SAÜ Fen Bil Der 20. Cilt, 3. Sayı, s. 441-450, 2016

Yüzde tabanlı String Eşleme Problemi için yeni bir donanım modülü tasarımı

İbrahim Şahin[1], Günay Temür[2*]

29.05.2015 Geliş/Received, 20.06.2016 Kabul/Accepted

doi: 10.16984/saufenbilder.99529

ÖZ

Bir verinin bir dizgi içerisinde veya bir gen yapısının bir DNA gen dizilimi içerisinde arama işleminin gerçekleştirilmesi için çeşitli algoritmalar kullanılmaktadır. Kullanılan bu algoritmalardan bazıları bize mutlak eşleşme olmadığı durumlarda olumsuz dönüt vermekte, bazıları ise "bunu mu arıyorsunuz" diye alternatifler sunmaktadır. Her iki algoritma da genel amaçlı PC'lerde saniyeler süren işlemler sonucunda bize dönüt verebilmektedir. Bu çalışmada bize hem mutlak eşleşmeyi hem de hedef dizgi içinde yüzdelik eşleşme oranlarının gerçekleştiği konumu veren FPGA çiplerine yönelik yüksek performanslı bir donanım modülü tasarlanmıştır. Geliştirilen modülün veri işleme hızı farklı PC'lerle karşılaştırılmış ve 2300 kata kadar daha hızlı arama gerçekleştirdiği karşılaştırma sonuçlarından elde edilen veriler ile doğrulanmıştır.

Anahtar Kelimeler: dizgi eşleme, FPGA, donanım modülü, yüzde tabanlı eşleme

A hardware module design for percentage-based String Matching Problem

ABSTRACT

Various algorithms are used to perform search operations in an array structure or a gene in a DNA gene sequence data. Some of these algorithms provide results if there is an exact matching between the source and the target arrays while some others provide us some alternative results and ask us "Did you mean this". Both types of algorithms return results after running for seconds on general purpose computers (PC) depending on the size of the data being searched. In this study, we designed a hardware module for FPGA chips to perform both exact and percentage based string matching. In the case of percentage based matching, the module provides a location in target string on which the highest percentage of matching between the source and target occurs. The module's performance was compared to different PCs and it was observed that it can return a result up to 2300 times faster than PCs.

Keywords: string matching, FPGA, hardware module, percentage based matching

* Sorumlu Yazar / Corresponding Author

1 Düzce Üniversitesi, Teknoloji Fakültesi, Bilgisayar Mühendisliği, Düzce - ibrahimsahin@duzce.edu.tr

2 Düzce Üniversitesi, Kaynaşlı Meslek Yüksekokulu, Bilgisayar, Düzce - gunaytemur@duzce.edu.tr

1. GİRİŞ (INTRODUCTION)

Bilgisayar bilimlerinde, çeşitli veri yapılarının (data structures) üzerinde bir bilginin aranması sırasında değişik arama algoritmaları kullanılmaktadır. Bunlara; doğrusal arama, ikili arama, çırpı (hash) arama algoritmalarını örnek verebiliriz [1-3]. Örneğin, bir veritabanındaki tabloda aranan bir veriyi hızlı bir şekilde bulmak veya veri karşılaştırma işlemlerini hızlandırmak, büyük bir dosyada aynı veya benzer kayıtları tespit etmek, DNA dizisinde benzer dizilimleri bulmak gibi işlemler bu algoritmaların çalışma alanlarına girer. Bu durum, arama algoritmalarının önemini daha da arttırmaktadır.

Arama algoritmaları genel olarak milyonlarca veri içerisinden istediğiniz bir veriye en kısa yoldan, en hızlı bir biçimde, en az bellek kullanımı ve en az kaynak harcayarak ulaşmamızı sağlayan algoritmalardır. Neredeyse her gün veri arama konusunda başvurduğumuz Google®'in arama motoru içerisinde çalışan Mobile-gedden, Moz Cast, Hummingbird gibi algoritmalar örnek verilebilir. Tabi ki Google alt yapısında bu algoritmaların hepsini çalıştırmaktadır. Mobile-gedden; mobil aramalarda kullanılan, Moz Cast; makale aramalarında kullanılan, Hummingbird; arama işlemlerinde uzun bir cümle yazdığımızda örneğin "Nerede kumpir yiyebilirim?" arama sorusuna sonuç olarak ana kelimeye ait alternatif "kumpir nedir", "kumpir tarifi", mekân adresleri gibi sonuçlar üreten algoritmalardır.

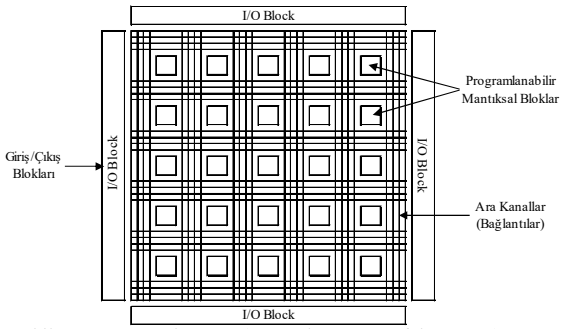
Bu makalede sunulan çalışmada; arama işlemlerini hızlandırmak ve bulunan en iyi yaklaşık sonucu geri döndürmek amacıyla, Alanda Programlanabilir Kapı Dizileri (Field Programmable Gate Array (FPGA)) çipleri üzerinde çalışabilecek bir donanım modülü tasarlanmıştır. Böylece daha ucuz, daha hızlı ve FPGA çiplerinin defalarca programlanabilme özellikleri sayesinde daha esnek bir yapının ortaya koyulması hedeflenmiştir. Tasarlanan modül, test verileri üzerinde işlemler yapılarak test edilmiş ve modülün ürettiği sonuçların doğrulaması yapılmıştır. Aynı veriler, değişik özellikteki genel amaçlı bilgisayarlarla işlenmiştir. Elde edilen sonuçlar kullanılarak, modülün veri işleme hızı bilgisayarlarla karşılaştırılmıştır.

Makalenin ikinci bölümünde tasarladığımız modül için kullandığımız donanım elemanı olan FPGA çipleri ve string arama algoritmaları hakkında kısa bilgiler verilmiştir. Üçüncü bölümünde, tasarlanan modül detaylarıyla anlatılmıştır. Dördüncü bölümde, gerçekleştirilen test çalışmaları ve elde edilen sonuçlar sunulmuştur. Son bölümde ise sonuçlar değerlendirilmiştir.

2. GENEL BİLGİLER (BACKGROUND INFORMATION)

2.1. FPGA Çipleri (FPGA Chips)

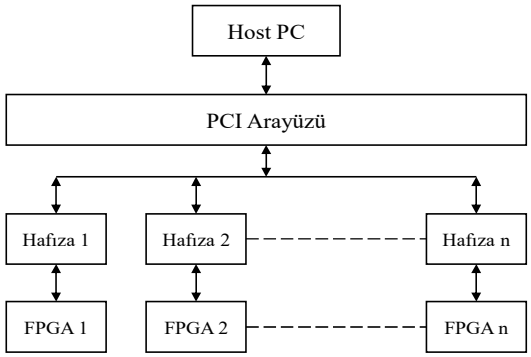
FPGA çipleri yeniden programlanabilir yapısı ve paralel işlem yapabilme yeteneği ile günümüzde tasarımcılara büyük kolaylık sağlayan entegre devrelerdir. FPGA'ların son yıllardaki gerçek zamanlı ve yüksek başarım gerektiren veri işleme uygulamalarındaki kullanımlarının oldukça yaygınlaştığı görülmektedir [4,5]. Şekil 1'de görüleceği gibi içerisinde bulunan yapılandırılabilir mantıksal bloklar (Configurable Logic Blocks (CLBs)), ara bağlantılar (interconnecting network) ve giriş/çıkış blokları (Input/Output Blocks (IOBs)) kullanıcının tasarladığı devreye göre programlanabilirler.



Şekil 1. FPGA genel yapısı. (General structure of the FPGA) [6]

2.2. F-CCMs (FPGA-based Custom Cumputing Machines)

FPGA tabanlı Özel Hesaplama Üniteleri (FPGA-based Custom Cumputing Machines (F-CCMs)) yüksek performans gerektiren işlemlerin FPGA çipleri üzerinde gerçekleştirildiği özel amaçlı bilgisayarlardır. Genel itibarı ile Yeniden Yapılandırılabilir Hesaplama (Reconfigurable Computing (RC)) olarak ta bilinirler. Bu platformlar genel amaçlı bir bilgisayar ve bu bilgisayara bağlı üzerinde bir ya da daha fazla FPGA çipi bulunan elektronik kartlardan oluşurlar [6-8]. Uygulama programları F-CCM'lere uyarlanırken, programın yüksek işlemci gücü gerektiren bölümleri özel olarak tasarlanmış donanım modülü kullanılarak FPGA çipleri üzerinde çalıştırılır ve bu sayede programın daha hızlı çalışması sağlanır [9]. F-CCM'lere uyarlanmış algoritmalar genel amaçlı bilgisayarlar ile karşılaştırıldığında şu ana kadar en düşük çalışma zamanını verdiği söylenebilir [10-11]. Şekil 2'de RC sistemlerin genel yapısı görülmektedir.



Şekil 2. RC sistem genel yapısı (General structure of the RC system)

Bu yapıda *n* adet FPGA çipi ve bunlara ait yerel hafızalar bulunur. FPGA kartı bilgisayar ile veri iletişimini genellikle bir PCI bus ara yüzü üzerinden yapar.

2.3. String Arama Algoritmaları (String Searching Algorithms)

Bilgisayar oratamında Crtl+F kombinasyonunu kullandığımız her noktada (internet tarayıcısı, masaüstü uygulamaları, Office uygulamaları vb.) karakter dizisi (*string*) arama algoritmalarını kullanmaktayız. Bu durum *string* arama algoritmalarının önemini oldukça arttırmaktadır ve bu algoritmaların çok hızlı çalışması beklenmektedir. *String* arama algoritmalarını iki gruba ayırmak mümkündür. Bunlar mutlak eşleme ve mutlak olmayan eşleme algoritmalarıdır.

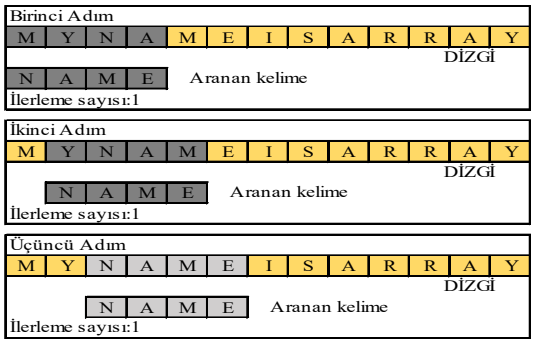
2.3.1. Mutlak String Eşleme Algoritmaları (Exact String Matching Algorithms)

Bilinen PC tabanlı klasik arama algoritmaları, aradığınız kısa bir verinin bir dizgi içerisinde tam karşılığını (%100 eşleşme) bulduğu anda olumlu dönüt vermektedir. Bu işlemi *string* fonksiyonları sayesinde gerçekleştirmektedir. *String* fonksiyonları, arama işlemi sırasında aranan kelimenin ilk karakteri ile hedef dizgi içindeki geçerli konumdaki karakteri karşılaştırırlar. Eğer eşleşme olursa, aranan kelime ve hedef dizgi içindeki takip eden karakterler aranan kelimenin son karakterine kadar karşılaştırılır. Karşılaştırma işlemi eşleşmenin olmadığı ilk noktada sonlandırılır. Eğer aranan kelimenin tüm karakterleri eşleşmiş ise fonksiyon **`true`** ya da hedef dizgi içinde bulunulan konumu geri döndürür.

Eğer kullanıcı aradığı kelimenin ilk karakterini girmeyi unutmuş veya yanlış girmiş ise ve hedef dizgimizde girilen verinin ilk karakteri ile eşleşen bir sonuç bulunmazsa **`false`** ya da **`null`** değeri geri döndürülür. Diğer karakterlerin karşılaştırılması yapılmaz. Arama algoritmaları bu sebeple hızlı çalışmaktadırlar.

Klasik saf dizi arama algoritmalarının dışında aynı %100 eşleşme mantığı ile çalışan birçok arama algoritması geliştirilmiştir. Bu algoritmalar sadece kelime arama amacı ile değil bir DNA gen dizilimi içerisinde arama yapmak gibi değişik amaçlar içinde de kullanılmaktadırlar. Kaba Kuvvet (Brute Force) , Horspoll ve Karp-Rabin gibi algoritmalar bu sınıftaki en çok bilinen algoritmalardan bazılarıdır.

Kaba Kuvvet (Brute Force) Algoritması: Bu algoritma aranan kelimeyi bulana kadar dizginin içerisindeki tüm karakterleri tek tek kontrol eder. Kelimenin ilk harf eşleşmesi gerçekleşmesi halinde diğer harflerinde kontrolünü yapar. Diğer harfler eşleşmezse hedef dizginin ikinci karakterinden itibaren eşleşme kontrolü yaparak devam eder. Herhangi bir sırada eşleşme olsa bile dizgi sonuna kadar aramasını gerçekleştirir. Şekil 3'te örnek olarak bir dizgi ve bir kelime verilmiş ve arama işlemi sonucunda eşleşme gösterilmiştir. Arama işlemini gerçekleştiren fonksiyona ait algoritma Şekil 4'te verilmiştir. Algoritma içerisinde kullanılan P[] pattern yani aranan *string*'i T[] text yani dizgiyi temsil etmektedir.

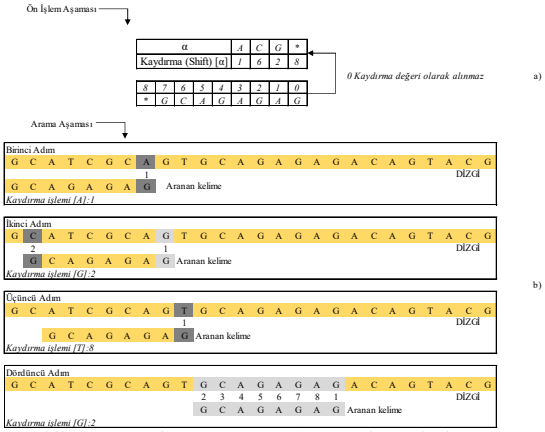


Şekil 3. Kaba Kuvvet arama aşaması (Brute Force search phase)

```
Brute_Force(T[0..n-1 ], P[0..m-1])
    for i ← 0 to n-m do
        j ← 0
    while j < m and P[j] = T[i+j] do
        j ← j+1
        if j=m return 1
    return -1
```
Şekil 4. Kaba Kuvvet arama algoritması (Brute Force search algorithm)

Horspool Algoritması: Genellikle metin veya DNA gen arama işlemlerinde etkin olarak kullanılan algoritmalardan bir tanesi de Horspool arama algoritmasıdır. Bu algoritma Brute Force algoritmasında olduğu gibi *string* ve dizgi içerisindeki karakterlerin karşılaştırılması ile sonuca ulaşmaya çalışır. Fakat algoritma alışılmışın dışında karşılaştırmaya ilk olarak aranan *string* ifadenin en sağındaki karakterden başlamaktadır. Algoritma arama işlemine başlamadan önce bir ön işlem safhasına sahiptir. Bu ön işlem safhasında arama yapılacak *string* ifadeyi oluşturan karakterlere bağlı bir tablo oluşturulur. Bu tablo karşılaştırma sırasında

eşleşme olmadığı durumlarda yapılacak olan kaydırma hesabında kullanılır. Şekil 5.a'da örnek bir aranan *string*' e ait kaydırma hesap tablosu verilmiştir.



Şekil 5. Horspool arama aşaması (Horspool search phase) [12]

Bu tablo arama yapılmaya başlandığı anda hızlı bir şekilde Şekil 6'da verilen algoritma tarafından oluşturulur. Aranan *string* ifadenin en sağındaki harf "0" olarak numaralandırılır ve sola doğru her harf sıralı değer alır. *String* içerisinde yinelenen harf olması durumunda harfin ilk aldığı değer kaydırma değeri olarak tabloya yazılır. Ardından karşılaştırma işlemine başlanır.

```
Shif_Tablo(P[0..m-1])
    for i ← 0 to size-1 do Table[i] ← m
    for j ← 0 to m-2    do Table[P[j]] ← m-1-j
    return Table
```

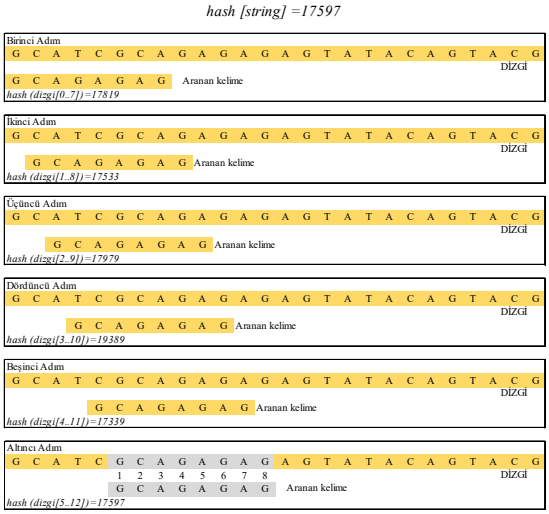
Şekil 6. Horspool ön işlem algoritması (Horspool pre-processing algorithm)

Karşılaştırma işlemi aranan *string* ifadenin en son karakterinin dizgide denk gelen karakter ile karşılaştırılmasıyla başlar. Şekil 5b.'de görüldüğü gibi eşleşme olmaması durumunda dizgideki karakterin tablo değeri kadar *string* sağa kaydırılır. Daha sonra ikinci adımda da görüldüğü gibi en sağ karakterin eşleşmesi gerçekleşirse ardından ilk karakterlerin karşılaştırılması yapılır. Bu durumda da yine eşleşme olmaması halinde dizgi içerisinde en sağ karaktere denk gelen dizgi karakterinin kaydırma tablosundaki değeri kadar *string* sağa kaydırılır. Bu şekilde arama işlemi devam eder. Algoritmayı asıl hızlı yapan özelliği ise, aranan *string* ifadenin içindeki harflerin dışında bir harfin dizgide bulunması durumudur. Bakılacak olursa Şekil 5.b'de aranan *string*' i oluşturan harflerin dışındaki bütün harfler için (*), Şekil 5.a'daki tabloda 8 değeri gösterilmektedir. Bu değer aranan *string* değerinin uzunluğuna bağlı olarak belirlenir. Bu 8 değeri, karşılaştırma sırasında dizgi içerisindeki karakterin aranan *string* içerisinde olmaması durumuna bağlı olarak *string*' imizin örnekte olduğu gibi 8 karakter birden sağa kaydırılmasını sağlamaktadır [13]. Bu şekilde aramaya devam ederek istenilen eşleşme sonucunun elde edilmesi durumunda algoritma sonlandırılır. Horspool algoritması Şekil 7'de verilmiştir.

```
Horspool (P[0..m-1]),(T[0..n-1])
    Shif_Tablo(P[0..m-1])
    i ← m-1
    while i ≤ n-1 do
        k ← 0
        while k ≤ m-1 and P[m-1-k]=T[i-k]  do
            k ← k+1
        if k=m return i-m+1
        else i ← i+Table[T[i]]
    return -1
```

Şekil 7. Horspool algoritması (Horspool algorithm)

Karp-Rabin Algoritması: Bu algoritma aranan bir *string* ifade için ön işlem safhası olarak bir *hash* değeri hesaplar. Aynı zamanda aranan *string* ifadenin uzunluğu da hafızada tutulur. Daha sonra algoritma dizgi içerisinden aranan kelime uzunluğu kadar *string*' in *hash* değerini hesaplar. Eğer iki *string* aynı *hash* değerine sahipse, aynı olabilirler. Fakat *hash* değerleri farklı ise zaten kesinlikle eşleşme olmadığı bilinir. Bu özellik sayesinde *string* arama işlemi gerçekleştirir. Dizgi içerisinde Brute Force algoritması mantığı ile *string* ifade tek karakter ilerler ve her ilerlemede dizgi içerisinde karşılık gelen *string* uzunluğundaki ifadenin *hash* değeri yeniden hesaplanır. *Hash* değerinin hesaplanması için özel bir formül yoktur. Kendimizin oluşturacağı karmaşık matematiksel işlemler bu hesaplamada kullanılabilir. Eğer karşılaştırma sonucunda aynı *hash* değerlerine sahip bir sonuç elde edilirse eşleşme olabilir ve bu sebeple kesin eşleşme sonucu için *string*' ler kendi içerisinde karşılaştırılır. Algoritmaya ait örnek Şekil 8'de [14], basit bir ön işlem fonksiyonu örneği (*hash*) matematiksel işlemi ve arama algoritması Şekil 9'da verilmiştir.



Şekil 8. Karp-Rabin arama aşaması (Karp-Rabin search phase) [14]

```
// ASCII G = 71, C = 67, A = 65
Hash ("GCAGAGAG") =
(71*2^7)+(67*2^6)+(65*2^5)+(71*2^4)+(65*2^3)+(71*2^2)+(65*2^1)+71 = 17597
RabinKarp (T[1..n], P[1..m])
hpattern := hash(P[1..m]); hs := hash(T[1..m])
for i ← 1 to n-m+1
    if hs = hpattern
        if T[i..i+m-1] = P[1..m]
            return i
    hs := hash (T[i+1..i+m])
return not found
```

Şekil 9. Karp-Rabin örnek ön işlem ve arama algoritması (Karp-Rabin sample pre-processing and search algorithm)

2.3.2. Mutlak Olmayan String Eşleme Algoritmaları (Non-Exact String Matching Algorithms)

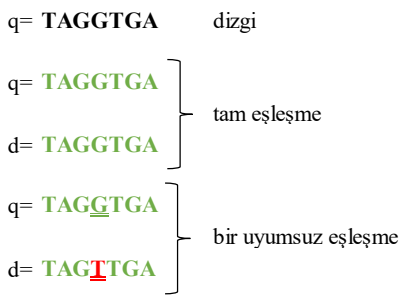
Bu algoritmalar var olan bir dizgi içerisinde aradığımız verinin her bir karakterini tek tek karşılaştırarak, eşleşme sonuçlarında %100 eşleşme olmadığı durumlarda bile en iyi eşleşme sonucunu ve konumunu geri döndürürler. Önceden yapılan indekslenmiş arama içeriklerine göre de tahminde bulunarak "bunu mu demek istediniz?" şeklinde alternatif sonuçlar vermektedirler.

Edit distance algoritması olarak bilinen Levenshtein distance algoritması önemli bir örnektir. Algoritma temel olarak iki kelimenin birbirlerine olan benzerliklerini ölçmek amacıyla kullanılır. Benzerlik ölçümü, aranan kelimenin, hedef dizgi içerisindeki hedef kelimeler ile uzaklık hesaplaması yapılarak gerçekleştirilmektedir. Sonuç tek bir sayısal değer olarak hesaplanmakta ve bir kelimenin diğerine dönüştürülmesi için gerekli olan işlem maliyetini vermektedir. Şekil 10'da örnek bir işlem maliyeti hesabı verilmiştir. Yuvarlak içine alınan "1" işlem maliyetini gösterir. Örnekteki "monkey" kelimesi 1 işlem maliyeti ile "money" kelimesine dönüştürülebilir. Levenshtein algoritmasına göre karakter silme, karakter değiştirme ve karakter ekleme işlemleri bir maliyet değerine sahiptir.

		m	o	n	k	e	y
	0	1	2	3	4	5	6
m	1	0	1	2	3	4	5
o	2	1	0	1	2	3	4
n	3	2	1	0	1	2	3
e	4	3	2	1	0	1	2
y	5	4	3	2	1	0	1

Şekil 10. Levenshtein algoritması örnek işlem maliyeti (Levenshtein algorithm sample process cost)

Levenshtein sonucuna göre gerekli olan işlem maliyeti ne kadar az ise iki kelime arasındaki benzerlik o kadar fazladır. Bu algoritma bir kelime üzerinde uygulandığı gibi bir cümle, bir DNA gen dizisi üzerinde veya protein zincirlerin içerisinde istenilen bir sıra diziliminin aranmasında da uygulanarak, benzerlik oranını rahatlıkla çıkarabilir. Bunun için temelinde aranan kelime ile dizgi içerisinde birbirine benzer karakter ifadelerin geçiyor olması gerekmektedir. Ayrıca bu benzerliklerin hesaplanması ve yakın kelimelerin bize sunulması için çok geniş ve sağlam bir kelime hazinesine sahip database'e ihtiyaç vardır. Şekil 11 algoritmanın bize bire bir eşleşme olmadığı durumlarda da dönüt verebileceğini göstermektedir.

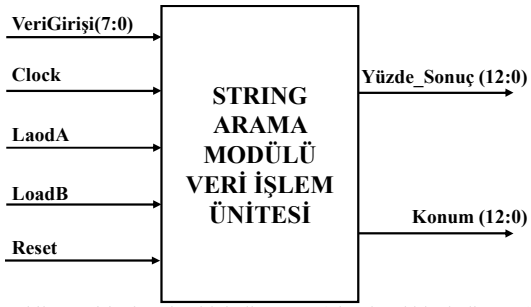


Şekil 11. Genom eşleme (Genome matching)

3. STRING ARAMA İÇİN FPGA MODÜL TASARIMI (FPGA MODULE DESIGN FOR STRING MATCHING)

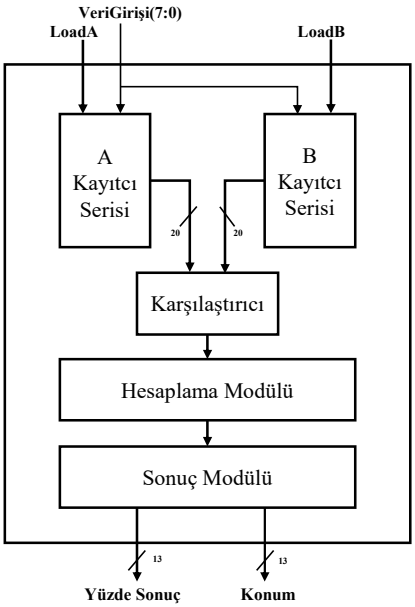
Bu çalışmada, arama işlemini yazılımsal çözümlere göre daha hızlı gerçekleştirecek FPGA tabanlı bir donanım modülü tasarlanmıştır. Bu modül aranan kelimenin hedef dizgi içerisindeki benzerlik oranının en yüksek olduğu konumu ve benzerlik yüzdesini sonuç olarak verir.

Tasarlanan modül iki seviyeden oluşmaktadır. Modülün en üst seviye blok diyagramı Şekil 12'de gösterilmiştir. Modül 8 bitlik *VeriGirişi* girişi ile birlikte birer bitlik *LoadA*, *LoadB*, *Clock* ve *Reset* girişlerine sahiptir. Ayrıca sonuç değerinin alındığı 13 bitlik bir *Yüzde_Sonuç* çıkışı ve en iyi sonucun konumu bildiren 13 bitlik *Konum* çıkışı mevcuttur. Tasarlanan modül, bir donanım tanımlama dili olan VHDL'de kodlanmış ve Xilinx ISE 14.5 yazılımı kullanılarak Virtex 6 XC6VLX75T FPGA çipi için sentezlenmiştir.



Şekil 12. Birinci seviye blok diyagram (First level block diagram)

İkinci seviye blok diyagram (sistemin içyapısı) Şekil 13'te de görüldüğü gibi 5 ana bölümden oluşmaktadır.

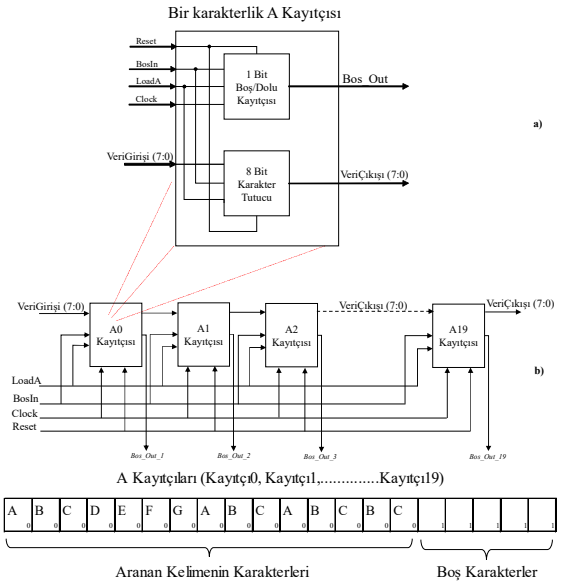


Şekil 13. İkinci seviye blok diyagram (Second level block diagram)

Bu bölümler; aranan kelimenin ve arama yapılan dizginin geçerli konumundaki kısmın tutulduğu kayıtçı üniteleri, aranan kelime ile hedef dizgiyi anlık olarak karşılaştıran karşılaştırma ünitesi, karşılaştırma sonucunda yüzdelik eşleşme değerini belirleyen hesaplama ünitesi ve bulunan en yüksek değer ile bu değerin elde edildiği konumunun tutulduğu sonuç üniteleridir.

A kayıtçı serisi içinde bulunan bir karakterlik kayıtçılar, normal bir kayıtçı mantığından farklı şekilde tasarlanmıştır. Şekil 14.a'da görüldüğü gibi kayıtçıda karşılaştırmaya dahil edilecek bir karakter tutulup tutulmadığını belirten boş biti bulunmaktadır. Modül resetlendiği anda A kayıtçıları boş bitlerine "1" bilgisi aktarılmaktadır. Bu özellik kayıtçıların içerisinde herhangi bir veri olup olmadığı durumunu göstermektedir. Veri kaydedilmeye başlandığı andan itibaren veri ile dolan kayıtçıların boş biti "0" durumuna geçmektedir. Bu sayede şekil 14.c'de görüldüğü gibi 20 karakterden kısa kelimeler de, kayıtçıların içerisindeki sıfırların sayısı bize kaç karakterlik bir kelime araması yapacağımızı vermektedir.

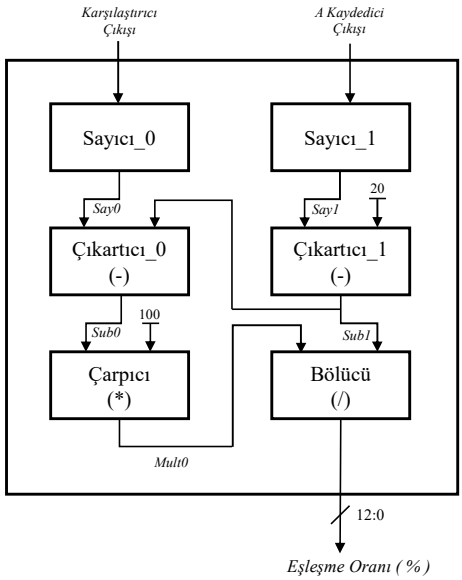
A ve B kayıtçı serileri, içinde birer karakterlik kayıtçılar ile birbirlerine seri olarak, bir zincir oluşturacak şekilde bağlanmıştır (Şekil 14.b). Bu şekildeki bir bağlantı, kayıtçıların daha kolay kontrol edilmesini sağlamakta ayrıca hafızadan okunan bilgilerin kayıtçılara aktarılmasında da herhangi bir dezavantaj oluşturmamaktadır.



Şekil 14. Kayıtçıların zincir bağlantısı ve A kayıtçısı içyapısı (Chain link of registeries' and A-register ınternal structure)

Kayıtçı serilerine sıra ile gelen *LoadA* ve *LoadB* sinyalleri, bu iki kayıtçı zincirine verilerin düzenli bir şekilde okunmasını ve güncellenmesini kontrol etmektedirler. Öncelikle *LoadA* sinyali "1" olmakta ve aranacak kelime A kayıtçılarına sıra ile aktarılmaktadır. Aktarma işlemi tamamlandıktan sonra *LoadA* sinyali "0" olmakta ve A kayıtçısına aktarım durmaktadır. Ardından B kayıtçılarına içinde arama yapılan dizginin aktarılması için *LoadB* sinyali "1" yapılmakta ve dizgimiz sıra ile B kayıtçılarına aktarılmaktadır. Her *LoadB=1* sinyali alındığında, kayıtçılar tarafından tutulan mevcut veriler zincirde bir birim ilerletilmekte ve *VeriGirişi* üzerinden gelen yeni veri ise, zincirde oluşan boşluğa yerleştirilmektedir. Dizgi aktarılma işlemi başladığı anda karşılaştırma işlemi de başlamaktadır.

A ve B kayıtçı serileri içerisindeki veriler tek tek ve aynı anda karşılaştırıcılarda (20 adet) karşılaştırılır. Karşılaştırıcılar A kayıtçılarından gelen boş sinyaline duyarlı olarak çalışmaktadırlar. A kayıtçısından gelen boş sinyalinin mantıksal "1" olması durumunda karşılaştırmaya gerek olmadan *Cikis* sinyali (*Cikis=1*) üretilir. Boş sinyalinin "1" olması, A kayıtçısı içerisinde veri olmadığını göstermekte ve hesaplama ünitesinde bu kayıtçının yüzde hesaplanmasına dâhil edilmemesini sağlamaktadır. Aynı şekilde, karşılaştırma işleminin sonucunda, karşılaştırıcıların herhangi birinin eşleşmesi durumunda *Cikis* sinyali *(Cikis=1)* üretilir. Eşleşme olmaması durumunda ise *Cikis* sinyali (*Cikis=0*) üretilir. Bu adımdan sonra çıkışlar şekil 15'te görülen hesaplama modülüne aktarılır.



Şekil 15. Hesaplama modülü (Calculation module)

Hesaplama modülü bize eşleşmelerin oranını veren modüldür. Bu modül, içerisinde matematiksel işlemler yapan üniteler barındırır. Modül içerisinde yer alan iki adet sayıcı karşılaştırıcıların çıkışlarındaki mantıksal sonuçları sayar. *Sayıcı_0* A ve B kayıtçıları arasındaki eşleşme olmayan durumları yani *eşleşmeyen karakter sayısını* bize vermektedir. *Sayıcı_1* ise A kayıtçıları içerisindeki boş kayıtçı sayısını saymaktadır. Sayma işleminin ardından çıkan her iki sonuçta ayrı ayrı hesaplama modülü içerisindeki çıkarma ünitelerine aktarılmaktadır.

FPGA çiplerinin sahip olduğu özellikten dolayı sayma işlemi combinational olarak gerçekleşmektedir. PC'lerdeki programlama yapısında ise sayma işlemi ardışıl olarak döngü içinde gerçekleştirilir. Ardışıl sayıcı, sayma işlemini arttırma yöntemi ile yani sıra ile gerçekleştirir. Bu durum sayma işleminin zaman alması anlamına gelmektedir. Fakat combinational sayıcılar sayma işlemlerini anlık olarak gerçekleştirirler. Sonuç tek bir clock işlemi sonrasında çıkışa aktarılır.

Hesaplama modülü çıkışında yüzdelik sonuç elde edebilmek için düzenlemiş bir hesaplama formülü (1)'de verilmiştir.

$$\frac{eşleşen\ karakter\ sayısı * 100}{aranan\ karakter\ sayısı} = eşleşme\ oranı\ (\%) \quad (1)$$

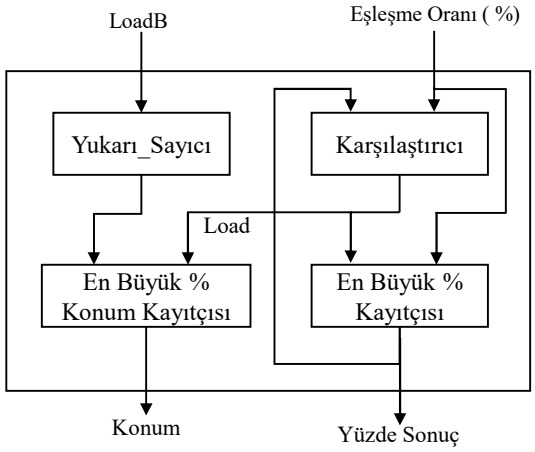
Modül içerisinde *Çıkartıcı_0* ve *Çıkartıcı_1* adında 2 adet çıkarma ünitesi mevcuttur. Modülümüzde maksimum 20 karakterlik arama yapılabildiğinden *Sayıcı_1*'den gelen *boş karakter sayısı* 20 sabit değerinden çıkartılarak *aranan kelimenin karakter sayısı* bulunur (2). Bu değerden karşılaştırma sonucunda *eşleşmeyen karakter sayısı Çıkartıcı_0* sayesinde çıkartılarak *eşleşen karakter sayısı* bulunur (3).

$$aranan\ karakter\ sayısı = 20 - boş\ kayıtçı\ sayısı \quad (2)$$

$$eşleşen\ karakter\ sayısı = aranan\ karakter\ sayısı - eşleşmeyen\ karakter\ sayısı \quad (3)$$

Eşleşen karakter sayısı formül (1)' de görüldüğü gibi önce *Çarpıcı* sayesinde 100 değeri ile çarpılır ardından *aranan kelimenin karakter sayısına* bölünerek *Eşleşme Oranı (%)* hesaplanmış olur.

Şekil 16'da görülen modül içerisindeki *Karşılaştırıcı*, arama yapılan dizginin her konumu için anlık olarak yeniden hesaplanan *Eşleşme Oranını* o ana kadar bulunmuş en yüksek *Eşleşme Oranı* ile tekrar karşılaştırır. Karşılaştırma da yeni gelen *Eşleşme Oranının* büyük olması durumunda modül içerisinde kullanılan *En Büyük % Kayıtçısına Load* sinyali giderek yeni gelen büyük değerin eski değerin yerine yazılması sağlanmıştır. Bu işleme paralel olarak modül içerisinde tasarlanan bir yukarı sayıcı yardımıyla her gelen değerin konumu sayılarak aynı mantıkla en yüksek değere sahip olan eşleşmenin konum bilgisi *En Büyük % Konum Kayıtçısı* içerisinde saklanmıştır. Sonuçta eşleşmeden gelen en iyi yüzdelik eşleşme ve hangi konumda gerçekleştiği bilgisi çıkışa aktarılmıştır. Sistem resetlendiği anda *En Büyük % Kayıtçısı*' na ve *En Büyük % Konum Kayıtçısı'* na "0" bilgisi aktarılır.



Şekil 16. Sonuç modülü (Result module)

Şekil 17'de modülün çalışması örnek veri üzerinde gösterilmiştir. Bu örnekte "PROBLEM" kelimesi hedef dizgi içinde aranmaktadır. Altıncı konumda %42 eşleşme tespit edilmiştir. Bu konum ve elde edilen yüzdelik değer ilgili kayıtçılarda bu anki en iyi sonuç olarak tutulur. Arama işleminin devamında dokuzuncu konumda %100 eşleşme tespit edilmiştir. Bu durumda öncekine göre daha iyi bir eşleşme oranı bulunduğundan, yeni eşleşme değeri ve bu değerin elde edildiği konum bilgisi ilgili kayıtçılara aktarılır.

Şekil 17. Modülün çalışması (Operation of the module)

4. KARŞILAŞTIRMALI TEST SONUÇLARI (COMPARATIVE TEST RESULTS)

Bu çalışmada tasarlanan *string* arama modülü, Xilinx firması tarafında üretilen XC6VLX75T numaralı en düşük özellikli Virtex 6 çipi için sentezlenerek, modülün çip istatistikleri incelenmiştir. Modülün belirli uzunluktaki bir veri içerisinden aranan veriye en iyi yüzde (%) ile ulaşma süresi, ISE benzetim programı kullanılarak elde edilmiş ve farklı özelliklere sahip iki bilgisayar ile aynı işlemler gerçekleştirilerek süreler karşılaştırılmıştır. Tablo 1'de modüle ait sentezleme sonuçları verilmiştir. Sentezleme sonuçunda kullanılan IOB (input/output pinleri) dikkate alındığında 6 adet modülün aynı anda tek bir çipe yerleştirilebileceği ve aynı anda birden fazla arama işlemlerinin yapılabileceği görülmektedir.

Tablo 1. Virtex-6 FPGA çip istatistikleri (Virtex-6 FPGA chip statistics)

Cihaz	Slice Reg. Sayısı	LUTs Sayısı	Slice FFs Sayısı	Bounded IOBs Sayısı
XC6VLX75T	456/93120	444/46560	260/655	39/240

Modülün çalışma performansını karşılaştırmak için özellikleri Tablo 2'de verilen iki adet bilgisayar seçilmiştir. Seçilen bilgisayarlardan PC-1 dört çekirdekli Hyper Treading (HT) teknolojisine sahip 4. nesil i7 işlemcili, PC-2 çift çekirdekli HT teknolojisine sahip olmayan işlemcili bilgisayarlardır.

Tablo 2. Deneylerde kullanılan pc'lerin özellikleri (Features of the pc used in the experiments)

PC Adı	CPU Hızı (Ghz)	CPU Cache Bellek (KB)	RAM (MB)	Hafıza Tipi	CPU Türü	Sistem Türü
PC-1	2.4	4096	8192	DDR3	İntel	64 Bit
PC-2	3.0	512	2048	DDR2	İntel	32 Bit

Deneylerde kullanılan 10KB, 20KB, 40KB, 60KB, 80KB, 100KB boyutlarındaki test verileri ayrı ayrı *.txt dosyaları içerisinde tutulmaktadır. Bilgisayarların, oluşturulan bu test verilerinde arama başarımının ölçülmesi için Şekil 18'de görülen C++ kod parçası geliştirilmiştir. Geliştirilen uygulama derleme aşamasında "Maximize Speed" (/O2) optimizasyon seçeneği seçilerek derlenmiştir. Bu program parçasının çalışma zamanı karmaşası $O(nm)$'dir, n içinde arama yapılan dizgi uzunluğu m ise aranan kelimenin karakter uzunluğudur. Tablo 3'te, geliştirilen uygulamanın PC1 ve PC2 üzerindeki performansı, çalışma kapsamında geliştirilen modülün performansı ile karşılaştırılmıştır. Arama işleminde bulmak istediğimiz kelimenin, test dizgisi içerisinde nerede olduğunun önemi yoktur. Çünkü uygulamamız test dizgisinin bütün konumlarını tek tek aramaktadır. Hangi konumda en iyi sonuca ulaşacağını bilmediğimizden %100 başarı sağlansa bile son veriye kadar arama işlemi devam etmektedir.

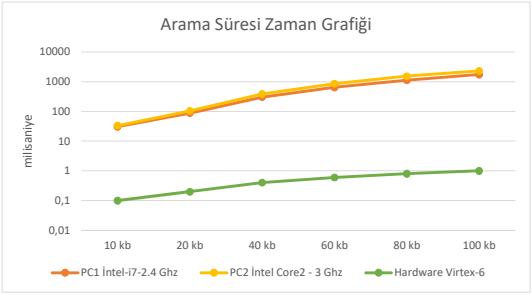
```
while(!Myfile.eof())
{
        data[i] = Myfile.get();
        i++;
}
p1 = data;
for (k = 0; k < strlen(data) - sayac; k++) {
        empty = 0; p2 = search; p3 = p1;
        for (l = 0; l < sayac; l++) {
                if (*p3 == *p2) {
                        empty++; p3++; p2++;
                }
        }
        if (empty>eslesme) {
                eslesme = empty;
        }
        p1++;
}
eslesme_orani=eslesen_karakter_sayisi *100/aranan_karakter_sayisi;
```

Şekil 18. Yüzde (%) eşleşme için arama algoritması (Search algorithm for percent match)

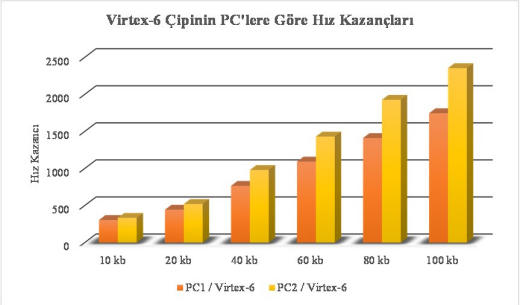
Tablo 3. PC'lerin ve Virtex-6 üzerinde tasarlanan modülün aranan kelimeyi eşleme süreleri (PCs and designed the modul on Virtex-6 match time for search word)

Dosya Uzunluğu (KB)	PC-1 (milisaniye)	PC-2 (milisaniye)	Hardware (milisaniye)
10	30.247	33.131	0.099
20	88.072	103.427	0.199
40	303.693	388.681	0.397
60	650.536	851.438	0.595
80	1119.358	1528.361	0.794
100	1731.679	2334.928	0.992

Şekil 19'da modülün virtex-6 çipi üzerinde çalışma sürelerinin PC'lerdeki çalışma süreleri ile karşılaştırılması görülmektedir. Tasarlanan modülün veri işleme süresi ile PC'lerin veri işleme süreleri arasında çok fazla fark olduğundan grafikte dikey eksen logaritmik skalada gösterilmiştir. Virtex-6'da modül test verisinin boyutunun değişmesine rağmen milisaniye cinsinden çok fazla değişim göstermemektedir. PC'lere bakıldığında ise test verisinin artması ile zamanın da yüksek oranlarda arttığı görülmektedir. Şekil 20'de görülen Virtex-6' nın karşılaştırmalı grafiğine bakıldığında, 100 KB test veri için tasarlanan modülün PC1'e göre 1500 kat, PC2'ye göre ise yaklaşık 2300 kat daha hızlı olduğu görülmüştür.



Şekil 19. Arama süresi zaman grafiği (The time graph of search time)



Şekil 20. Sentezlenen modülün PC'lere göre hız kazançları (According to PCs, synthesized modul's speed gains)

5. SONUÇ VE TARTIŞMA (RESULTS AND CONCLUSIONS)

Yüzde tabanlı dizgi eşlemede PC tabanlı algoritmaların performansları Tablo 3'te de görüldüğü gibi dizgi uzunluğu arttıkça düşmektedir. Bu durum metin tabanlı arama veya DNA gen eşlemesi içinde aynıdır. Aranılan sonuçlara ulaşmak oldukça zaman almaktadır. Bu çalışmada geliştirilen modül ile bu problem çözülmeye çalışılmıştır ve yine Tablo 3'te görüldüğü gibi FPGA çipi ile yapılan denemelerde PC'lerde saniyeler mertebesinde tamamlanan arama işlemi tasarlanan modül sayesinde mikro saniyelerde tamamlanabilmektedir. Bu çalışmanın devamında geniş bir veri tabanı üzerinde alınan sonuçlar neticesinde bir tahminleme yaparak "bunu mu demek istediniz" şeklinde alternatif sonuçlar sunan yeni modül tasarımları planlanmaktadır.

KAYNAKLAR (REFERENCES)

[1] S. Jose, The Programmable Logic Data Book, Xilinx Inc, CA, 1994.

[2] D. Knuth, The Art of Computer Programming 3 (3rd ed.), Addison-Wesley, 1997, pp. 396-408.

[3] T. H. Cormen, C. E. Leiserson ve R. L. Rivest, Introduction to Algorithms, MIT Press and McGraw-Hill, 1990.

[4] A. Dehon, The Density Advantage of Reconfigurable Computing, IEEE Computer, 2000, pp. 33,41-49.

[5] D. Knuth, The Art of Computer Programming, vol. 3, 1973, p. 506–542.

[6] S. Qasim, A. S.A. ve B. Almashary, An Overview of Advanced FPGA Architectures for Optimized Hardware Realization of Computation Intensive Algorithms, *IMPACT'09*, 2009.

[7] F. Rincon ve L. Teres, "Reconfigurable Hardware Systems", *International Semiconductor Conference*, 1998.

[8] I. Sahin, *"A Compilation Tool for Automated Mapping of Algorithms onto FPGA Based Custom Computing Machines"*, Raleigh-USA: NC State University, 2002.

[9] I. Sahin, C. Gloster ve C. Doss, “Feasibility of Floating-Point Arithmetic in Reconfigurable Computing Systems”, *NASA Military and Aerospace Applications of Programmable Devices and Technology Conference*, Washington, DC., 2000.

[10] K. Sridharan ve T. Priya, "A Hardware Accelerator and FPGA Realization for Reduced Visibility Graph Construction Using Efficient Bit Representations", *IEEE Transactions on Industrial Electronics,* cilt 54, no. 3, 2007.

[11] İ. Koyuncu ve İ. Şahin, “Generic Fpga Modules for Integer 2D and 3D Transformations”, *12th. Conference for Computer Aided Engineering and System Modeling with Exhibition*, Antalya, Turkey, 2007.

[12] H. R.N., *Practical fast searching in strings,* Software - Practice & Experience, 1980, pp. 10(6):501-506.

[13] A. Aho, Algorithms for finding patterns in strings, *Handbook of Theoretical Computer Science*, Elsevier, Amsterdam., 1990, pp. 255-300.

[14] K. R.M. ve R. M.O., *Efficient randomized pattern-matching algorithms,* IBM J. Res. Dev., 1987, pp. 31(2):249-260.