Gazi Üniv. Müh. Mim. Fak. Der.
Cilt 25, No 1, 83-91, 2010

J. Fac. Eng. Arch. Gazi Univ.
Vol 25, No 1, 83-91, 2010

ANAHTARLAMA FONKSİYONLARI İÇİN YENİ YAKIN MİNİMUM SADELEŞTİRME ALGORİTMASI

Fatih BAŞÇİFTÇİ ve Şirzat KAHRAMANLI*
Elektronik ve Bilgisayar Eğitimi Bölümü, Teknik Eğitim Fakültesi, Selçuk Üniversitesi, Kampüs/Konya
*Bilgisayar Mühendisliği Bölümü, Mühendislik Mimarlık Fakültesi, Selçuk Üniversitesi, Kampüs/Konya
basciftci@selcuk.edu.tr, sirzat@selcuk.edu.tr

(Geliş/Received: 04.02.2009 ; Kabul/Accepted: 03.11.2009)

ÖZET

Anahtarlama fonksiyonlarının sadeleştirilmesi tasarımcılara daha kısa zaman süresinde, daha sade lojik devreler tasarlama imkanı sağlamaktadır. Sadeleştirilmiş olan bir fonksiyon daha az güç tüketimi, daha az hacim ve daha az maliyet gerektirir. Bu konu ile ilgili olarak geliştirilen yöntemlerin çoğu iki ana adımda gerçekleştirilir. Birinci adımda, asal çarpım terimlerinin tümü belirlenir. İkinci adımda fonksiyonu sadeleşmiş olarak örtecek, esas asal çarpım terimler kümesi belirlenir. Anahtarlama fonksiyonlarını sadeleştirecek algoritmaların tümü $O(2^n)$ karmaşıklığına sahiptirler. Araştırmalar göstermiştir ki n'in çok yüksek değerlerinde esas asal çarpım terimlerin tam kümesini belirleme yöntemi pratik olarak gerçekleştirilemez duruma gelmektedir. Bu yüzden bu çalışmada, asal çarpım terimlerin belli kıstaslara cevap verecek alt kümeleri oluşturularak, doğrudan örtme prensibine dayanan yakın minimum sadeleştirme algoritması geliştirilmiştir. Geliştirilen algoritma çeşitli problemler üzerinde test edilmiş ve dünyaca örnek olarak kabul edilen ESPRESSO algoritması ile karşılaştırılmıştır. Karşılaştırma kıstasları olarak algoritmaların; çözüm sonucunda buldukları çarpım terimlerinin toplam ifadelerinin sayısı, çözüme ulaşma süreleri ve çözüme ulaşırken kullandıkları bellek kapasitesi alınmıştır. Karşılaştırma sonuçlarına göre geliştirilen algoritmanın başarılı sonuçlar verdiği görülmüştür.

Anahtar Kelimeler: Anahtarlama fonksiyonu, sadeleştirme, asal çarpım terim, Off-küme tabanlı minimumlaştırma, doğrudan örtme prensibi.

A NEW NEAR MINIMUM SIMPLIFICATION ALGORITHM FOR SWITCHING FUNCTIONS

ABSTRACT

The minimization of Switching functions allows designers to make use of fewer components, thus reducing the cost of particular system. Simplified as a function requires less power consumption, less volume and less cost. Most of minimization techniques work on a two–step principle, the first step identifies all of the prime implicants and the second step selects the subset of prime implicants that covers the function(s) being minimized. All procedures for Boolean networks into prime and irredundant form have $O(2^n)$ complexity. Prime Implicants identification step can be computational impractical as n increases. Therfore, in this study, subsets of prime implicants that can prove direct cover principle which based on definite criterions use for mimimization method. The method has been tested on several different kinds of problems and results of which were compared with ESPRESSO Comparison of algorithms as benchmarks; solution as a result they find that their total number of product terms, the solution times and reach a solution when they reach the memory capacity is taken. According to the results of the comparison developed algorithm gives successful results.

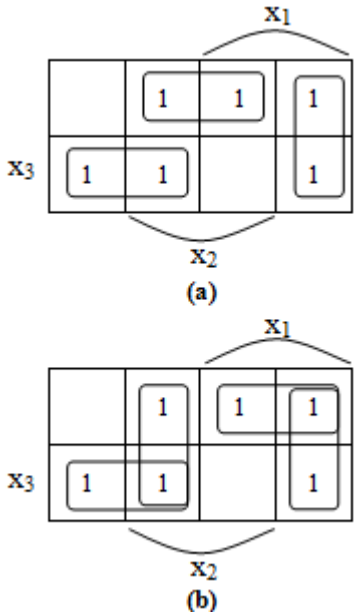
Keywords: Switching function, simplification, prime implicant, Off-set based minimization, direct-cover principle.

1. GİRİŞ (INTRODUCTION)

Günümüzde Boole cebri olarak bilinen matematiksel sistem üzerine ilk çalışmalar 1854 yılında George Boole tarafından başlatılmıştır [1]. 1904 yılında Amerikalı Matematikçi E.V. Huntington, Boole cebrine yeni aksiyomlar eklemiştir. 1938 yılında Shannon, Boole cebrini devre tasarımlarına uygulamıştır. Bunun sonucunda Anahtarlama Cebri (Switching Algebra) adı altında yeni bir bilim dalı ortaya çıkmıştır [2].

İki seviyeli lojik sadeleştirme lojik sentezin temel problemidir [3]. Geniş fonksiyon kümeleri için kesin minimum çarpımların toplamı (sum of product-SOP) ifadeleri elde edecek algoritmalar olmasına rağmen [4,5], pratik sistemlerin çoğunluğu buluşsal (heuristic-tecrübeye dayalı) lojik sadeleştirme algoritmalarını kullanır. Bu algoritmalar mutlak minimum olmayan, gerekli SOP ifadelerini üretirler. Örneğin, PRESTO [6,7], MINI [8], ESPRESSO [2] ve diğer yöntemler [9,10] minimum olmayan gerekli SOP ifadeleri üretirler. SOP sadeleştirme; PLA (Programlanabilir Lojik Dizi) optimizasyonunda, çok seviyeli lojik sentezde, durum şifrelemede, güç kestirimde, test üretmede ve diğer alanlarda kullanılır [11]. Gerekli SOP, asal çarpım terimlerin (Prime Implicant-PI) OR (veya) işlemi ile bağlanmış halidir. Öyle ki herhangi bir PI'ın silinmesi fonksiyonu değiştirir. Örneğin, Şekil 1.a ve 1.b'de gösterilen $x_1\bar{x}_2 + x_2\bar{x}_3 + \bar{x}_1 x_3$ ve $x_1\bar{x}_2 + x_1\bar{x}_3 + \bar{x}_1 x_3 + \bar{x}_1 x_2$ ifadeleri minimum SOP ve en kötü SOP ifadeleridir [1].

Boole ifadelerinin sadeleştirmesi, mantık devrelerinin ve dolaylı olarak bilgisayar programlarının daha etkili olmasına yol açmaktadır. Sadeleştirme ifadeleri



Şekil 1. Minimum SOP ve en kötü SOP ifadeleri (Minimum SOP and worst SOP)

önemlidir. Çünkü elektrik devreleri, verilen Boole ifadelerinin her bir terim veya literallerinin uygulanması için bireysel bileşenler içerir. Bu tasarımcıların daha az bileşen kullanmasını ve böylece de belirli sistemlerin maliyetlerinin düşmesini sağlamış olur. Tek çıkışlı veya çok çıkışlı Boole sadeleştirme teknikleri [12,13] anlatılmıştır. Bu tekniklerin birçoğu iki adımda çalışır. İlk adımda bütün PI belirlenir ve ikinci adımda da verilen Boole ifadesini örtecek (kapsayacak) PI'ların altkümesini seçer [1,14].

Bütün PI'ların belirlenmesi sürecinde son sonucun tam olarak belirlenmesi için ayrı durumlarda hesaplama yapılabilir. Özellikle, eğer her bir PI tam olarak k tane 0, k tane 1 ve k tane belirsiz terim (don't care) içeriyorsa, PI'nın tamamlanmış kümesinin gücü $M=(3k)!/(k!)^3$ dür [1,12,15]. Örneğin k=1,2,3,4 için sırasıyla M=6, 90, 1680 ve 34650 dır. n değişkenli bir fonksiyon için PI'ların sayısı $3^n/n$ kadar büyük olabilir [14,15]. Sonuç olarak, PI belirleme adımı değişken sayısı n arttıkça elverişsiz bir duruma gelebilir [14]. Açıkça görülmektedir ki iki seviyeli veya çok seviyeli Boole ifadelerini sadeleştirme prosedürlerinin hepsi tüm durumlarda $O(2^n)$ karmaşıklığına sahiptir [16-20]. Bu yüzden bu çalışmada, asal çarpım terimlerin belli kıstaslara cevap verecek alt kümeleri oluşturularak, doğrudan örtme prensibine dayanan yakın minimum sadeleştirme algoritması geliştirilmiştir. Böylece, PI'ların minimum kümesini bulmak için $O(2^n)$ karmaşıklığı yerine $O(n)$ karmaşıklığı metodu kullanılabilir [1,20].

Bu çalışmada mantık fonksiyonlarının ifade biçimleri, sadeleştirme yöntemleri, algoritmaları ve programları kullanılmıştır. Bu yolda elde edilmiş son teorik sonuçlara dayanarak ve minterm yöntemiyle küp cebri yöntemleri bir arada kullanılarak daha etkin olan yeni bir algoritma meydana çıkarılmıştır. Geliştirilen algoritmanın programlanması için C programlama dili kullanılmıştır.

2. TANIMLAR ve SİMGELER (DEFINITIONS and NOTATIONS)

Boole fonksiyonlarında, fonksiyonun değişken sayısına göre sahip olduğu çıkış durumları değişmektedir. n sayıda değişkene sahip olan fonksiyon 2^n sayıda mintermle ilişkide olur. Bu ilişkinin karakterine göre söz konusu mintermler aşağıdaki gibi çeşitli gruplara bölünür [21]:

1) Fonksiyonun değerinin 1'e eşit olduğu mintermler,
2) Fonksiyonun değerinin 0'a eşit olduğu mintermler,
3) Fonksiyonun değerinin belirsiz olduğu mintermler.

Yalnız 1. ve 2. grup mintermlerle ilişkili olan fonksiyonlar ***Tam Belirlenmiş Fonksiyonlar*** olarak, bunların hepsi ile ilişkili olan fonksiyonlar ise ***Tam Belirlenmemiş Fonksiyonlar*** olarak tanımlanırlar.

n girişli ve m çıkışlı bir Boole fonksiyonu aşağıdaki gibi tanımlanır [5,20]:

$$\begin{array}{ll} \text{Giriş} & : B=\{0,1\}, \\ \text{Çıkış} & : Y=\{0,1,d\}, \\ \text{Fonksiyon } f & : B^n \rightarrow Y^m. \end{array} \quad (1)$$

Burada, çıkışta gösterilen d değeri (belirsiz terim) tam belirlenmemiş değer manasındadır ve fonksiyonun istenildiği yerinde 0 veya 1 olarak kabul edilebilir. Aşağıda gösterilen giriş ve çıkış kısımlarını her bir PI içerir [1,20].

Giriş kısmı: n sabitler $\{0,1,x\}$ olabilir;
Çıkış kısmı: m sabitler $\{0,1,d\}$ olabilir.

Giriş kısmındaki x değeri bu değişken için 0 veya 1 değeri olabilir.
S_{ON}: Fonksiyonun değerini 1 yapan ON mintermlerinin kümesi,
S_{OFF}: Fonksiyonun değerini 0 yapan OFF mintermlerinin kümesi,
S_{DC}: Belirsiz mintermlerin kümesi.

Bu çalışmada sunulan algoritmada S_{ON} kümesi ve S_{OFF} kümesi kullanılmıştır. S_{DC} kümesi ise kullanılmamıştır.

3. KÜP CEBRİ İŞLEMLERİ (CUBE ALGEBRA OPERATIONS)

Küp cebri işlemleri, önce anahtarlama fonksiyonlarının (Switching Functions-SFs) en son durumunu bulmak için geliştirilmiş ve uygulanmıştır [22,23]. Yine bu işlem SF'nin ilk terimlerini (local PI) bulmak içinde kullanılmıştır. Daha sonra lojik fonksiyonların sadeleştirilmesi üzerinde kullanılmıştır [9,23].

3.1. Koordinatlı Çıkarma İşlemi (The Coordinate Subtraction Operation (# Operation))

A ve B küpleri aynı boyuta sahip iki küp olsun; $A=a_1, a_2,..., a_n$ ve $B=b_1, b_2,..., b_n$ şeklinde gösterilen iki küp arasındaki koordinatlı çıkarma işlemi iki aşamada gerçekleştirilir. Birinci aşamada aşağıdaki kurallara göre $VS=A \otimes B=v_1v_2...v_i...v_n$ şeklinde çıkarma vektörü (vector of subtraction-VS) oluşturulur [1,17,20,23].

Eğer $b_i=x$ veya $b_i=a_i$, ise $v_i=z$
Eğer $a_i=x$ ve $b_i \neq x$, ise $v_i=\overline{b_i}$ (2)
Eğer $a_i=\overline{b_i}$, ise $v_i=y$

İkinci aşamada VS'ye göre; A küpü, C sonuç küpüne dönüştürülür. C sonuç küpü bir küp veya bir kaç küpü içeren bir küme olabilir. v_i vektörünün durumuna göre aşağıdaki dönüşümler gerçekleştirilir.

- Eğer $\exists i$, $v_i=y$ ise çıkarma işlemi mümkün değildir. $C=A \# B=A$
- Eğer hiç $\exists i$, $v_i=y$ yoksa ve $v_j,..., v_k,...,v_m \in \{0,1\}$ varsa, çıkarma işleminin sonucu $\{a_1a_2...a_{j-1}v_ja_{j+1}...a_n,\ a_1a_2...a_{k-1}v_ka_{k+1}...a_n,...,a_1a_2...a_{m-1}v_ma_{m+1}...a_n\}$ kümesi olacaktır.
- Eğer $\forall i$, $v_i=z$ ise çıkarma işleminin sonucu boş kümedir. $C=A \# B=\varnothing$

3.2. Dönüşümlü Yutma İşlemi (The Commutative Absorption Operation (∇ Operation))

A ve B küpleri aynı boyuta sahip iki küp olsun; $A=a_1, a_2,..., a_n$ ve $B = b_1, b_2,..., b_n$ şeklinde gösterilen iki küp arasındaki dönüşümlü yutma işlemi iki aşamada gerçekleştirilir. Birinci aşamada aşağıdaki kurallara göre $AV=A \nabla B=v_1v_2...v_i...v_n$ şeklinde yutma vektörü (vector of absorption-AV) oluşturulur [1,17,20,23].

Eğer $a_i=b_i$, ise $v_i=z$
Eğer $a_i=x$ ve $b_i \neq x$, ise $v_i=g$ (3)
Eğer $a_i=\overline{b_i}$, ise $v_i=y$
Eğer $a_i \neq x$ ve $b_i=x$, ise $v_i=l$

İkinci aşamada AV'ye göre; A küpü, C sonuç küpüne dönüştürülür. C sonuç küpü bir küp veya bir kaç küpü içeren bir küme olabilir. v_i vektörünün durumuna göre aşağıdaki dönüşümler gerçekleştirilir.

- Eğer $\exists i$, $v_i=y$ ise yutma işlemi mümkün değildir. $C=A \nabla B=\{A,B\}$
- Eğer $\forall i$, $v_i=z$ ise $A=B$. $C=A \nabla B=A$
- Eğer $\exists i$, $v_i=g$ ve hiç $\exists i$, $v_i=l$, ise $C= A \nabla B=A$
- Eğer $\exists i$, $v_i=l$ ve hiç $\exists i$, $v_i=g$, ise $C=A \nabla B=B$
- Eğer $\exists i$, $v_i=g$ ve $\exists i$, $v_i=l$, ise yutma işlemi mümkün değildir. $C=A \nabla B=\{A,B\}$

4. YAKIN-MİNİMUM SADELEŞTİRME ALGORİTMASI (NEAR-MINIMAL SIMPLIFICATION ALGORITHM)

PI'ların belirlenmesi sürecinin karmaşıklığı S_{ON} kümesindeki her bir minterm ile S_{OFF} kümesindeki her bir mintermin ayrılmasıyla sadeleştirilebilir [23,24]. S_{ON} kümesindeki $A=\alpha_{n-1}\alpha_{n-2}\ \ldots\alpha_0$ küpünü genişletici olarak kullanarak S_{OFF} kümesindeki bütün mintermleri aşağıdaki formülü kullanarak genişletebiliriz.

Eğer $\alpha_i=\overline{\beta_i}$ ise $\gamma_i=\beta_i$ değilse $\gamma_i=x$ $i=1,2,\ldots,n$ (4)

Burada α_i, β_i ve γ_i sırasıyla küp genişletici A'nın, genişletilen B küpünün ve genişlemiş B_E küpünün $i.$ koordinat değerleridir.

Bu algoritma, 3.1 ve 3.2 de tanımlanan işlemlere ve (4) nolu kurala dayanmaktadır. Bu algoritmanın sonucu, sadeleştirilmiş Boole fonksiyonunun olası sonuçlarından bir tanesidir. Bu sonuç, bir veya bir kaç ekstra çarpım teriminin (AND kapısı) önemli olmadığı birçok pratik uygulamada yeterli olabilir. Genellikle, elde edilen sonucun, fonksiyonun en

sadeleşmiş biçimi olması S_{ON} kümesindeki mintermlerin sıralanmasına bağlıdır. Ancak, uygun sıralamayı elde etmek, düzensiz PI kümesi elde etmekten daha zordur. Herhangi bir sırada verilmiş herhangi bir Boole fonksiyon için yakın-minimum örtme algoritması aşağıdaki gibidir:

Algoritma Yakın-Minimum Sadeleştirme (S_{ON}, S_{OFF})
Begin
while ($S_{ON} \neq \varnothing$) **do**
begin
$\lambda \leftarrow$ ilk_eleman (S_{ON})
$Q_0 \leftarrow S_{OFF}$ genişlet (X)
// (4) nolu formül kullanılarak
$Q_1 \leftarrow$ dönüşümlü_yutma_işlemi (Q_0)
// (3) nolu formül
$S_{PI}(x) \leftarrow (xx\ldots xx)\ \#\ Q_1$
// (2) nolu formül kullanılarak
if {*EPI*_seçme_işlemi ($S_{PI}(x)$)} > 1 **then**
$EPI(x) \leftarrow$ {*EPI*_seçme_işlemi ($S_{PI}(x)$)}nin herhangi bir elemanı
else
$EPI(x) \leftarrow$ *EPI*_seçme_işlemi ($S_{PI}(x)$)
$S_{ONi} \leftarrow S_{ON}\ \#\ EPI(x)$
$S_{EPI} \leftarrow S_{EPI} \cup EPI(x)$
end
End

Örnek: $f(x_3,x_2,x_1,x_0)=\Sigma(0,1,2,3,5,7,10,11,13,14,15)$ Boole fonksiyonunu Yakın-Minimum Sadeleştirme Algoritmasını kullanarak minimumlaştırın.
S_{ON1} = {0000, 0001, 0010, 0011, 0101, 0111, 1010, 1011, 1101, 1110, 1111},
S_{OFF} = {0100, 0110, 1000, 1001, 1100},
S_{ON} kümesinin ilk mintermi 0000, λ1=0000,

Tablo 1'den görüldüğü gibi, $Q1.1 = \{x1xx,1xxx\}$. Tam küpten (*xxxx*), $Q1.1$ değerleri koordinatlı çıkarma işlemi ile çıkarılarak seçilen 0000 minterminin örttüğü küp kümesi belirlenir. Bu işlem aşağıda 1.2'de gösterilmiştir.
1.2. 0000 minterminin örttüğü küp kümesinin belirlenmesi.
$S_{PI}(1) = xxxx\ \#\ Q1 = \{((xxxx\ \#\ x1xx)\ \#\ 1xxx)\}$
$= \{x0xx\ \#\ 1xxx\} = \{00xx\}$
$S_{PI}(1) = \{S_{PI}\} = \{00xx\}$.

Tablo 1. 0000 mintermi için $Q0.1$ ve $Q1.1$ kümelerinin belirlenmesi (Determination of $Q0.1$ and $Q1.1$ sets for 0000 minterm)

S_{OFF}	$Q0.1$	Küp Durumu	$Q1.1$
0100	*x1xx*	Asal	*x1xx*
0110	*x11x*	*x1xx* tarafından kapsandı	
1000	*1xxx*	Asal	*1xxx*
1001	*1xx1*	*1xxx* tarafından kapsandı	
1100	*11xx*	*x1xx* tarafından kapsandı	

0000 minterminin örttüğü küp kümesi belirlendikten sonra elde edilen sonuçta tek bir küp olduğu için bu küp esas asal çarpım terimler kümesine dahil edilir. Bu tek küp S_{ON} kümesinden çıkarılarak yeni S_{ON} kümesi elde edilir.
1.3. En büyük küpün belirlenmesi.
$P1.1 = S_{ON1}\ \#\ S_{PI}(1) = S_{ON1}\ \#\ 00xx$
$= \{0000, 0001, 0010, 0011, 0101, 0111, 1010, 1011, 1101, 1110, 1111\}\ \#\ 00xx,$
$= \{0101,0111,1010,1011,1101,1110,1111\}.$
Yukarıdaki işlemlerde görüldüğü gibi $S_{PI}(1)$ kümesi tek elemanlıdır. Böylece, $EPI(1)=00xx$, $S_{EPI} = \{00xx\}$.

S_{ON} kümesinden *00xx* küpü çıkarılarak yeni S_{ON} kümesi belirlenir.
2.1. S_{ON} kümesinin örtülmeyen kısmının belirlenmesi.
$S_{ON2}=P1.1 = \{0101,0111,1010,1011,1101,1110,1111\}$,
Buradan, yeni S_{ON2} kümesinin ilk mintermi 0101,
$\lambda 2 = 0101$,

Tablo 2'den görüldüğü gibi, $Q1.2 = \{xxx0,10xx\}$. Tam küpten (*xxxx*), $Q1.2$ değerleri koordinatlı çıkarma işlemi ile çıkarılarak seçilen 0101 minterminin örttüğü küp kümesi belirlenir. Bu işlem aşağıda 2.2'de gösterilmiştir.
2.2. 0101 minterminin örttüğü küp kümesinin belirlenmesi.
$S_{PI}(2) = xxxx\ \#\ Q1 = \{(xxxx\ \#\ xxx0)\ \#\ 10xx\}$
$= \{xxx1\ \#\ 10xx\} = \{0xx1,x1x1\}$,
$S_{PI}(2) = \{\ S_{PI}(2.1), S_{PI}(2.2)\} = \{0xx1,x1x1\}$.

Tablo 2. 0101 mintermi için $Q0.2$ ve $Q1.2$ kümelerinin belirlenmesi (Determination of $Q0.2$ and $Q1.2$ sets for 0101 minterm)

S_{OFF}	$Q0.2$	Küp Durumu	$Q1.2$
0100	*xxx0*	Asal	*xxx0*
0110	*xx10*	*xxx0* tarafından kapsandı	
1000	*10x0*	*xxx0* tarafından kapsandı	
1001	*10xx*	Asal	*10xx*
1100	*1xx0*	*xxx0* tarafından kapsandı	

0101 minterminin örttüğü küp kümesi belirlendikten sonra elde edilen sonuçta iki küp vardır. Bu küplerden hangisinin esas asal çarpım terimler kümesine dahil edileceği aşağıda gösterilmiştir.
2.3. En büyük küpün belirlenmesi.
$P2.1 = S_{ON2}\ \#\ S_{PI}(2.1) = S_{ON2}\ \#\ 0xx1$
$= \{0101,0111,1010,1011,1101,1110, 1111\}\ \#\ 0xx1 = \{1010,1011,1101,1110,1111\},$
$P2.2 = S_{ON2}\ \#\ S_{PI}(2.2) = S_{ON2}\ \#\ x1x1$
$= \{0101,0111,1010,1011,1101,1110, 1111\}\ \#\ x1x1 = \{1010,1011,1110\},$
Yukarıdaki işlemlerde görüldüğü gibi *P2.2* kümesi (3 eleman) *P2.1* kümesinden (5 eleman) daha güçlüdür. Sonuç olarak *x1x1* küpü *0xx1* küpünden daha büyüktür. Böylece,
$EPI(2) = x1x1$, $S_{EPI} = \{00xx,x1x1\}$.

S_{ON} kümesinden *x1x1* küpü çıkarılarak yeni S_{ON} kümesi belirlenir.
3.1. S_{ON} kümesinin örtülmeyen kısmının belirlenmesi.

$S_{ON3} = P2.2 = \{1010,1011,1110\}$, Buradan, yeni S_{ON3} kümesinin ilk mintermi 1010, $\lambda 3 = 1010$,

Tablo 3'den görüldüğü gibi, $Q1.3 = \{xx0x,01xx\}$. Tam küpten (*xxxx*), $Q1.3$ değerleri koordinatlı çıkarma işlemi ile çıkarılarak seçilen 1010 minterminin örttüğü küp kümesi belirlenir. Bu işlem aşağıda 3.2'de gösterilmiştir.

3.2. 1010 minterminin örttüğü küp kümesinin belirlenmesi.

$S_{PI}(3) = xxxx \# Q1 = \{(xxxx \# xx0x) \# 01xx\}$
$= \{xx1x \# 01xx\} = \{1x1x,x01x\}$
$S_{PI}(3) = \{ S_{PI}(3.1), S_{PI}(3.2)\} = \{1x1x,x01x \}$.

Tablo 3. 1010 mintermi için $Q0.3$ ve $Q1.3$ kümelerinin belirlenmesi (Determination of $Q0.3$ and $Q1.3$ sets for 1010 minterm)

S_{OFF}	$Q0.3$	Küp Durumu	$Q1.3$
0100	*010x*	*01xx* tarafından kapsandı	
0110	*01xx*	Asal	*01xx*
1000	*xx0x*	Asal	*xx0x*
1001	*xx01*	*xx0x* tarafından kapsandı	
1100	*x10x*	*xx0x* tarafından kapsandı	

1010 minterminin örttüğü küp kümesi belirlendikten sonra elde edilen sonuçta iki küp vardır. Bu küplerden hangisinin esas asal çarpım terimler kümesine dahil edileceği aşağıda gösterilmiştir.

3.3. En büyük küpün belirlenmesi.

$P3.1 = S_{ON3} \# 10x0 = \{1010,1011,1110\} \# 1x1x = \varnothing$
$P3.2 = S_{ON3} \# x000$
$= \{1010,1011,1110\} \# x01x = \{1110\}$

Yukarıdaki işlemlerde görüldüğü gibi *P3.1* kümesi (0 eleman) *P3.2* kümesinden (1 eleman) daha güçlüdür. Sonuç olarak *1x1x* küpü *x01x* küpünden daha büyüktür. Böylece,

$EPI(3) = 1x1x$, $S_{EPI} = \{00xx,x1x1,1x1x\}$.

Sonuç olarak, sadeleştirme işlemi tamamlanmıştır. Çünkü S_{ON} kümesindeki bütün mintermler örtülmüştür.

4.1. S_{ON} kümesinin örtülmeyen kısmının belirlenmesi.

$S_{ON4} = P3.1 = \varnothing$.

Fonksiyonun sadeleştirilmiş durumu;

$S_{EPI} = \{00xx,x1x1,1x1x\}$

$$f = \bar{x}_3\,\bar{x}_2 + x_2\,x_0 + x_3\,x_1$$

5. ALGORİTMALARIN KARŞILAŞTIRILMASI (COMPARISON OF ALGORITHMS)

Geliştirilmiş olan Yakın Minimum Sadeleştirme Algoritması (YMSA), ESPRESSO algoritması ile karşılaştırılmıştır. Karşılaştırma kriteri olarak üç ana durum belirlenmiştir. Bunlar;

- Algoritmaların çözüm sonucunda buldukları SOP sayısı,
- Algoritmaların çözüme ulaşma süreleri,
- Algoritmaların çözüme ulaşırken kullandıkları bellek kapasitesi

YMSA'sı ve ESPRESSO, C programlama dilinde kodlanmışlardır. Algoritmalar aynı dosya formatını kullanmıştır. Algoritmaları aynı şartlarda karşılaştırabilmek için ESPRESSO algoritmasının belirlediği durumlar dikkate alınmıştır. Karşılaştırmaların gerçekleştirilmesini kolaylaştırmak için Delphi programlama dilinde ara yüz programı yazılmıştır. Tablo 4 ve 5'de değerlendirme testlerine (benchmarklara) ait, giriş değişken sayısı, S_{ON} sayısı, S_{OFF} sayısı, SOP sayısı, algoritmaların sadeleştirme zamanları ve kullandıkları bellek kapasiteleri verilmiştir.

Performans ve sonuç kalitesini karşılaştırmak için değerlendirme testleri, YMSA ve ESPRESSO tarafından sadeleştirilmiştir. Karşılaştırmalar AMD Sempron 2600 işlemcili 1.83 GHz ve 512 MB RAM belleği olan standart bir PC'de gerçekleştirilmiştir. Karşılaştırma için kırk sekiz farklı tek-çıkışlı fonksiyon kullanılmıştır. Çarpım terimlerinin toplamı ifadesi şeklinde verilen sonuçlar (SOP sayısı) açısından algoritmalar karşılaştırıldığında elde edilen bilgiler şöyledir. YMSA ile ESPRESSO algoritması karşılaştırıldığında; fonksiyonların %75'inde eşit sayıda SOP sayısına sahip oldukları görülmüştür. Bu algoritmalardan ESPRESSO, fonksiyonların %18,75'inde daha iyi sonuç bulurken YMSA %6,25'inde daha iyi sonuç bulmuştur. SOP sayılarının ortalama değerlerine göre YMSA ile ESPRESSO'yu karşılaştırdığımızda ESPRESSO'nun daha iyi sonuç bulduğu fonksiyonlarda ortalama %9,7 daha az SOP bulmuştur. YMSA'nın daha iyi sonuç bulduğu fonksiyonlarda ortalama %45 daha az SOP bulmuştur.

YMSA ile ESPRESSO karşılaştırıldığında YMSA'sının sadeleştirme işlemlerini çok daha hızlı gerçekleştirdiği görülmektedir. Fonksiyonların %89,6'sında YMSA daha hızlı bir şekilde sadeleştirme yapıp sonuca ulaşmıştır. %10,4'ünde ise algoritmaların sonuca ulaşma zamanları eşittir. Bu iki algoritma açısından bakıldığında YMSA'sı ESPRESSO algoritmasına göre çok daha hızlıdır. Ortalama olarak YMSA ESPRESSO'ya göre 7,9 kat daha hızlı sadeleştirme yapmaktadır. Bu algoritmalar için sonuca ulaşma süresine bakıldığı zaman, genellikle, YMSA'nın göreceli hızı $|S_{OFF}|/|S_{ON}|$ değerinin azaldığı durumlarda artmaktadır. Örneğin; 9 değişkene sahip olan *prom1* değerlendirme testinin $|S_{OFF}|/|S_{ON}|$=14/488 = 0.03 olduğu durumda YMSA, ESPRESSO'dan 115 kat daha hızlı sadeleştirme yapmaktadır. Bunun yanında, 10 değişkene sahip olan *t12* değerlendirme testinin $|S_{OFF}|/|S_{ON}|$= 758/266=2,85 olduğu durumda YMSA, ESPRESSO'dan 4 kat daha hızlı sadeleştirme yapmıştır.

Tablo 4. SOP Sayısı ve Çalışma Zamanları (SOP number and execution time)

Değerlendirme Testleri	Değişken Sayısı	S_{ON} Sayısı	S_{OFF} Sayısı	SOP Sayısı			Çalışma Zamanı (mili sn.)		
				YMSA (S_Y)	ESPRESSO (S_E)	$\frac{S_E}{S_Y}$	YMSA (T_Y)	ESPRESSO (T_E)	$\frac{T_E}{T_Y}$
check	04	4	9	1	1	1	15,625	15,625	1
check1	04	4	8	1	1	1	15,625	15,625	1
check2	04	4	6	1	1	1	15,625	15,625	1
check3	04	8	5	2	2	1	15,625	15,625	1
wim	04	9	1	4	4	1	15,625	31,25	2
p82	05	11	13	4	4	1	15,625	15,625	1
squar	05	9	23	2	2	1	15,625	31,25	2
m	06	17	8	4	4	1	15,625	40,625	2,6
m5	06	27	5	4	4	1	15,625	40,625	2,6
new2	06	3	4	2	2	1	15,625	40,625	2,6
poperom	06	56	8	7	7	1	15,625	56,25	3,6
sqr	06	18	46	2	2	1	31,25	56,25	1,8
inc	07	12	22	6	6	1	15,625	56,25	3,6
linrom	07	65	63	24	24	1	15,625	40,625	2,6
max128	07	29	99	8	8	1	31,25	56,25	1,8
max3	07	12	116	7	7	1	15,625	56,25	3,6
sqn	07	48	48	8	8	1	31,25	40,625	1,3
z5xp1	07	25	103	3	3	1	15,625	40,625	2,6
dist	08	53	203	12	12	1	15,625	65,625	4,2
e	08	65	128	21	20	0,95	15,625	65,625	4,2
ex5	08	33	223	2	2	1	15,625	81,25	5,2
exp	08	18	52	3	4	1,33	31,25	65,625	2,1
exps	08	65	131	21	20	0,95	31,25	81,25	2,6
exp1	08	24	42	3	6	2	15,625	65,625	4,2
f51m	08	128	128	23	23	1	15,625	81,25	5,2
m3	08	98	30	16	14	0,88	31,25	116,25	3,72
m4	08	223	26	26	23	0,88	31,25	337,5	10,8
mlp4	08	32	224	9	9	1	31,25	81,25	2,6
root	08	15	241	4	4	1	31,25	65,625	2,1
rd84	08	120	136	84	84	1	46,875	186,25	3,97
apex4	09	4	434	4	4	1	31,25	100,625	3,22
max4	09	38	8	6	6	1	31,25	287,5	9,2
min	09	87	46	6	6	1	15,625	81,25	5,2
max512	09	267	245	10	10	1	15,625	100,625	6,44
prom1	09	488	14	23	19	0,83	31,25	3586,87	114,8
prom2	09	142	145	7	7	1	15,625	131,875	8,44
max1024	10	516	508	4	4	1	15,625	116,25	7,44
t10	10	134	189	48	49	1,02	46,875	715,625	15,27
t11	10	266	371	83	76	0,92	46,875	650	13,87
t12	10	266	758	109	99	0,91	62,5	256,25	4,1
br11	12	29	5	3	3	1	15,625	456,25	29,2
br1	12	26	8	5	4	0,8	15,625	156,25	10
br2	12	29	6	2	2	1	15,625	140,625	9
t3	12	27	121	6	6	1	15,625	140,625	9
pdc	16	29	1886	4	4	1	31,25	184,375	5,9
spla	16	67	2036	31	29	0,93	46,875	284,375	6,07
den	18	24	3	4	4	1	31,25	537,5	17,2
bca	26	15	13	1	1	1	31,25	468,75	15

Tablo 5. Bellek Kullanımları (Memory usage)

Değerlendirme Testleri	Değişken sayısı	S_{ON} sayısı	S_{OFF} sayısı	Bellek Kullanımı (kilobyte)		
				YMSA (M_Y)	ESPRESSO (M_E)	$\frac{M_E}{M_Y}$
check	04	4	9	2.064	2.482	1,20
check1	04	4	8	1.995	2.441	1,22
check2	04	4	6	2.318	2.449	1,06
check3	04	8	5	1.802	2.469	1,37
wim	04	9	1	1.946	2.478	1,27
p82	05	11	13	2.314	2.428	1,05
squar	05	9	23	2.290	2.437	1,06
m	06	27	5	1.954	2.449	1,25
m5	06	27	5	2.269	1.982	0,87
new2	06	3	4	2.310	2.670	1,16
poperom	06	56	8	2.249	2.666	1,19
sqr	06	18	46	2.281	2.461	1,08
inc	07	12	22	1.937	2.428	1,25
linrom	07	65	63	2.277	2.457	1,08
max128	07	29	99	2.285	2.154	0,94
max3	07	12	116	2.310	2.056	0,89
sqn	07	48	48	2.310	2.256	0,98
z5xp1	07	25	103	2.281	2.416	1,06
dist	08	53	203	1.945	2.404	1,24
e	08	65	128	1.908	2.379	1,25
ex5	08	33	223	2.298	2.408	1,05
exp	08	18	52	2.298	2.379	1,04
exps	08	65	131	2.298	2.007	0,87
exp1	08	24	42	1.966	2.416	1,23
f51m	08	128	128	2.318	2.437	1,05
m3	08	98	30	2.285	2.387	1,04
m4	08	223	26	2.248	2.396	1,07
mlp4	08	32	224	1.941	2.469	1,27
root	08	15	241	2.040	2.404	1,18
rd84	08	120	136	2.019	2.142	1,06
apex4	09	4	434	2.298	2.379	1,04
max4	09	38	8	2.306	2.379	1,03
min	09	87	46	2.302	2.379	1,03
max512	09	267	245	2.278	2.347	1,03
prom1	09	488	14	1.872	2.887	1,54
prom2	09	142	145	2.302	2.289	0,99
max1024	10	516	508	2.232	2.301	1,03
t10	10	113	201	2.318	2.818	1,22
t11	10	266	371	2.244	2.408	1,07
t12	10	266	758	2.318	2.441	1,05
br11	12	29	5	1.937	2.338	1,21
br1	12	26	8	2.281	2.375	1,04
br2	12	29	6	2.244	1.957	0,87
t3	12	27	121	2.286	2.379	1,04
pdc	16	29	1886	2.338	2.289	0,98
spla	16	67	2036	2.220	2.428	1,09
den	18	24	3	2.273	2.396	1,05
bca	26	15	13	2.281	2.363	1,04

Algoritmaların sadeleştirme yaparken kullandıkları bellek alanı bakımından değerlendirilmesi yapıldığında, ESPRESSO'nun YMSA'na göre %16,7 fonksiyonda daha iyi olduğu görülmesine rağmen %83,3 fonksiyonda YMSA daha az bellek alanı kullanmıştır. Algoritmaların daha az bellek alanı kullandıkları fonksiyonlardaki durumlarına bakıldığında ise YMSA %13,2 daha az bellek alanı kullanırken ESPRESSO %9 daha az bellek alanı kullanmıştır.

6. SONUÇ (CONCLUSION)

Bu çalışmada anahtarlama fonksiyonlarını sadeleştirmek için yeni bir algoritma sunulmuştur. Sunulan algoritmada küp cebri işlemleri kullanılmıştır. Sunulan algoritmada küp cebrinin koordinatlı çıkarma ve dönüşümlü yutma işlemleri kullanılmıştır. Sunulan algoritmada verilen fonksiyonun ON kümesi mintermlerinden bir tanesini rasgele seçilmekte ve bu mintermi kapsayan asal çarpım terimler oluşturulmaktadır. YMSA "Büyük veya Daha Az" işlemi kullanılarak esas asal çarpım terimler (EAI) bir bir seçilmektedir. Belirlenen asal çarpım terim için eşit sayıda minterm örtülürse üretilmiş PI'lardan bir tanesi seçilmektedir. Bu işlemlerin yapılması ile fonksiyonun sadeleşmiş halini temsil edecek esas asal çarpım terimler belirlenmiş olur. Sunulan YMSA önemli bir şekilde var olan metotlardan hızlı çalışmaktadır ve daha az bellek kapasitesine ihtiyaç duymaktadır. Çünkü minimum sayıda geçici sonuçlar üreterek işleme tabi tutmaktadır. Bu özellikler sunulan algoritmayı özlü ve son derece verimli yapmaktadır.

Sunulan YMSA'sı C programlama dilinde kodlanmıştır. Karşılaştırması yapılan ESPRESSO programı da C programlama dilinde kodlanmıştır. Sunulan algoritmada ve karşılaştırması yapılan ESPRESSO programında aynı dosya yapısı kullanılmıştır. Programların kullanımını kolaylaştırmak için Delphi programlama dilinde ara yüz programı yazılmıştır. Karşılaştırmalarda tek çıkışlı fonksiyonlar kullanılmıştır. Karşılaştırması yapılan fonksiyonlar tam tanımlanmamış veya tam tanımlanmış fonksiyonlardır. Algoritmaların karşılaştırması üç duruma göre yapılmıştır. Bunlar, algoritmaların verilen fonksiyonları sadeleştirdikten sonra elde ettikleri çarpım terimlerinin toplamı (SOP) sayısına göre, algoritmaların sadeleştirme zamanları ve bellek kullanma durumlarıdır.

SOP sayısına göre yapılan karşılaştırma sonucunda, fonksiyonların %75'inde eşit sayıda, %18,75'inde ESPRESSO daha iyi sonuç bulurken YMSA %6,25'inde daha iyi sonuç bulmuştur. Algoritmaların sadeleştirme zamanlarına göre yapılan karşılaştırma sonucunda, fonksiyonların %89,6'sında YMSA daha hızlı bir şekilde sonuca ulaşırken, %10,4'ünde ise algoritmaların sonuca ulaşma zamanları eşittir. Algoritmaların sonuca ulaşırken kullandıkları bellek alanına göre yapılan karşılaştırmada, ESPRESSO %16,7 fonksiyonda, YMSA %83,3 fonksiyonda daha az bellek alanı kullanmıştır.

TEŞEKKÜR (ACKNOWLEDGEMENT)

Bu çalışma, Selçuk Üniversitesi Bilimsel Araştırma Projeleri Koordinatörlüğü'nün 2002/034 nolu projenin bir parçası olup vermiş olduğu katkılarından dolayı teşekkür ederiz.

KAYNAKLAR (REFERENCES)

1. Başçiftçi F., **Anahtarlama Fonksiyonları İçin Yerel Basitleştirme Algoritmaları,** Doktora Tezi, Selçuk Üniversitesi, Fen Bilimleri Enstitüsü, 2006.
2. Brayton R.K., Hachtel G.D., McMullen C., Sangiovanni-Vincentelli A.L., **Logic Minimization Algorithms For VLSI Synthesis,** ISBN 0-89838-164-9, Hardbound, Kluwer Academic Publishers, 1984.
3. Sasao T., Butler J.T., Worst and Best Irredundant Sum-of–Product Expressions, **IEEE Transactions on Computers**, Vol. 50(9), pp. 935-947, 2001.
4. Coudert O., Two-Level Logic Minimization: an Overview. Integration, **The VLSI Journal**, 17-2, pp. 97-140, October 1994.
5. Dagenais M.R., Agarwal V.K., Rumin N.C., McBOOLE: A New Procedure for Exact Logic Minimization, **IEEE Transactions On Computer Aided Design**, Vol. CAD-5, No:1, Jaunary 1986.
6. Brown, D.W., A State-Machine Synthesizer – SMS. Proc. **18th Design Automation Conference**, pp.301-304, Nashville, June 1981.
7. Svoboda A., White D.E, **Advanced Logical Circuit Design Techniques**, New York: Garland Pres, 1979.
8. Hong S.J., Cain R.G. and Ostapko D.L., MINI: A Heuristic Approach For Logic Minimization, **IBM J. of Res. and Dev.**, Vol.18, pp.443-458, September 1974.
9. Dietmeyer D.L., **Logic Design of Digital Systems,** MA:Ally and Bacon, Boston 1979.
10. Nguyen K., Perkowski M., Goldstein N., Palmini-Fats Boolean Minimizer for Personal Computers, **Proc. Design Automation Conf.**, pp. 615-621, Aug, 1987.
11. Mishchenco A., Sasao T., Large-Scale SOP minimization Using Decomposition and Functional Properties, **DAC**, pp149-154, June 2-6 2003.
12. Miller R.E., **Switching Theory**, Vol. 1 Combination Circuits, New York; John Wiley and sons, 1965.
13. Mano M. M., **Digital Design**, Prentice-Hall International Editions, 1984.
14. Perkins S.R., Rhyne T., An Algorithm for Identifying and Selecting The Prime Implicants

of a Multiple-Output Boolean Function, **IEEE Transactions On Computer Aided Design**, Vol. 7, No:11, November 1988.
15. Gurunath B., Biswas N.N., An Algorithm for Multiple Output Minimization, **IEEE Transactions On Computer Aided Design**, Vol. 8, No:9, September 1989.
16. Bartlett K.A., Brayton R.K., Hachtel G.D., Jacoby R.M., Morrison C.R., Rudell R.L., Sangiovanni-Vincentelli A., Wang A.R., Multilevel Logic Minimization Using Implicit Don't Cares, **IEEE Transactions On Computer Aided Design**, Vol. 7, No:6, June 1988.
17. Allahverdi N.M., Kahramanlı Ş.Ş., Erciyeş K., A Fault Tolerant Routing Algorithm Based On Cube Algebra For Hypercube Systems, **Journal of Systems Architecture**, 46, pages 201-205, 2000.
18. Bernasconi A., Ciriani V., Luccio F., Pagli L., Fast Three-Level Logic Minimization Based on Autoymmetry, **http://citeseer.nj.nec.com/cachedpage/462188/1** 2001.
19. Beckert B., Iiahnle R., Escalada-Imaz G., Simp. of Many-Valued Logic Formulas Using Anti-Links, **http://citeseer.nj.nec.com/cachedpage /221052/1**, 1997.
20. Kahramanlı Ş., Başçiftçi F., Boolean Functions Simplification Algorithm Of O(n) Complexity, **Mathematical & Computational Applications**, Volume 8 Numbers 1-3, pages:271-278. ISSN:1300-686X, 2003.
21. Kahramanlı Ş., Özcan M., **Lojik Tasarımın Temelleri ve Uygulamaları**, Atlas Yayın Dağıtım, İstanbul 2002.
22. Roth J.P., Algebric Topological Methods for the Synthesis of Switching Systems in n-variables, **The Instute for Advanced Study**, Princeton, New Jersey, 1956.
23. Nadjafov E.M., Kahramanov S.S., On the Synthesis of Multiple Output Switching Scheme, **Scientific Notes of Azerbaijan Institute of Petrolium and Chemistry**, Baku, Azaerbaijan, Vol. IX, No 3 65-69, 1973.
24. Kahramanlı S.S., Allahverdi N.M., Compact Method of Minimization of Boolean Functions with Multiple Variables. **Proc. Inter. Symp. Application of Computers**, Selçuk University, Konya, Turkey, 433-440, 1993.