IPsec Protokolü için FPGA üzerinde Anahtar-Değer Saklama Donanımı
A Custom Key-Value Store Hardware on FPGA for IPsec Protocol

Murat Benli[1,2], Erdem Özcan[2], Ufuk Türeli[1]

[1]Elektronik ve Haberleşme Mühendisliği Bölümü
Yıldız Teknik Üniversitesi
murat.benli@tubitak.gov.tr , utureli@yildiz.edu.tr

[2]Bilişim ve Bilgi Güvenliği İleri Teknolojiler Araştırma Merkezi
BİLGEM-TÜBİTAK
erdem.ozcan@tubitak.gov.tr

Özet

Bu makale IPsec (Internet Protocol Security) güvenlik protokolünde kullanılan ve anahtar dağıtım altyapısında ihtiyaç duyulan eşsiz anahtar seçimi için tasarlanan anahtar - değer mekanizmasını ele almaktadır. Anahtar – değer mekanizmaları günümüzde Intel, Cisco gibi birçok firma tarafından büyük veri saklama merkezleri için kullanılan bir mekanizmadır. Eşsiz anahtar seçimi için kullanılabilecek bir diğer yapı ise IP(Internet Protocol) ağlarında rotalama için kullanılan CAM (Content Addressable Memory) yapısıdır. Bu çalışmada tasarlanan özet algoritması tabanlı, anahtar – değer mekanizması anlatılırken alternatifleriyle de kıyaslanmaktadır. Mimarideki temel amaç hafıza içerisinde hızlı bir arama mekanizması oluşturmak, bunu yaparken de güç ve kaynak tüketimini arttırmamaktır. Anahtar – değer mimarisinde özet algoritması kullanılmış bunun karşılığında ise çakışma problemine çözüm bulmak gerekmiştir. Çalışmada XILINX Virtex 7 ailesi FPGA (Field Programmable Gate Array) kullanılmıştır.

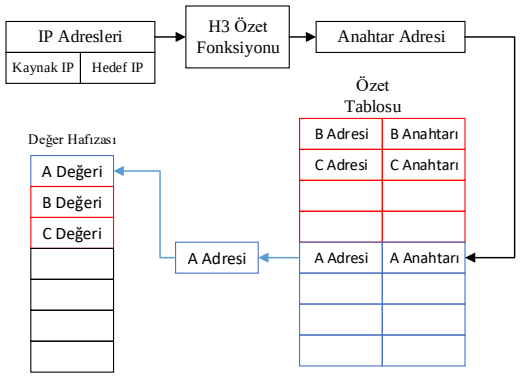
Abstract

This paper addresses the key-value mechanism used in the IPsec(Internet Protocol Security) security protocol and designed for the unique key selection required in the key distribution infrastructure. Key-value mechanisms are a mechanism used in big data storage centers by many companies such as Intel and Cisco today. Another structure that is an alternative to this mechanism is the CAM (Content Addressable Memory) structure used for routing in IP(Internet Protocol) networks. While describing the key-value mechanism based on the hash algorithm designed in this study, it is compared with its alternatives. The main purpose of the mechanism is to create a fast search mechanism in the memory and not to increase the power and resource consumption while doing this. For this, a hash algorithm has been used, in return, it is necessary to face the conflict problem. XILINX Virtex 7 family FPGA (Field Programmable Gate Array) was used in the study.

1. Giriş

Eşsiz anahtar seçimi için ele alınan yapılar: CAM ve anahtar – değer yapılarıdır. Daha çok IP ağlarında rotalama için kullanılan CAM yapıları tek saat darbesinde tam eşleşme sağlasa da kaynak tüketimi açısından anahtar – değer yapılarıyla kıyaslandığında daha verimsizdirler. Bu kıyaslama sonraki bölümlerde ele alınacaktır.

Anahtar değer yapıları iki temel bileşenden oluşan yapılardır. Anahtar değer yapıları temel olarak Şekil 1'deki gibidir. Burada anahtar, değeri içeren hafızaya adreslerken, değer ise anahtara karşılık gelen bilgiyi tutmaktadır. Kullanımına örnek vermek gerekirse; anahtar kullanıcı adı ya da numarası olurken, değer ise o kullanıcının şifresi, adresi gibi aranan veriler olabilir [1]. Bu kullanımın dışında anahtar değer yapılarını başka uygulamalarda da kullanmak mümkündür. Bunlardan bazıları paket sınıflandırma, trafik sınıflandırma ve ağır vuruş algılamadır [2].



Şekil 1: Anahtar Değer Mimarisi

Büyük veriler için anahtar değer mekanizmalarının çalışma prensibini en temele indirgeyecek olursak, iki komut vardır: *Kaydet* ve *Getir* [3]. Bu komutlara benzer şekilde, çalışmada Oku ve Yaz komutları kullanılmıştır. Anahtar değer yapılarında temel prensip arama eylemine düşen işi azaltmak olduğundan, yapının kalbi olan özet fonksiyonları ve özet tabloları kullanılır. Özet

fonksiyonu seçiminde ihtiyaç duyulan parametrelerden en önemlileri mimarideki anahtar boyuna uyumluluğu ve hedeflenen çakışma oranını sağlanmasıdır. Özet fonksiyonu seçiminde temel motivasyon yukarıdaki etmenlerin dışında ölçeklenebilmesi ve uygulamadaki kolaylığıdır. Çalışmada gerçekleme ve ölçekleme işini en kolay yönetebilmek ve yapının en hızlı çalışmasını sağlamak için FPGA kullanılmıştır. Çalışılacak veri seti anahtar değer yapısının genel kullanım senaryosunun aksine küçük olduğundan harici DRAM (Dynamic Random Access Memory) kullanımı yerine BRAM (Block Random Access Memory) kullanımını tercih edilmiştir. Çalışma sırasında karşılaşılan 3 temel zorluk sıralanacak olursa:

1. Özet algoritmasından dolayı oluşacak olan çakışma problemini giderme gereği,
2. Ardışık düzenli bir yapı kullanıldığında oluşacak gecikmeleri tolere edebilecek bir ara bellek oluşturma gerekliliği.
3. Anahtar tablosunun kapasitesi dolduğunda eski kaydı yenisiyle değiştiren bir "yenileme" yapısı gerekliliği.

Makalenin geri kalanında 2. bölümde literatürdeki benzer çalışmalar, 3. bölümde özet algoritması seçimi, 4. bölümde anahtar – değer mimarisi ve FPGA gerçeklemesi, 5. bölümde kaynak tüketimi ve 6. bölümde sonuçlar ve gelecek çalışmalar ele alınacaktır.

2. İlgili Çalışmalar

Özet fonksiyonları ve anahtar – değer yapıları yüksek veri işleme hızı gerektiren birçok yapıda kullanılmıştır. Mahoney et al.[4] çalışmasında CAM yapısına alternatif olacak bir özet algoritması yapısı önermiş, paralel çalışan bu yapıda SRAM (Static Random Access Memory)'ler kullanmasına rağmen CAM'e göre çok daha az güç tüketen ve daha az gecikmeli bir sistem tasarımı yapabilmiştir.

Irfan et al.[5] TCAM (Ternary Content Addressable Memory) çalışmasında DRAM hafızasını 4 bölüme ayırarak CAM araması yapılacak hafıya bölüm sayısıyla ters orantılı olarak azaltmayı başarmıştır. Fakat aynı hafıza kısımlarını adresleyen çok fazla girdi olması durumunda oluşacak taşmaya karşın alternatif bir senaryo planlamamış ve çözümü sonraki çalışmalara bırakmıştır.

Istvan *et al.*[3] yüksek veri hızlarını hedeflediği anahtar – değer mekanizmasını ardışık düzenli ve ölçeklenebilir kullanmıştır. Önerdiği dinamik hafıza yapısı sayesinde farklı boydaki anahtar değerlerini destekleyebilmiştir. Bu da mimarinin karmaşık yapılardaki kullanılabilirliğini arttırmıştır. Çalışmasında 2 milyon girdi için 24 GB harici DDR3 (Double Data Rate 3) RAM (Random Access Memory) kullanmış ve 10 Gbps bağlantı hızına ulaşabilmiştir.

Prasanna *et al.*[2] 85 Gbps hızlarına ulaştığı çalışmasında 24 GB harici DDR3 RAM kullanmış ve mimarisini 1-16 milyon girdi için test etmiştir. Dinamik anahtar boyunu desteklediği mimarisi 16-128 bitlik anahtarlarla kullanılabilmektedir. Ardışık düzenli mimarisi 3 aşamadan (özet alma, inceleme ve uygulama) oluşmaktadır. Bu aşamalarda her incelenen girdi DFU (Data Forwarding Unit)'dan geçerek özet tablosunda ilgi yerde tutulmaktadır. Hafızaya yazma kısmındaki rasgele yerleştirme özelliği sayesinde hafizayı verimli şekilde kullanmıştır.

3. Özet Algoritması Seçimi

Bu çalışmada özet fonksiyonu olarak H3 (Universal Hash Function) kullanılmıştır. Bu seçimdeki temel etmen H3 algoritmasının ardışık düzenli yapıya uygunluğu ve çakışma oranının küçük veri seti için düşük olmasıdır [4,7].

Eğer H evrensel ise, o zaman herhangi bir x ∈ H için N boyutundaki herhangi bir S ⊆ H kümesi için, H'ye göre rastgele h oluştururuşak, x ile y arasında beklenen çakışma sayısı S'deki tüm ögeler için en çok N / M'dir (5). $H \longrightarrow \{1, \dots, M\}$ ve $\forall y \in S$:

$$Pr_{H \to h}[h(x) = h(y)] \leq 1/M, \quad (1)$$

$$C_{xy} = \begin{cases} 1, & \text{x ve y çakışmış ise} \\ 0, & \text{diğer} \end{cases}, \quad (2)$$

$$C_x = \sum_{y \in S, y \neq x} C_{xy}, \quad (3)$$

$$E[C_{xy}] = Pr(\text{x ve y çakışmış}) \leq 1/M, \quad (4)$$

$$E[C_x] = \sum_y E[C_{xy}] < N/M \quad (5)$$

Burada N sayısı yani seçilen örnek uzayın derinliği arttıkça çakışma oranının artacağı görülmektedir. Bunun yanı sıra algoritmanın ölçeklenebilir yapısı sistem mimarisine uygun olduğundan fonksiyon çıktısı daha küçük boyutlarda anahtar değeri üretimini desteklemektedir.

Carter ve Wegman tarafından tanımlanan H3 algoritmasında, *i x j* olarak tanımlanan bir Q boolean matrisiyle işlem yapılmaktadır [8]. Her $q \epsilon Q$ için q matrisinin k'ıncı satırını ifade eden bit dizisi bulunmaktadır, *q(k)*. Özeti alınacak dizinin k'ıncı elemanı da *x(k)* ile ifade edilecek olursa $h_q(x)$ özet fonksiyonu aşağıdaki gibidir (6): $h_q(x): A \longrightarrow B$ [9]

$$h_q(x) = x(1).q(1) \oplus x(2).q(2) \oplus \dots \oplus x(i).q(i) \quad (6)$$

Yukarıdaki formülde "." İşlemi "mantıksal ve", "⊕" işlemi "mantıksal özel veya" işlemini temsil etmektedir. Mantıksal işlem yapısının basitliği donanım ortamında kolay gerçeklenebilmesini sağlamaktadır. Q matrisi küçük bir hafıza kısmında saklanabilir. Bu sayede en az kaynak ve işlem gücü harcayarak özet alma işlemini gerçekleştirilmektedir.

Bu algoritmaya alternatif olarak incelenen bir başka algoritma ise, Bob Jenkins'in özet algoritması "lookup3" fonksiyonudur [8]. Lookup3 algoritmasının çakışma oranı uzun anahtar (32 bit, 64 bit, 96 bit) boyları için düşüktür. Fakat bu çalışmada veri seti derinliği 2048 olduğundan 11 bit uzunluğunda anahtar adresi kullanılmış ve bu uzunluktaki anahtar adresi için her iki algoritmanın da performansının yaklaşık aynı olduğu görülmüştür.

Her iki algoritma için de performans ölçümleri Python dilinde senaryolar yazılarak yapılmıştır. Test, 64 bit uzunluğunda 2048 girdi için uygun 11 bit uzunluğunda anahtar üretilerek yapılmıştır. Özeti alınacak veriler rasgele oluşturulmuş ve 64 bitlik kaynak ve hedef IP adreslerinin birleşimini temsil etmektedir. Anahtarlar ise

algoritmalara uygun olarak oluşturulmuş 11 bitlik değerlerdir. Özet tablosunun tutulduğu hafıza, anahtar değerine karşılık gelen bölmeler ve her bölme için 4 adet hücreden oluşmaktadır (Şekil 3). Hücre sayısı arttıkça çakışma oranı düşmektedir. Fakat hafıza alanı genişlediği için kullanılan BRAM sayısı da artmaktadır. Bu yüzden veri setini çok büyük olduğu çalışmalarda hafızanın verimli kullanılması üzerinde durulması gereken bir konudur.

Çizelge 1- Çakışma oranı

Çakışma Oranı (%)	1 Hücre	2 Hücre	3 Hücre	4 Hücre
Lookup3	41.3	8.69	1.41	0.24
Universal H3	41.65	9.1	1.56	0.24

Çizelge 1'de görüldüğü gibi her iki özet algoritması için de çakışma oranları tüm durumlarda yaklaşık olarak eşittir. Lookup3 algoritmasının donanım gerçeklemesi daha karmaşık olduğundan ve daha çok kaynak tüketeceğinden tercih edilmemiştir.

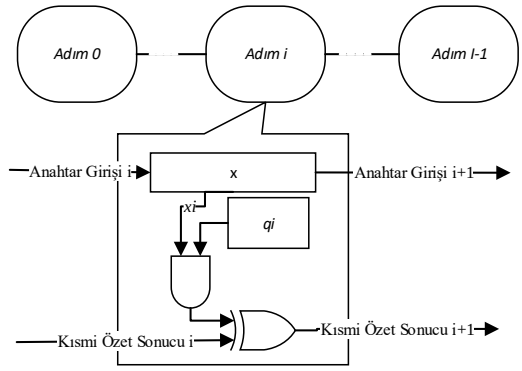
H3 algoritması ve dört hücreli hafıza yapısı kullanarak oluşturulan anahtar değer mekanizmasının çakışmanın yönetilebileceği ve kolay uygulanabilir olduğu görülmüştür.

4. Anahtar Değer Mimarisi ve FPGA Gerçeklemesi

Anahtar – değer yapıları önceki bölümlerde belirtildiği gibi birçok ağ altyapısında kullanılmaktadır. Bu çalışmada kullanılma amacı kaynak ve hedef IP adresi çifti için tutulan eşsiz bir değeri en hızlı şekilde bulmak ve getirmektir.

Tasarlanan mimarideki akış ağdan gelen IP paketlerini alıp parçalayarak 64 bitlik bir IP adresi çiftine dönüştürülerek başlamaktadır. Elde edilen IP adresleri H3 özet fonksiyonundan geçirilerek, özet tablosu adres bilgisi elde edilmektedir. Bir sonraki adımda ise tabloda tutulan değer adresi kullanılarak hafızadaki değer bölmesine ulaşılmaktadır.

H3 özet fonksiyonunun ardışık yapıya uygunluğu sayesinde tek bir saat darbesinde özet alma işlemi gerçekleştirilebilmiştir. Özet alma işleminin her aşaması Şekil 2'de gösterildiği gibidir. Burada anahtar boyu ne kadar artarsa kaynak kullanımı o kadar artacağından büyük anahtar boyundaki uygulamalarda kaynak kullanımı artacak ama algoritmanın donanımı ölçeklenebilir olduğu için büyük anahtar boylarına da uyum sağlayacaktır.



Şekil 2: Ardışık düzenli özet alma işlemi

Özet alma işlemi tamamlanan her sonuç anahtar olarak ikinci aşamaya girdi olarak verilir. Sisteme giren her girdinin bir saat darbesi sonrası özeti alınır ve sonrasında elde edilen anahtar değerine göre özet tablosunda ilgili hafıza bölmesine yönlendirilir. Her hafıza bölmesi 4 hücreden oluşmaktadır. N bitlik anahtar için 2^n bölme gerekmektedir. Küçük veri setleri için BRAM kullanımı okuma yazma hızı açısından harici RAM'lere göre avantaj sağlar. Yönlendirme yapıldıktan sonra işlem yapılacak hücrenin doluluğu kontrol edilir.

Hücre boş ise *Yaz* komutu uygulanır. IP çifti değeriyle birlikte ömür bilgisini temsil eden sayaç değeri ve anahtara karşılık gelen değer bilgisi eklenerek hafızaya yazılır. Eşsiz değer bilgisinin seçiminde ise değer hafızasındaki tüm adresler özdeş olduğundan ve adreslerin sırası önemli olmadığından tüm hafıza için sıralı artan değerler kullanılmıştır. Yazma işlemi bitirildiğinde bekleme durumuna gidilir ve yeni sistem girdisi beklenir.

Kontrol edilen hücre dolu ise ve sistemin girdisi olan IP adresleri hücrede tutulan IP çiftiyle aynı ise *Oku* komutu adımına geçilir. Okuma işlemi seçilen bölmedeki değer adresine giderek adreste bulunan değeri alır ve sistem çıktı olarak çıkış kapısına yönlendirilir. Kullanılan BRAM'ler iki kapılı seçildiğinden RAM'lerin giriş ve çıkış kapıları aynı anda kullanılabilmektedir. Bu sayede aynı saat darbesinde hafızaya yeni bir değer gelirken çıkış almak mümkün olmaktadır. Okuma işlemi yapıldıktan sonra yazma komutu gibi bekleme komutuna gidilir ve yeni girdi beklenir.

Şekil 3: Hafızadaki bölme ve hücre yapısı

Hücre dolu ve okuma işlemi yapılmayacak ise sonraki hücre kontrol edilir ve önceki tüm bu adımlar yeni hücre için de uygulanır. Hafıza kontrol işlemi bir komuta karar verilene kadar ya da anahtarın ait olduğu hafıza bölmesindeki tüm hücrelerin dolu olduğu anlaşılana kadar devam eder. Tüm hücrelerin dolu olması durumunda sistem güncelleme durumuna gider ve ilgili IP paketiyle ilgili bir hata uyarısı oluşturur. Bu aşamada sistemin yenileme mekanizması devreye girer ve ilgili bölmenin hücrelerinde bulunan en az kullanılmış kayıt silinerek yeni değer bu hücreye yazılır. Mimarinin operasyon algoritması, Algoritma 1'de verilmiştir.

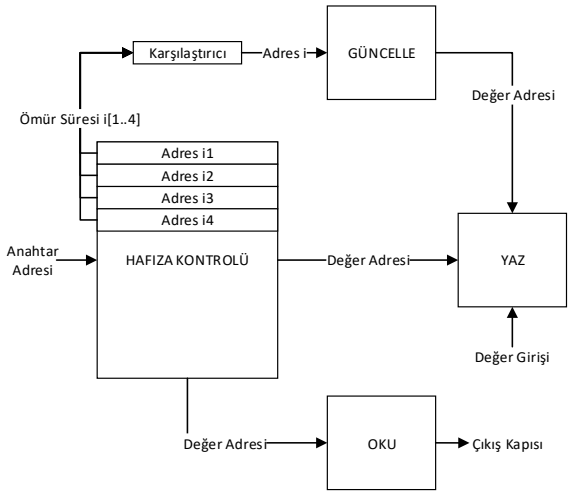
Algoritma 1 Operasyon

Değişkenler:
din_sd_val = Kaynak, hedef IP adreslerini içeren sistem girdisi
hash_val = H3 algoritmasıyla hesaplanan özet değeri
key_val = H3 çıktısı için tanımlanan anahtar değeri
state = Durum makinesi değişkeni
key_addr[0..3] = key_val'den üretilen hücre adres değeri
sd_val = Hücrede yazılı olan IP adresi çifti

dout_val = Özet tablosu hücresindeki değer çıktısı
value = Hücrede yazılı verinin değer kısmı
BRAM_din = BRAM giriş kapısı
life_time = Hücredeki kaydın yaşam süresini tutan sayaç değeri
F/E = Hücrenin doluluk/boşluk biti
err_flag = Hata durumu bayrağı

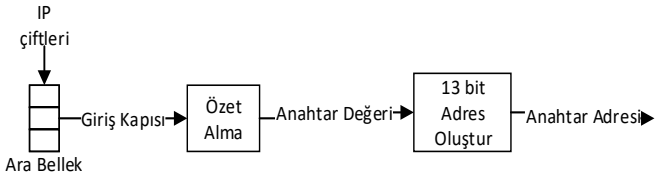
Adım 1: Özet Alma
```
1:  if din_sd_val = new then
2:   key_val <= hash_val
3:  end if
```
Adım 2: Hafıza Kontrolü
```
1:  Anahtar değerlerinde key_addr[0..3] oluştur
2:  if state = chk_mem then
3:   if key_addr[i] = empty then
4:        state <= write
5:   else
6:        if key_addr[i].sd_val = din_sd_val then
7:             state <= read
8:        else
9:             if i<3 then
10:                 i <= i+1
11:                 state <= chk_mem
12:            else
13:                 state <= update
14:            end if
15:       end if
16:  end if
17: end if
```
Adım 3: Uygulama
```
1:   if state = read then
2:    dout_val <= key_addr[i].value
3:    state <= idle
4:   end if
5:   if state = write then
6:    key_addr[i].din <= din_sd_val & value & life_time & F/E
7:    state <= idle
8:   end if
9:   if state = update then
10:  err_flag <= 1
11:  state <= refresh
12: end if
13: if state = refresh then
14: Bölmedeki tüm hücrelerdeki sayaç değerlerini kıyasla ve en
15: büyük değerliyi sil yerine yeni kaydı yaz
16: state <= idle
17: end if
```

Yenileme mekanizması için tabloda her hücreye ait bir yaşam süresi bulunmaktadır. Bu sayaç değeri hücrede her işlem yapıldığında azalır. Böylelikle yeni kayıt yapılacağında en az kullanılan kayıt silinebilmektedir. Sayaç değerlerini kıyaslamak için karşılaştırıcı kullanılmıştır. Karşılaştırıcının sonucunda seçilen adrese ait değer adresi kullanılarak yazma işlemine geçilmektedir. Yazma işlemi sonucunda bekleme durumuna gidilir ve yeni girdi beklenir. Bu çalışmada sayaç başlangıç değeri 128 olarak belirlenmiştir. Hafıza operasyon şeması Şekil 4'de gösterildiği gibidir.



Şekil 4: Hafıza operasyonu şeması

Mimarideki bir diğer bileşen ise sistemin bir girdiyi işleme sırasında gelecek olan yeni IP çiftlerini bekleteceği bir ara bellek tasarımıdır. Sistem her girdi için en geç 16 saat darbesi sürede işlemini tamamlamaktadır. 2048 girdi için düşünüldüğünde en kötü durumda 16KB'lık ara bellek sistem girdilerinin hiç birini kaybetmeden çalışma yapılması için yeterli olacaktır. Bu durumda tasarlanan anahtar değer sisteminin giriş akış şeması Şekil 5'deki gibi olacaktır.



Şekil 5: Sistemin giriş akış şeması

5. Kaynak Tüketimi

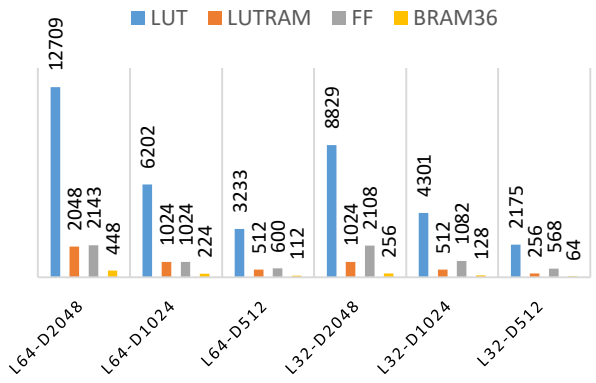
Bu bölümde tasarlanan mimarinin kaynak tüketimi açısından değerlendirilmesi yapılacak ve CAM yapısıyla kaynak tüketimi açısından kıyaslanacaktır.

Çizelge 2: Kaynak Tüketimi

Kaynak Tüketimi	LUT	LUT RAM	FF	BRAM36	Güç Tüketimi (W)
Anahtar Değer Mimarisi	171	0	194	10	0,371
CAM XILINX	12709	2048	2143	448	1,734

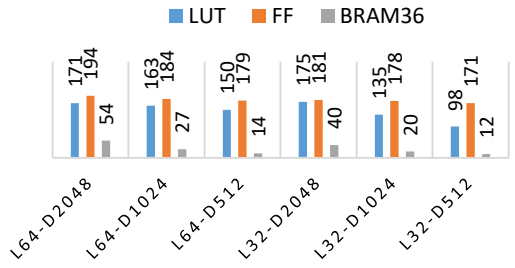
CAM teknolojisi tablo üzerinde arama yapmak için çok hızlı ve kusursuz eşleşme sağlayan bir yöntemdir. Fakat kaynak tüketimi ve güç tüketimi oldukça fazladır. Bu yüzden küçük IP yönlendirme tabloları için en iyi çözüm olsalar da büyük veri ile işlem yapan sistemler için uygulanılırlıkları azalmaktadır. Bu çalışmada tasarlanan sistem ile karşılaştırılacak CAM olarak XILINX firması tarafından tasarlanan CAM seçilmiştir. Bu CAM

hem BRAM'ler hem de yazmaçlar üzerinde gerçeklenebilir olarak tasarlanmıştır [10].



Şekil 6: XILINX CAM kaynak tüketimi
(L*n*-D*m*, *n*: Uzunluk, *m*: Derinlik)

Bu çalışmada önerilen mimaride BRAM'ler kullanıldığından, CAM gerçeklemesi de BRAM'ler kullanarak yapılmıştır. Aynı FPGA kullanılarak yapılan gerçeklemelerde, anahtar boyu 11 bit olarak belirlenmiş ve 2048 sistem girdisi için kaynak tüketimi tablosu Çizelge 2'de verilmiştir. Anahtar değer yapısı için giriş uzunluğu 64 bit derinlik ise 2048'dir. Benzer şekilde CAM tasarımının da uzunluğu 64 bit ve derinliği 2048 olarak ayarlanmıştır. Tasarımın gerçeklenmesinde FPGA olarak XILINX Virtex 7 serisi xc7vx690t kullanılmıştır.



Şekil 7: Anahtar değer mimarisi kaynak tüketimi
(L*n*-D*m*, *n*: Uzunluk, *m*: Derinlik)

Farklı giriş boyları ve veri seti derinlikleri için gerçekleme yapıldığındaki kaynak tüketimleri Şekil 6 ve Şekil 7'de görüldüğü gibidir. Gerçeklemelerin sonucunda uzunluk ve derinlik arttıkça CAM yapısındaki kaynak tüketimi artışının anahtar değer mimarisinine göre daha fazla olduğu görülmüştür.

6. Sonuçlar ve Gelecek Çalışmalar

Bu makalede ardışık yapılı bir anahtar değer yapısı tasarımı anlatılmıştır. Bu mimari hafıza içerisinde hızlı arama yapma imkanı sağlamaktadır. Mimari sabit anahtar uzunluğu ve BRAM kullanarak tasarlanmıştır. Hafıza hücrelerine veriler sıralı olarak yerleştirilmektedir. Yenileme mekanizması ömür süresi mantığına dayanmaktadır. H3 özet fonksiyonu az donanım kaynağı tüketerek özet tablosu ihtiyacını karşılamıştır.

Kaynak tüketimi konusunda XILINX CAM yapısına göre çok daha az donanım kaynağı tüketilmiş ve güç tüketimi açısından çok daha verimli bir yapı oluşturulmuştur. Böylelikle anahtar değer yapılarının veri seti derinliği arttıkça CAM'lere göre donanım kaynağı ve güç tüketimi açısından daha verimli olduğu görülmüştür.

Gelecek çalışmalarda değişken anahtar boyları için uygun özet algoritması araştırması yapılacaktır. Anahtar değer çiftlerini hafıza içerisine daha verimli yerleştirmek için yazma ve okuma mekanizması geliştirilecektir. Bu sayede değişken anahtar boyları ve farklı uzunluklardaki girdiler işlenebilecektir. BRAM'ler ile tasarlanan anahtar değer mekanizması derinliği düşük veri setleri için kaydırma yazmaçları kullanarak gerçeklenecek biraz daha fazla kaynak tüketilse bile çok daha hızlı çalışan bir sistem elde edilecektir.

7. Kaynaklar

[1] Wei Liang, Wenbo Yin, Ping Kang and Lingli Wang, "Memory efficient and high performance key-value store on FPGA using Cuckoo hashing," *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, Lausanne, 2016, pp. 1-4, doi: 10.1109/FPL.2016.7577355.

[2] D. Tong, S. Zhou and V. K. Prasanna, "High-Throughput Online Hash Table on FPGA," *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*, Hyderabad, 2015, pp. 105-112, doi: 10.1109/IPDPSW.2015.149

[3] Z. István, G. Alonso, M. Blott and K. Vissers, "A flexible hash table design for 10GBPS key-value stores on FPGAS," *2013 23rd International Conference on Field programmable Logic and Applications*, Porto, 2013, pp. 1-8, doi: 10.1109/FPL.2013.6645520.

[4] P. Mahoney, Y. Savaria, G. Bois and P. Plante, "Parallel hashing memories: an alternative to content addressable memories," The 3rd International IEEE-NEWCAS Conference, 2005., Quebec City, Que., 2005, pp. 223-226, doi: 10.1109/NEWCAS.2005.1496691.

[5] M. Irfan, R. C. C. Cheung and Z. Ullah, "Bank-selective Strategy for Gate-based Ternary Content-addressable Memory on FPGAs," 2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP), New York, NY, USA, 2019, pp. 288-291, doi: 10.1109/ASAP.2019.00019.

[6] L. Carter and M. Wegman, "Universal Classes of Hashing Functions," J. Computer and System Sciences, vol. 18, no. 2, pp. 143-154, 1979

[7] Ahmadi, Mahmood, Wong and Stephan, Hashing Functions Performance in Packet Classification, 2007

[8] B. Jenkins, "Function for producing 32bit hashes for hash table lookup," http://burtleburtle.net/bob/c/lookup3.c , 2006.

[9] M. Ramakrishna, E. Fu, and E. Bahcekapili, "Efficient hardware hashing functions for high performance computers," Computers, IEEE Transactions on, vol. 46, no. 12, pp. 1378–1381, Dec 1997.

[10] K.Locke, Parameterizable Content-Addressable Memory. https://www.xilinx.com/support/documentation/application_notes/xapp1151_ Param_CAM.pdf, 2011