^((~BCa)&  BCe );

    Abo ^= Do;
    BCa = ROL(Abo, 28);
    Agu ^= Du;
    BCe = ROL(Agu, 20);
    Aka ^= Da;
    BCi = ROL(Aka,  3);
    Ame ^= De;
    BCo = ROL(Ame, 45);
    Asi ^= Di;
    BCu = ROL(Asi, 61);
    Ega =   BCa ^((~BCe)&  BCi );
    Ege =   BCe ^((~BCi)&  BCo );
    Egi =   BCi ^((~BCo)&  BCu );
    Ego =   BCo ^((~BCu)&  BCa );
    Egu =   BCu ^((~BCa)&  BCe );
```

Şekil A.2 : Keccak Kod



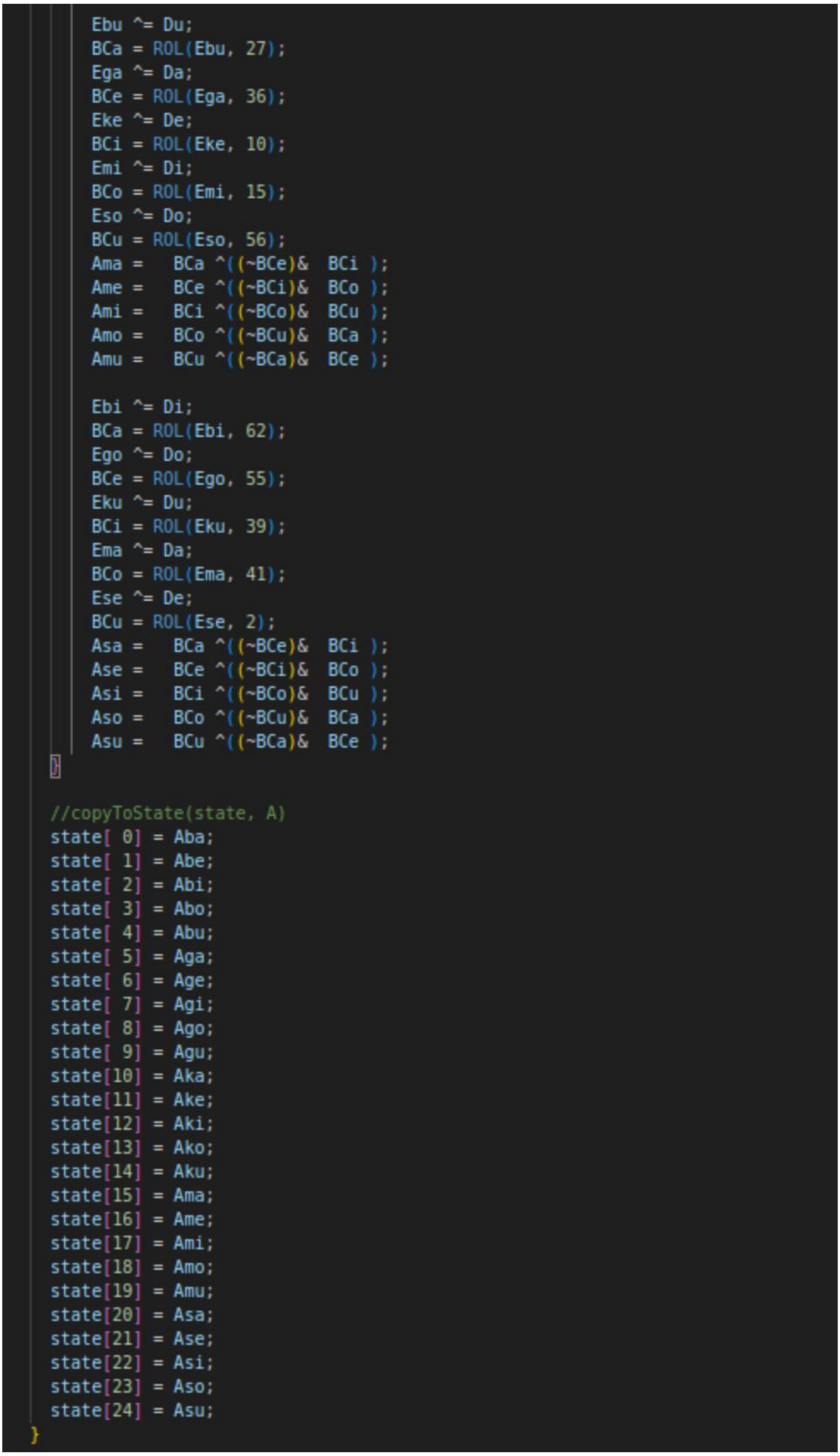
```
Abe ^= De;
BCa = ROL(Abe,  1);
Agi ^= Di;
BCe = ROL(Agi,  6);
Ako ^= Do;
BCi = ROL(Ako, 25);
Amu ^= Du;
BCo = ROL(Amu,  8);
Asa ^= Da;
BCu = ROL(Asa, 18);
Eka =   BCa ^((~BCe)&  BCi );
Eke =   BCe ^((~BCi)&  BCo );
Eki =   BCi ^((~BCo)&  BCu );
Eko =   BCo ^((~BCu)&  BCa );
Eku =   BCu ^((~BCa)&  BCe );

Abu ^= Du;
BCa = ROL(Abu, 27);
Aga ^= Da;
BCe = ROL(Aga, 36);
Ake ^= De;
BCi = ROL(Ake, 10);
Ami ^= Di;
BCo = ROL(Ami, 15);
Aso ^= Do;
BCu = ROL(Aso, 56);
Ema =   BCa ^((~BCe)&  BCi );
Eme =   BCe ^((~BCi)&  BCo );
Emi =   BCi ^((~BCo)&  BCu );
Emo =   BCo ^((~BCu)&  BCa );
Emu =   BCu ^((~BCa)&  BCe );

Abi ^= Di;
BCa = ROL(Abi, 62);
Ago ^= Do;
BCe = ROL(Ago, 55);
Aku ^= Du;
BCi = ROL(Aku, 39);
Ama ^= Da;
BCo = ROL(Ama, 41);
Ase ^= De;
BCu = ROL(Ase,  2);
Esa =   BCa ^((~BCe)&  BCi );
Ese =   BCe ^((~BCi)&  BCo );
Esi =   BCi ^((~BCo)&  BCu );
Eso =   BCo ^((~BCu)&  BCa );
Esu =   BCu ^((~BCa)&  BCe );

//    prepareTheta
BCa = Eba^Ega^Eka^Ema^Esa;
BCe = Ebe^Ege^Eke^Eme^Ese;
BCi = Ebi^Egi^Eki^Emi^Esi;
BCo = Ebo^Ego^Eko^Emo^Eso;
BCu = Ebu^Egu^Eku^Emu^Esu;
```

Şekil A.3 : Keccak Kod



```
//thetaRhoPiChiIotaPrepareTheta(round+1, E, A)
Da = BCu^ROL(BCe, 1);
De = BCa^ROL(BCi, 1);
Di = BCe^ROL(BCo, 1);
Do = BCi^ROL(BCu, 1);
Du = BCo^ROL(BCa, 1);

Eba ^= Da;
BCa = Eba;
Ege ^= De;
BCe = ROL(Ege, 44);
Eki ^= Di;
BCi = ROL(Eki, 43);
Emo ^= Do;
BCo = ROL(Emo, 21);
Esu ^= Du;
BCu = ROL(Esu, 14);
Aba =   BCa ^((~BCe)&  BCi );
Aba ^= (uint64_t)KeccakF_RoundConstants[round+1];
Abe =   BCe ^((~BCi)&  BCo );
Abi =   BCi ^((~BCo)&  BCu );
Abo =   BCo ^((~BCu)&  BCa );
Abu =   BCu ^((~BCa)&  BCe );

Ebo ^= Do;
BCa = ROL(Ebo, 28);
Egu ^= Du;
BCe = ROL(Egu, 20);
Eka ^= Da;
BCi = ROL(Eka, 3);
Eme ^= De;
BCo = ROL(Eme, 45);
Esi ^= Di;
BCu = ROL(Esi, 61);
Aga =   BCa ^((~BCe)&  BCi );
Age =   BCe ^((~BCi)&  BCo );
Agi =   BCi ^((~BCo)&  BCu );
Ago =   BCo ^((~BCu)&  BCa );
Agu =   BCu ^((~BCa)&  BCe );

Ebe ^= De;
BCa = ROL(Ebe, 1);
Egi ^= Di;
BCe = ROL(Egi, 6);
Eko ^= Do;
BCi = ROL(Eko, 25);
Emu ^= Du;
BCo = ROL(Emu, 8);
Esa ^= Da;
BCu = ROL(Esa, 18);
Aka =   BCa ^((~BCe)&  BCi );
Ake =   BCe ^((~BCi)&  BCo );
Aki =   BCi ^((~BCo)&  BCu );
Ako =   BCo ^((~BCu)&  BCa );
Aku =   BCu ^((~BCa)&  BCe );
```

Şekil A.4 : Keccak Kod



```
            Ebu ^= Du;
            BCa = ROL(Ebu, 27);
            Ega ^= Da;
            BCe = ROL(Ega, 36);
            Eke ^= De;
            BCi = ROL(Eke, 10);
            Emi ^= Di;
            BCo = ROL(Emi, 15);
            Eso ^= Do;
            BCu = ROL(Eso, 56);
            Ama =   BCa ^((~BCe)&  BCi );
            Ame =   BCe ^((~BCi)&  BCo );
            Ami =   BCi ^((~BCo)&  BCu );
            Amo =   BCo ^((~BCu)&  BCa );
            Amu =   BCu ^((~BCa)&  BCe );

            Ebi ^= Di;
            BCa = ROL(Ebi, 62);
            Ego ^= Do;
            BCe = ROL(Ego, 55);
            Eku ^= Du;
            BCi = ROL(Eku, 39);
            Ema ^= Da;
            BCo = ROL(Ema, 41);
            Ese ^= De;
            BCu = ROL(Ese, 2);
            Asa =   BCa ^((~BCe)&  BCi );
            Ase =   BCe ^((~BCi)&  BCo );
            Asi =   BCi ^((~BCo)&  BCu );
            Aso =   BCo ^((~BCu)&  BCa );
            Asu =   BCu ^((~BCa)&  BCe );
    }

    //copyToState(state, A)
    state[ 0] = Aba;
    state[ 1] = Abe;
    state[ 2] = Abi;
    state[ 3] = Abo;
    state[ 4] = Abu;
    state[ 5] = Aga;
    state[ 6] = Age;
    state[ 7] = Agi;
    state[ 8] = Ago;
    state[ 9] = Agu;
    state[10] = Aka;
    state[11] = Ake;
    state[12] = Aki;
    state[13] = Ako;
    state[14] = Aku;
    state[15] = Ama;
    state[16] = Ame;
    state[17] = Ami;
    state[18] = Amo;
    state[19] = Amu;
    state[20] = Asa;
    state[21] = Ase;
    state[22] = Asi;
    state[23] = Aso;
    state[24] = Asu;
}
```

Şekil A.5 : Keccak Kod

ÖZGEÇMİŞ

Ad-Soyad **: Ahmet Çelik**

Doğum Tarihi ve Yeri **: 23.06.2000 Balıkesir, Türkiye**

E-posta : celikah19@itu.edu.tr

Lisans : İstanbul Teknik Üniversitesi – Elektronik ve Haberleşme Mühendisliği (2019- 2024)
İkinci Anadal: İstanbul Teknik Üniversitesi – Bilgisayar Mühendisliği (2021-2024)

DENEYİM:

- 2023 - ASPOWER – Yazılım Mühendisi
- 2022 - BOSCH – Yapay Zeka ve Veri Bilimi Stajı
- 2021 - Baykar Technologies – İnsan Makine Etkieşimli Yazılım Teknolojileri Stajı

YAYIN:
A.Celik, F. Yilmaz, M. A. Korkmaz, and B. Ors, ”Implementation of CRYSTALS-Kyber Post-Quantum Algorithm Using RISC-V Processor”, *in 30th IEEE International Conference on Electronics Circuits and Systems (ICECS),* Istanbul, Turkey, 2023.

ÖZGEÇMİŞ

Ad-Soyad	**: Fatih Yılmaz**
Doğum Tarihi ve Yeri	**: 07.03.2001 Samsun, Türkiye**
E-posta	**:** fatihellibes@gmail.com

Lisans : İstanbul Teknik Üniversitesi – Elektronik ve Haberleşme Mühendisliği (2019- 2024)

DENEYİM:

- 2023 - ASPOWER – Yazılım Mühendisi
- 2023 - HAVELSAN - Gömülü Sistem ve Donanım Tasarımı Stajı
- 2023 - Turksat Uydu Haberleşme - Uydu Frekans Gözlem Stajı
- 2022 - Roketsan - Gömülü Sistem Yazılım ve Donanım Tasarım Mühendisi Stajı
- 2022 - Aselsan - Sistem Mühendisliği Stajı
- 2021 - Baykar Technologies - Ar-Ge Proje Yönetimi ve Sistem Mühendisliği Stajı

YAYIN:

A.Celik, **F. Yilmaz**, M. A. Korkmaz, and B. Ors, ”Implementation of CRYSTALS-Kyber Post-Quantum Algorithm Using RISC-V Processor”, *in 30th IEEE International Conference on Electronics Circuits and Systems (ICECS)*, Istanbul, Turkey, 2023.

ÖZGEÇMİŞ

Ad-Soyad : **Mehmet Anıl Korkmaz**

Doğum Tarihi ve Yeri : **01/05/2001 Aksaray, Türkiye**

E-posta : anil68krk@gmail.com

Lisans : İstanbul Teknik Üniversitesi – Elektronik ve Haberleşme Mühendisliği (2019- 2024)

DENEYİM:

- 2023 - Borda Teknoloji Gömülü Sistem Yazılım Mühendsi
- 2022-2023 - İTÜ Güneş Arabası Ekibi Teknik Takım Lideri
- 2022-2023 - İTÜ Güneş Arabası Ekibi Gömülü Sistemler Grup Sorumlusu
- 2019-2022 - İTÜ Güneş Arabası Ekibi Gömülü SIstemler Grup Üyesi

YAYIN:

A.Celik, F. Yilmaz, **M. A. Korkmaz**, and B. Ors, ”Implementation of CRYSTALS-Kyber Post-Quantum Algorithm Using RISC-V Processor”, *in 30th IEEE International Conference on Electronics Circuits and Systems (ICECS),* Istanbul, Turkey, 2023.