İSTANBUL KEMERBURGAZ ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

SRAM TABANLI FPGA DEVRELERİNDE LUT SEVİYESİNDE FONKSİYONLAR ARASINDAKİ NPN İLİŞKİLERİN MAKSİMİZE EDİLMESİ

YÜKSEK LİSANS TEZİ

UĞUR CORUH

İSTANBUL,2014

SRAM TABANLI FPGA DEVRELERİNDE LUT SEVİYESİNDE FONKSİYONLAR ARASINDAKİ NPN İLİŞKİLERİN MAKSİMİZE EDİLMESİ

Uğur CORUH

Bilgisayar Mühendisliği Lisans, Selçuk Üniversitesi, 2009

İstanbul Kemerburgaz Üniversitesi, Fen Bilimleri Enstitüsü

Yüksek Lisans, Elektrik ve Bilgisayar Mühendisliği Bölümü'ne

Sunulmuştur.

Bu çalışma tarafımızca incelenmiş olup, kapsam ve kalite açısından Yüksek Lisans tezi olmaya yeterli bulunmuştur.

Doç. Dr. Fatih Koçan

Eş Danışman

Yrd. Doç. Dr. Tuğçe Ballı Altuğlu

Danışman

İnceleme Komitesi Üyeleri (İlk isim jüri başkanına, ikinci isim tez danışmanına aittir.)

Doç. Dr. Oğuz Bayat	(Jüri)	
Yrd. Doç. Dr. Tuğçe Ballı Altuğlu	(Jüri)	
Doç. Dr. Fatih Koçan	(Jüri)	
Yrd. Doç. Dr. Mehmet Dinçer Erbaş	(Jüri)	
Yrd. Doç. Dr. Akıner Tüzüner	(Jüri)	

Bu çalışma bir Yüksek Lisans tezinin tüm gerekli şartlarını taşımaktadır.

Prof. Dr. Veysel Gazi

Bölüm Başkanı

[Üniversite] onayı 28 / 02 / 2014

Doç. Dr. Oğuz Bayat

Enstitü Müdürü

Bu dökümandaki tüm bilgilerin akademik kural ve etiğe bağlı kalınarak yazıldığını ve tez yazım kuralları kapsamında bu çalışmada bulunan ve orijinal olmayan bütün bilgi ve materyallerin referanslandırıldığını temin ederim.

Uğur CORUH

İTHAF

Bu yüksek lisans tezimi, benim bu günlere gelmem için hiçbir fedakârlıktan kaçınmayan ve her zaman yanımda olan sevgili aileme ve sabırla bana destek olan canım eşime ithaf ediyorum.

TEŞEKKÜR

Hayatımdaki önemli bir adımı daha tamamlamış olmanın mutluluğu içerisindeyim. Bu mutluluğu bana yaşatan, bilgisayar mühendisi olmama ve bu yüksek lisansa başlamama vesile olan her zaman kendisinden gurur duyduğum saygıdeğer hocam Doç. Dr. Fatih Koçan'a, yüksek lisans süresince her türlü yardımı ve hoşgörüyü benden esirgemeyerek bu süreci tamamlama yardım eden saygıdeğer ve çok kıymetli hocalarım Doç. Dr. Oğuz Bayat ve Yrd. Doç. Dr. Tuğçe Ballı Altuğlu'ya, Bilgisayar ve Elektrik bölümünde bana yardımcı olan hocalarıma , ekiplerindeki herkese ve takım halinde çalıştığımız sınıf arkadaşlarıma, bugünlere gelmemde büyük pay sahibi olan aileme ve dostlarıma, bu süreçte bana sabırlı bir şekilde destek olan canım eşime,

Teşekkürlerimi sunarım.

Uğur CORUH

ÖZET

SRAM TABANLI FPGA DEVRELERİNDE LUT SEVİYESİNDE FONKSİYONLAR ARASINDAKİ NPN İLİŞKİLERİN MAKSİMİZE EDİLMESİ

Uğur Coruh

Fen Bilimleri Enstitüsü, İstanbul Kemerburgaz Üniversitesi

Danışman: Yrd. Doç. Dr. Tuğçe Ballı Altuğlu

Eş Danışman: Doç. Dr. Fatih Koçan

Mayıs 2014

Bu tezde, yeni bir FPGA (Field Programmable Gate Array) sentez algoritması geliştirmek için araştırma yapılmıştır. Yeni algoritma LUT (Lookup Table) seviyesindeki devrelerde NPN (Input Negation, Input Permutation and Output Negation) denk fonksiyon sayısını artırmayı amaçlamaktadır. Hedefimiz, NPN denk fonksiyonlar arasında SRAM (Static Random Access Memory) paylaşımı sağlayan, yeni ve verimli bir FPGA mimarisi geliştirmektir. Bu tez çalışmasında, hızlı bir şekilde NPN denklik kontrolü yapmak amacıyla, FPGA sentez aracı olan ABC (A System for Sequential Synthesis and Verification) aracı için eklenti olarak çalışan bir araç geliştirilmiştir. Geliştirmiş olduğumuz aracın çalışma prensibi şöyledir: Öncelikle, iki adet mantıksal fonksiyonu hiper çizge olarak formülize eder. Daha sonra, çizge izomorfizm kontrolü yapan Nauty izomorfizm karşılaştırma aracını çağırıp bu iki çizgenin denkliklerini kontrol eder. Kullanıcılar, referans fonksiyon kümesini ABC aracına girdi olarak verirler. ABC aracı

geliştirdiğimiz eklentiyi kullanarak kullanıcılardan aldığı fonksiyon kümesindeki fonksiyonlara NPN denk, LUT fonksiyonları oluşturur. Kullanıcının girdi olarak verdiği fonksiyonlar şöyle seçilir: İlk önce, devre orijinal ABC aracı ile sentezlenir. Normal ABC aracı sentezleme sonucu oluşan devreler analiz edilip en çok tekrar eden fonksiyon sınıfları belirlenir. Sonrasında bu sınıflardan seçilen temsilci fonksiyonlar geliştirmiş olduğumuz eklenti ile ABC aracında sentezleme için seçilir. Bir defaya mahsus da geliştirdiğimiz eklenti aktifleştirilip ABC aracı kullanılarak sentezleme yapılır ve oluşturulan LUT'ların girdi olarak verilen fonksiyonlara NPN denk olması sağlanır. Geliştirmiş olduğumuz ABC aracı eklentisi, olabilecek en iyi şekilde kullanıcıların girdi olarak verdiği fonksiyonlara NPN denk LUT'lar oluşturmaya çalışır. Önerilen araştırmanın performansı, MCNC (Microelectronics Center of North Carolina) referans devreleri ile yapılan bir dizi FPGA sentezleme ile ölçülmüştür. Bu tez çalışmasında, beş ve altı girişli fonksiyonlar araştırılmıştır. Beş ve altı girişli fonksiyonların seçilmesinin nedeni pratik olarak kullanılabilir olmalarıdır.

Anahtar Kelimeler: ABC aracı, FPGA, Lojik Sentezleme, NPN denklik, SRAM Paylaşımı

ABSTRACT

MAXIMIZING NPN EQUIVALENT FUNCTIONS IN A LUT-LEVEL CIRCUIT FOR SRAM-BASED FPGAS

Uğur Coruh

Graduate School of Science and Engineering, Istanbul Kemerburgaz University

Supervisor: Asst. Prof. Dr. Tuğçe Ballı Altuğlu

Co-Supervisor: Assoc. Prof. Dr. Fatih Koçan

May 2014

In this thesis, we investigate a novel FPGA (Field Programmable Gate Array) synthesis algorithm. The new algorithm aims at maximizing the number of NPN (Input Negation, Input Permutation and Output Negation) equivalent functions in a LUT (Lookup Table) level circuit. The goal is to efficiently utilize newly proposed FPGA architectures that enable SRAM (Static Random Access Memory) sharing among NPN-equivalent functions. Our algorithm utilizes a fast NPN-equivalence checker tool and incorporates it into an FPGA synthesis tool, called the ABC (A System for Sequential Synthesis and Verification) tool. The NPN-equivalence checker tool formulates two logic functions as hyper-graphs and calls a hyper-graph isomorphism checker tool, called Nauty graph isomorphism tool, to check the equivalency of two hyper-graphs. The user provides a set of functions as input to the extension of ABC tool that we developed and forces the

ABC tool to generate LUT functions that are NPN-equivalent to the user inputted functions. The user specified functions are identified as follows. First, the circuit is synthesized with the original, unmodified ABC tool. The resulting circuit is analyzed and the most frequently occurring NPN equivalence classes are identified. Later, a representative function from each of these classes is selected as the input to the second synthesis process. After that, we synthesize the circuit one more time with the modified ABC tool to force the LUTs to be equivalent to the identified functions. The modified ABC tool does its best to generate LUTs that are NPN-equivalent to the user specified functions. The performance of the proposed algorithm is evaluated by performing the synthesis on a set of benchmark circuits, namely MCNC (Microelectronics Center of North Carolina) synthesis benchmarks. In our experiments, we have investigated five and six input functions, since these input function sizes were applicable in practice.

Keywords: ABC, FPGA, Logic Synthesis, NPN, SRAM Sharing

İÇİNDEKİLER

TABLO LİSTESİ

ŞEKİL LİSTESİ

KISALTMALAR LİSTESİ

ABC	A System for Sequential Synthesis and Verification
ASIC	Application Specific Integrated Circuit
BLE	Basic Logic Element
BLIF	Berkeley Logic Interchange Format
CEC	Combinational Equality Checking, ABC komutu cec
CLB	Clustered Logic Block
COGRE	A Configuration Memory Reduced Reconfigurable Logic Cell Architecture for Area Minimization
CPLD	Complex Programmable Logic Device
CUT	Circuit Partitions
FPGA	Field Programmable Gate Array
GAL	Generic Array Logic
HDL	Hardware Description Language
IOB	Input/Output Block
LUT	Lookup Table
MUX	Multiplexer

NPN	Input Negation, Input Permutation and Output Negation
PAL	Programmable Array Logic
PLA	Programmable Logic Array
PLD	Programmable Logic Devices
PROM	Programmable Read-Only Memory
RTL	Register Transfer Level
SAT	Boolean Satisfiability
SEC	Sequential Equality Checking, ABC komutu sec
SPLD	Simple Programmable Logic Device
SRAM	Static Random Access Memory
VHDL	Very High Speed Integrated Circuit HDL
VPR	Versatile Place and Route

1. GİRİŞ

Bu çalışma, FPGA kaynaklarının verimli bir şekilde kullanılarak performanslı tasarımlar oluşturulmasını araştırmaktadır. Yapmış olduğumuz çalışmada, LUT tabanlı FPGA'lerde birbirine NPN denk fonksiyon sayısını artırarak SRAM paylaşımını artırmak ve böylece FPGA'lerde kullanılan alanı ve devre gecikmelerini azaltmak amaçlanmıştır.

Günümüzdeki FPGA'lerde kullanılan teknolojik alt yapının getirmiş olduğu kısıtlar, mantıksal seviyede belli bir amaca hizmet etmek için tasarlanmış devrelerin istenen performans ve maliyette gerçekleştirilmesini zorlaştırmıştır. Mantıksal seviyede, tasarım aşamasındaki esnekliği, uygulama aşamasında FPGA'in kullanmış olduğu donanımların sağlaması zordur. Bununla birlikte istenen devre FPGA kullanılarak gerçekleştirilmiş olsa da istenen performans ölçütlerini sağlamaması sorun teşkil etmektedir.

Mantıksal olarak tasarlanmış bir devrenin FPGA ile gerçekleştirilmesi aşamasında bu gibi sorunları çözmeyi amaçlayan ve performansı iyileştirmeye yönelik çalışmalar devre sentezleme adı altında toplanmıştır. Devre sentezleme, FPGA devre tasarımındaki adımlardan biridir. Sentezleme ile mantıksal olarak tasarlanmış devre, olabilecek en performanslı şekliyle kullanılan FPGA'in kullandığı teknolojik alt yapı ile gerçekleştirilmeye çalışılır. Bu çalışmamızda sentezleme adımında geliştirme yaparak FPGA'de SRAM paylaşımını artırmak amaçlanmıştır.

Sentezlemenin başarılı olması mantıksal olarak tasarlanmış devrenin FPGA ile istenen performans ölçütlerinde uygulanmasına bağlıdır. Başarılı bir tasarım kullanılan FPGA tasarım alanının küçük olmasına, kritik yolun gecikmesinin ve FPGA'in ihtiyaç duyduğu gücün istenen aralıkta olması gibi parametrelere bağlıdır. Saydığımız kıstaslardan tasarım alanının küçülmesi, devredeki gecikmeleri ve kullanılan enerjiyi azalttığı için yapılan çalışmalarda öncelikli olarak çalışılan bir alandır. Yaptığımız çalışma, FPGA'de SRAM paylaşımı ile SRAM kullanım miktarını azaltmayı amaçlayarak, kullanılan yapılandırma bit sayısı, devrelerdeki gecikme ve kullanılan güç gibi performans ölçütlerinde iyileştirme sağlamayı hedefler.

Bu çalışmada, FPGA tasarımında kullanılan alanı azaltmaya yönelik çalışmalar içerisinden NPN kuramını kullanarak mantıksal seviyede tasarlanmış devrelerin oluşturduğu çizgeleri sıkıştırmayı sağlayan yöntemlerde faydalanılmıştır ([1]–[3]). Bu yöntemlerde kullanılan çizge sıkıştırma işleminin NPN benzerliğe bakılarak, birbiri yerine kullanılabilecek fonksiyonların tespiti ile yapılmasından esinlenilmiştir. Bu yöntemler belli bir devre kümesini analiz edip fonksiyonlar arasındaki NPN ilişkileri incelerler. Elde ettikleri sonuçları diğer bütün devrelerin sentezinde kullanırlar. Bu tarz yaklaşım kütüphane tabanlı sentezlemeye girmektedir. Bu tür sentezlemelerde, sentez öncesinde bir kütüphane oluşturulur ve bu kütüphane diğer bütün devreler için kullanılır.

Önerdiğimiz yeni kütüphane tabanlı sentezleme yönteminde, bir devre içindeki fonksiyonların birbirileri ile olan NPN ilişkisine bakılarak diğer fonksiyonlar ile en çok NPN ilişkisine sahip fonksiyonlar sentezlemede referans kütüphane fonksiyonu olarak kullanılmak üzere seçilir. Bu seçimi yaparken geleneksel yöntemler ile belli bir devre kümesini analiz edip seçim yaparken, fonksiyonun bu devre kümesindeki NPN ilişki sayısına bakılırsa etkin bir seçim yapılmamış olur. Çünkü devre içindeki ilişki sayısı değil küme içindeki diğer bütün devrelerdeki fonksiyonlar ile olan ilişki sayısı referans alınmış olur ve sentezlemede kullanılacak fonksiyonların seçimini etkiler. Bizim bu çalışmamızda önerdiğimiz yöntem devreye özel bir çözüm sunmaktadır. Çalışmamızda kullanmak üzere ABC aracının sentezleme fonksiyonuna bir eklenti geliştirtirilmiştir Geliştirdiğimiz bu eklenti girdi olarak vereceğimiz fonksiyona NPN denk olan fonksiyonları sentez için kullanmasını sağlamaktadır.

Girdi olarak kullanacağımız fonksiyon seçimi de şöyle yapılmaktadır. İzlediğimiz yöntemde devre içindeki bütün fonksiyonları sadeleştirip birbirine benzeyen fonksiyonlardan birer adet alırız. Oluşturduğumuz bu sadeleşmiş fonksiyon kümesi orijinal devreninkinden daha az fonksiyon içerdiği için daha hızlı karşılaştırma yapma olanağı sunmaktadır. Bu sadeleştirdiğimiz fonksiyonların birbirleri arasındaki NPN ilişkilerinin sayılarına bakıp büyükten küçüğe sıralarız. Bu sıralamada en çok ilişkiye sahip fonksiyonu sentezleyeceğimiz mimariye göre seçeriz. Örneğin sentez yaparken beş girişli LUT'a sahip FPGA'ler kullanılacak ise beş girişli fonksiyonlar içerisinde diğerleri ile en çok NPN ilişkiye sahip fonksiyonu alırız. Elde ettiğimiz bu fonksiyonu ABC için geliştirdiğimiz eklentiye sentezlemek istediğimiz devre için referans fonksiyon

olarak verdiğimizde devreyi sadece verilen bu referans fonksiyona NPN denk fonksiyonlardan oluşturmaya çalışır. Bu şekilde çıktı olarak elde ettiğimiz devredeki fonksiyonların büyük çoğunluğu birbirine NPN denk olur. Birbirine yapısal olarak benzetilen fonksiyonlar FPGA üzerinde daha az yer kapladığı için kullanılan SRAM oranında azalma olur. Ayrıca bu azalma miktarı kritik yol gecikmesinin ve kullanılan gücün azalmasına da katkı sağlar.

Yazmış olduğumuz tez giriş bölümüyle birlikte beş bölümden oluşmaktadır. Algoritma ile kodların bir kısmı ek olarak tezin sonuna eklenmiştir. Tez içerisindeki giriş bölümü dışındaki bölümlerin içeriği şu şekilde düzenlenmiştir:

İkinci bölüm teknolojik alt yapı ve literatür özetini içerir. Çalışmamızla ilgili mevcut çalışmalara yer verilmiştir. FPGA ile ilgili tanımlar ve yapısal özellikleri açıklanmıştır. Çalışmamızla ilgili olarak LUT tabanlı FPGA mimarilerine yer verilmiştir. FPGA sentezleme çalışmamızın temelini oluşturan NPN denklik üzerine gerekli bilgileri içermektedir. Ayrıca NPN denklik kapsamında incelenmesi gereken sabit atama ve köprüleme gibi özel durumlar bu başlık altında açıklanmıştır. Bununla birlilkte bu bölümde FPGA tasarımındaki adımlar açıklanmıştır. Tasarım, sentezleme, paketleme ve yerleştirme ve bağlama adımları bu bölümde incelenmiştir. Mevcut FGPA tasarım yöntemleri bu bölümde açıklanmıştır.

Üçüncü bölüm LUT seviyesindeki devrelerde NPN denkliği artırmak için geliştirdiğimiz yöntemi açıkladığımız bölümdür. Bu bölümde yapılan işlemler sırasıyla başlıklar altında açıklanmıştır.

Dördüncü bölüm çalışmamız sonucunda elde ettiğimiz sentezlenmiş devrelerin analiz bilgilerini içermektedir. Bu bölümde şu veriler elde edilmiştir

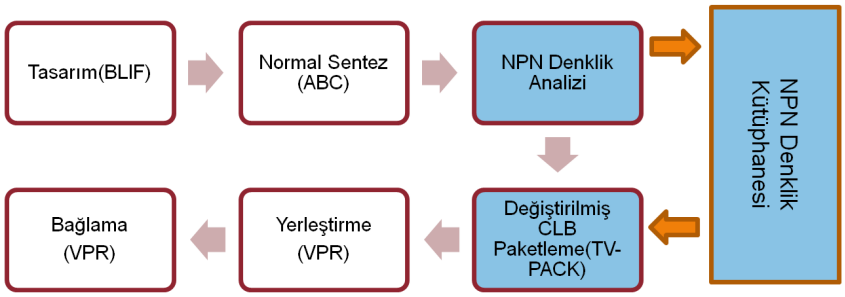
1. Normal ve NPN sentez arasındaki NPN yoğunluk farkı ile NPN ilişkili çift artışının analizi.
2. Normal ve NPN sentez arasındaki devrelerdeki fonksiyon artış farkı ile devrelerin sentez sonrası alan olarak ne kadar büyüdüğünün kontrolü.
3. Sentezlemede kullanılan devre ile ilişkili referans fonksiyon sayısının NPN yoğunluğa etkisi.

4. Çalışmamızda gerçekleştirdiğimiz normal ve NPN sentezleme ile elde ettiğimiz devrelerdeki fonksiyonlardan NPN ilişkili çiftlerin gruplanması sonucu elde edilen NPN küme sayılarının değişimi.
5. FPGA'de sentezlenmiş fonksiyonların oluşturduğu NPN fonksiyon kümelerinin elemanlarının fonksiyon girişlerine göre dağılımları.
6. Devrelerdeki fonksiyonların girişlerine göre dağılımları.
7. Sentezleme için geçen sürelerin analizi.

Beşinci bölüm olan sonuç ve ileride yapılacaklar bölümünde ise yaptığımız çalışma sonrası elde ettiğimiz çıkarımlar ile çalışmamızı geliştirmek amaçlı ileride yapılacaklar listelenmiştir.

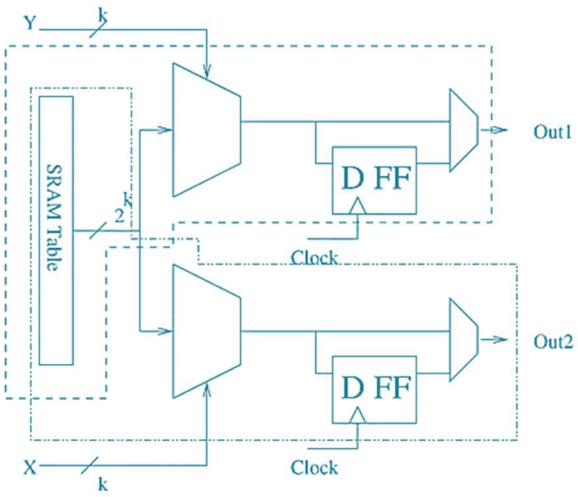
2. ALT YAPI ve LİTERATUR ÖZETİ

FPGA kaynaklarının verimli bir şekilde kullanılması için performanslı tasarımlar sağlamak amacıyla SRAM paylaşımını artırmayı amaçlayan çalışmalar mevcuttur. SRAM tabanlı FPGA mimarisinde SRAM BLE (Basic Logic Element)'ler tarafından paylaşımlı kullanılabilmektedir ([1], [2]). Farklı fonksiyonlara ait doğruluk tabloları SRAM üzerinde ortak bir alana denk gelebilir. Bu durum ancak iki adet fonksiyonun doğruluk tablosu belli noktalarda örtüştüğünde sağlanır. Bu örtüşmeyi sağlamak için yapılan çalışmalardan ilki fonksiyonları parçalar halinde ifade edip benzer parçaları farklı fonksiyonlar için kullanmayı sağlayan çalışmadır ([4]). Örtüşen kısımlar için aynı SRAM hücreleri kullanılacağı için alandan tasarruf yapılmış olur ve programlanan FPGA alanı küçülür ([1], [2], [4]). Bu alandaki ikinci çalışmada ise ilk kez NPN kuramı FPGA alanında kullanılmıştır ve MCNC referans devrelerinde yapılan izomorfizm analizleri sonucu belli başlı fonksiyonların tekrar ettiği gözlenmiştir ([1]). İzomorfizm analizi yapmak için NPN kuramını kullanarak sentezlenen MCNC referans devrelerindeki fonksiyonlar incelenerek aralarındaki NPN ilişkileri gösteren matris oluşturulmuştur. Sonrasında FPGA paketleme adımında TV-PACK aracını geliştirip oluşturulan bu NPN denk fonksiyonlar aynı CLB (Clustered Logic Block) içindeki BLE'lere yerleştirerek izomorfik devrelerin kullandıkları SRAM'lerde örtüşen bitlerin olması sağlanmıştır. Bu örtüşme hem alanda hem de gecikmelerde iyileştirme sağlanmıştır ([1]). Şekil 2-1'de Meyer ve Kocan [1]'ın çalışmasında yaptıkları FPGA tasarım akışındaki değişiklik mavi ile gösterilmiştir. Şekil 2-1'de beyaz ile gösterilen diğer adımlar değiştirilmemiş adımlardır.



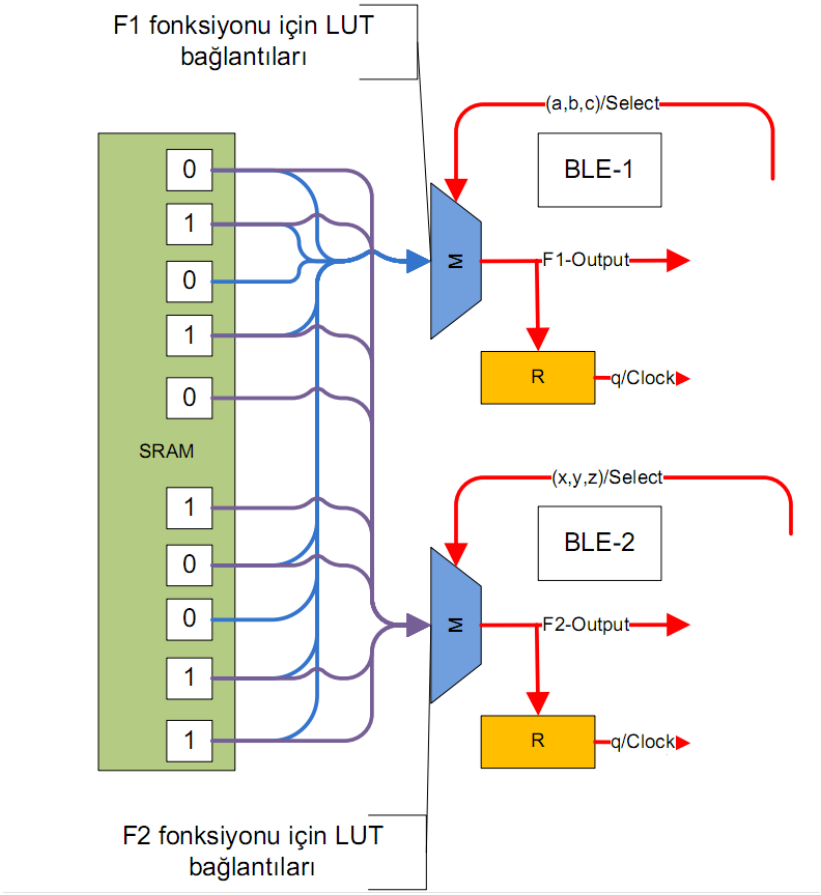
Şekil 2-1- Meyer ve Kocan ([1]) FPGA Tasarım Akışı

Yukarıdaki açıklamalara göre Şekil 2-2'de iki adet BLE'nin SRAM paylaşımı gösterilmiştir.



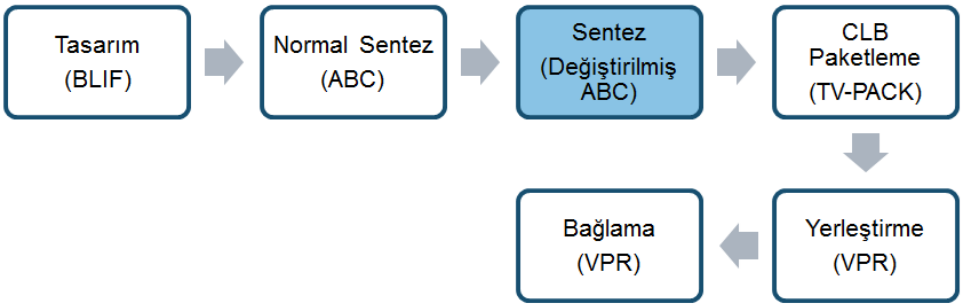
Şekil 2-2-SRAM Paylaşan 2 Adet BLE([1])

Şekil 2-3'da görüldüğü gibi $F1$ ve $F2$ fonksiyonları ayrı fonksiyonlar olmasına rağmen doğruluk tablosu değerleri SRAM üzerinde örtüşmektedir. Örtüşen yerler için ortak SRAM hücresi kullanılır ve tasarım alanında küçülme olur. Bu örtüşmenin sağlanması için devre sentezlenirken, paketleme adımında doğruluk tabloları bir şekilde kesişen veya farklı şekillerde ifade edilen fonksiyonları aynı blokta sentezlemek gerekir. Sonrasında FPGA programlama için oluşturulacak bit dizisi de paylaşıma göre oluşturulup FPGA içine indirilir. Oluşturulan bu bit dizisi de paylaşım sayesinde normalden daha küçük olur ([1], [2]).



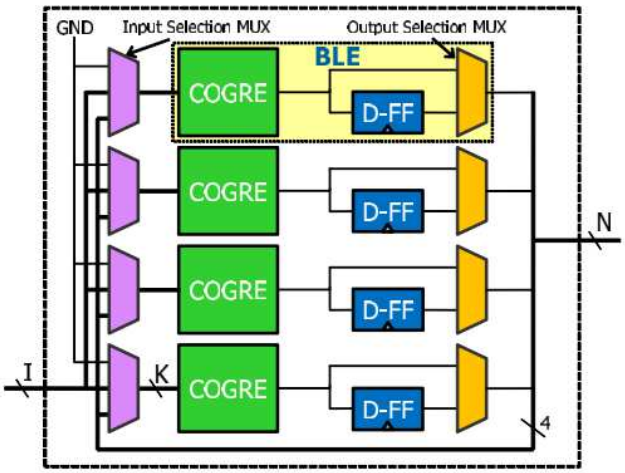
Şekil 2-3-FPGA SRAM Paylaşımı

Bu alanda yapılan üçüncü çalışma COGRE (A Configuration Memory Reduced Reconfigurable Logic Cell Architecture for Area Minimization, [3]) ise [1]'de NPN kuramının kullanımından yola çıkarak paketleme adımında yapılan değişiklikten farklı olarak sentezleme adımında değişiklik yapmıştır. MCNC referans devrelerindeki fonksiyonları inceleyip fonksiyonların devrelerde görülme sıklığını hesaplamışlardır. En çok görülen altı fonksiyona NPN denk olan devre fonksiyonlarını ABC aracına geliştirdikleri eklenti ile sentezleme adımında kullanmasını sağlamıştır ([3]). Şekil 2-4'de COGRE'deki FPGA tasarım akışı değişikleri mavi ile gösterilmiştir.



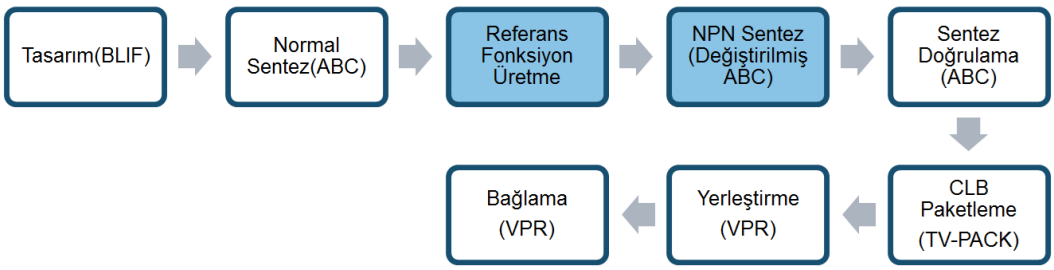
Şekil 2-4- COGRE FPGA Tasarım Akışı ([3])

COGRE ile sentez aşamasında altı fonksiyona NPN denk fonksiyonlardan oluşan devrede fonksiyonlar arası benzerlik artmıştır. Bu artış fonksiyonların doğruluk tabloları üzerinde örtüşmeleri artırmıştır ve normal senteze göre daha fazla SRAM paylaşımı sağlamıştır ([3]). Bu çalışmada temel kısıt kullanılan referans fonksiyonların ilk altı fonksiyondan oluşmasıdır. Şekil 2-5'de COGRE'nin temsili örnek yapısı verilmiştir.



Şekil 2-5-COGRE Yapısı ([3])

Bizim geliştirdiğimiz yöntemde COGRE'den farklı olarak referans fonksiyon seçimi devreye özel bir yöntem ile yapılmaktadır. Ayrıca devreden bağımsız istenilen sayıda referans fonksiyonlarını da kullanımına olanak sağlar. Geliştirdiğimiz yöntem COGRE gibi sentez aşamasında verilen referans fonksiyonlara NPN denk olan fonksiyonların devrede kullanılmasını sağlar. Çalışmamızda Nauty Kütüphanesi ile NPN denklik karşılaştırması hızlı bir şekilde yapılmaktadır. Şekil 2-6'da bu çalışmamızda değiştirdiğimiz FPGA tasarım akışı gösterilmiştir. Şekil 2-6'daki akışta değiştirilen adımlar mavi ile gönderilmiştir. Değiştirilen sentezleme adımının öncesine referans fonksiyon üretme adımı konulmuştur.



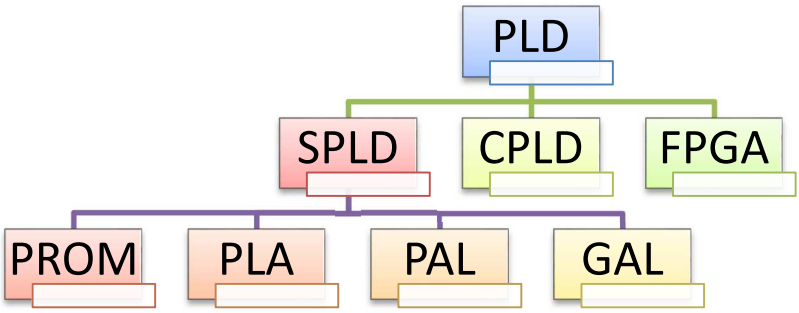
Şekil 2-6 – Çalışmamızdaki FPGA Tasarım Akışı

İlerleyen bölümlerde literatürde kullanılan teknolojilerin ve yöntemlerin detayları verilmiştir.

2.1. FPGA

2.1.1. FPGA Tanımı

FPGA , mantıksal fonksiyonları gerçekleştirmek için geliştirilmiş, programlanabilen bir PLD (Programmable Logic Device)'dir. Özel bir işi yapmak için tasarlanmış ASIC (Application Specific Integrated Circuit) donanımların fabrikasyon sonrasında programlanabilen bir türüdür ([2], [5]). Aşağıdaki Şekil 2-7'de FPGA'ler ile birlikte diğer PLD türleri verilmiştir.



Şekil 2-7-PLD Türleri ve FPGA ([5, p. 29])

Yukarıdaki Şekil 2-7'de yer alan SPLD (Simple Programmable Logic Device) üzerinde mantıksal kapı matrislerinin bağlantılarının düzenlenmesi ile programlanabilen bir donanımdır. SPLD türlerine kısaca değinecek olursak; PROM (Programmable Read-Only Memory), fabrikasyon aşamasında programlanmış olan sadece okunabilen mantıksal VE dizisi ile tek seferlik programlanabilen mantıksal VEYA dizisinden oluşmaktadır. PLA (Programmable Logic Array) fabrikasyon sonrasında tek seferlik programlanabilen mantıksal VE dizisi ve mantıksal VEYA dizisinden oluşmaktadır. PAL(Programmable Array Logic) , tek seferlik programlanabilen mantıksal VE dizisi ve fabrikasyon

aşamasında programlanmış sadece okunabilen mantıksal VEYA dizisinden oluşmaktadır. GAL (Generic Array Logic), üzerinde tekrar programlanabilen mantıksal VE dizisi ile fabrikasyon aşamasında programlanmış sadece okunabilen mantıksal VEYA dizisinden oluşmaktadır ([5]).

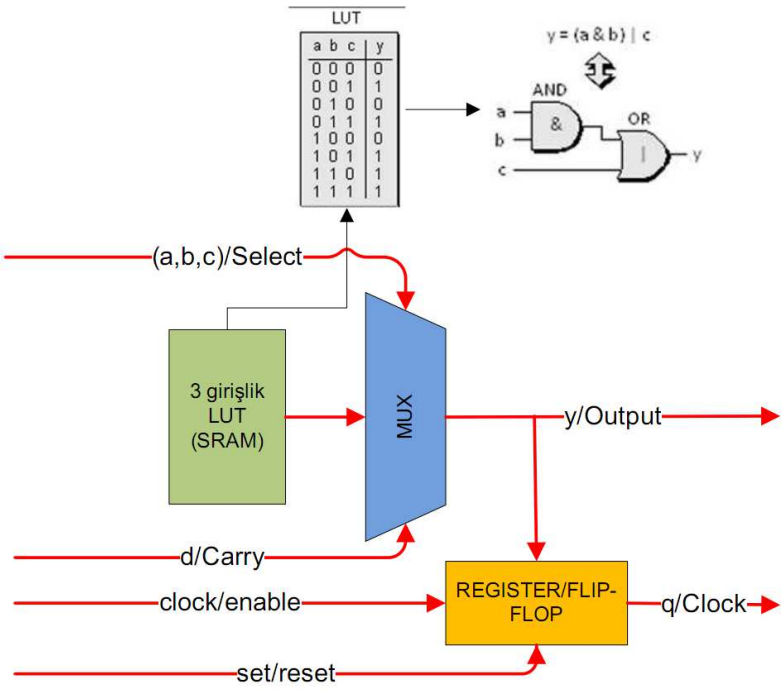
SPLD'lerin bir araya gelmesi ve bunların aralarındaki bağlantıları düzenleyen birleşenlerin birleşmesi ile SPLD'ye göre daha gelişmiş CPLD (Complex Programmable Logic Device) donanımlar ortaya çıkmıştır ([5]). Yukarıda açıklanan donanımlara göre FPGA daha yüksek özelliklere sahip yeniden programlanabilme imkânı sunduğu için çalışmamızda tercih edilmiştir.

2.1.2. FPGA Bileşenleri

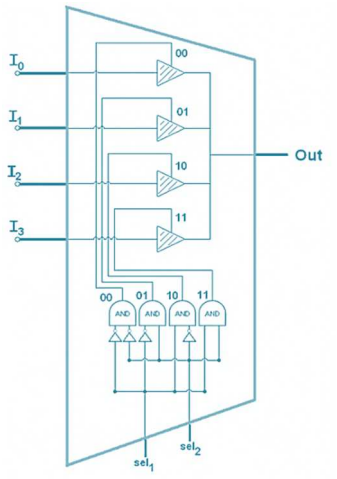
Bu kısımda açıklayacağımız FPGA bileşenleri SRAM tabanlı FPGA'ler temel alınarak açıklanmıştır. SRAM tabanlı FPGA'lerin en temel bileşenlerinde biri olan SRAM, FPGA'lerde yapılandırma bitlerini (doğruluk tablolarını) tutmak için kullanılmaktadır. SRAM güç verildiği sürece içeriğini korur ve dinamik belleklere göre daha hızlı çalışıp, daha az güce ihtiyaç duyması nedeniyle FPGA'lerde tercih edilmektedir. Temel FPGA bileşenleri programlanabilen en küçük mantıksal birim BLE'dir, BLE'lerin bir araya gelerek oluşturduğu CLB, CLB'ler arasındaki iletişimi sağlayan bağlantı ve komütatör birimleri ile dış ortamla iletişimi sağlayan IOB (Input/Output Block)'tan oluşmaktadır. Temel FPGA bileşenleri daha detaylı bir şekilde takip eden başlıklarda anlatılmıştır.

2.1.2.1. BLE (Basic Logic Element)

MUX (Multiplexer) tabanlı FPGA mimarisi Şekil 2-8'de görülen MUX içeren programlanabilen BLE isimli hücrelerden oluşmaktadır. Şekil 2-8'de verilen MUX ([5]) Şekil 2-9'de detaylı olarak gösterilmiştir. FPGA tasarım yapılırken mantıksal kapılar yerine devrenin doğruluk tablosu kullanılır ([1], [2]). Bu şekilde mantık kapıları ile karmaşık tasarıma sahip devrelerin gerçekleştirilmesi kolaylaşır. FPGA bileşenlerinden BLE gerçekleştirilmek istenen fonksiyonu iki değişkene bağlı olarak gerçekleştirebilir, bunlar BLE içerisindeki MUX'ın giriş sayısı ve FPGA'in sahip olduğu SRAM miktarıdır. SRAM üzerinde çalışması istenen fonksiyonun doğruluk tablosu tutulur.



Şekil 2-8-BLE Mimarisi



Şekil 2-9-MUX Mimarisi

BLE mimarisi Şekil 2-8'de gösterildiği üzere SRAM içine programlanmış doğruluk tablosuna (LUT) göre seçim uçlarından seçilen değeri, çıktı olarak verir ([1], [2], [5]). FPGA programlanırken yapılan düzenlemede, SRAM'in düzenlenmesi ile donanımsal olarak değişiklik yapılmaz, sadece yapılandırma bitleri değiştirilir. Örnek olarak Şekil 2-8'de mantıksal devreler ile gerçekleştirilmiş devrenin karşılığı olan yapılandırma bitleri SRAM'e programlanıp BLE ile gerçekleştirilmiştir.

BLE temelde üç adet bileşenden oluşmaktadır. Modern mimari de flip-flop devrelerindeki farklılıklar yüzünden ek bileşenler olabilir. Bunun dışında basit haliyle incelemek gerekirse BLE bileşenleri şunlardır:

1. k girişli bir fonksiyon için 2^k bit doğruluk tablosunu tutan SRAM hücre bloğu barındırır. Örneğin; $0111000 \dots 001$
2. Doğruluk tablosundaki değerlerin seçimi için çoklayıcı barındırır. Özellikle MUX tabanlı mimari için kullanılmaktadır.
3. Saat sinyali ile çalışmasını sağlayan flip-flop devreleri içerir.

Bir BLE'nin tuttuğu doğruluk tablosuna ve çalışma prensibine şu şekilde örnek verebiliriz; k , BLE için giriş sayısı olmak üzere k'ya 3 değeri verildiğinde doğruluk tablosu için $2^3 = 8$ bit gerekir. Tablo 2-1'de örnek bir LUT verilmiştir.

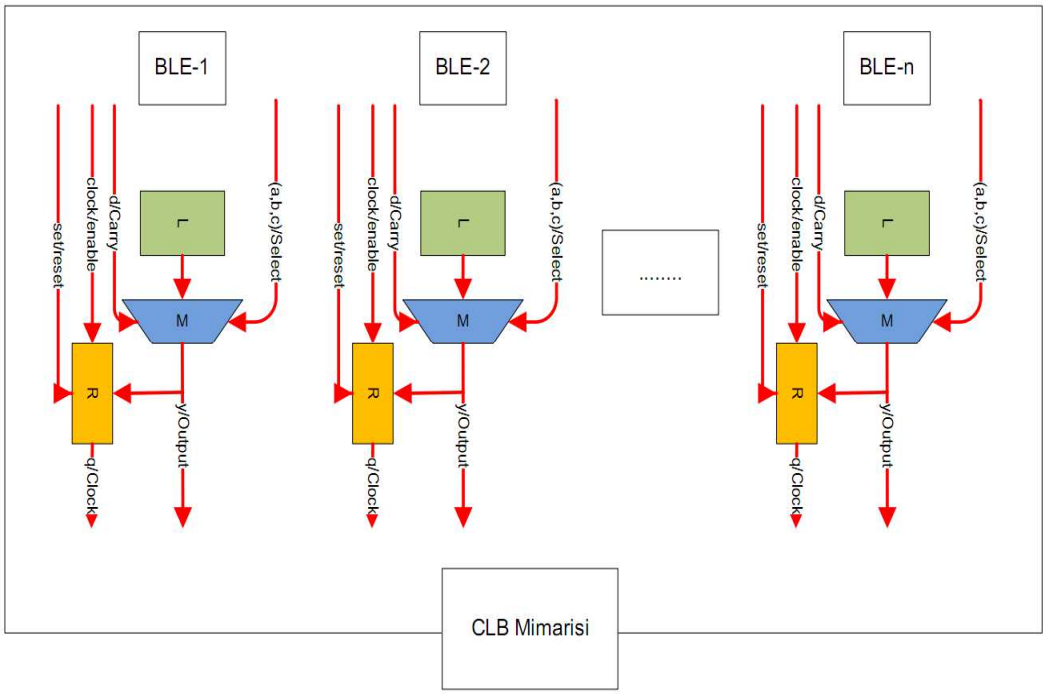
Tablo 2-1- Üç girişli fonksiyon için LUT Örneği

a	b	c	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Burada a , b ve c girişler ve f ise çıkıştır. FPGA ile programlama yaparken SRAM'de f değerlerini ifade eden veri tutulur. BLE girişlerine değer verdiğimizde BLE bileşeni olan çoklayıcı ile bu doğruluk tablosundaki değerlere denk gelen yerler seçilir [2]. Örnek olarak a için bir, b için sıfır ve c için bir şeklinde BLE girişlerini düzenlediğimizde çoklayıcı bu değere eşit değeri Tablo 2-1'de f sütununa denk gelen bir değerini SRAM'den seçer ve çıktı olarak verir. Yukarıdaki tabloya denk BLE devresinin mantıksal fonksiyonu $y = (a \wedge b) \vee c$ dir. Bu şekilde istenilen bir fonksiyonun sadece doğruluk tablosu FPGA için programlanıp kullanılabilir. Buradaki kısıtlardan en önemlisi BLE yapısında kullanılan MUX giriş sayısıdır. Eski mimarilerde üç, dört ve beş girişli BLE'ler kullanılırken ([1], [2]) Xilinx Virtex-6 gibi modern mimarilerde sekiz girişli MUX'lar kullanılarak programlanabilecek fonksiyon kapasitesi artırılmıştır ([6]).

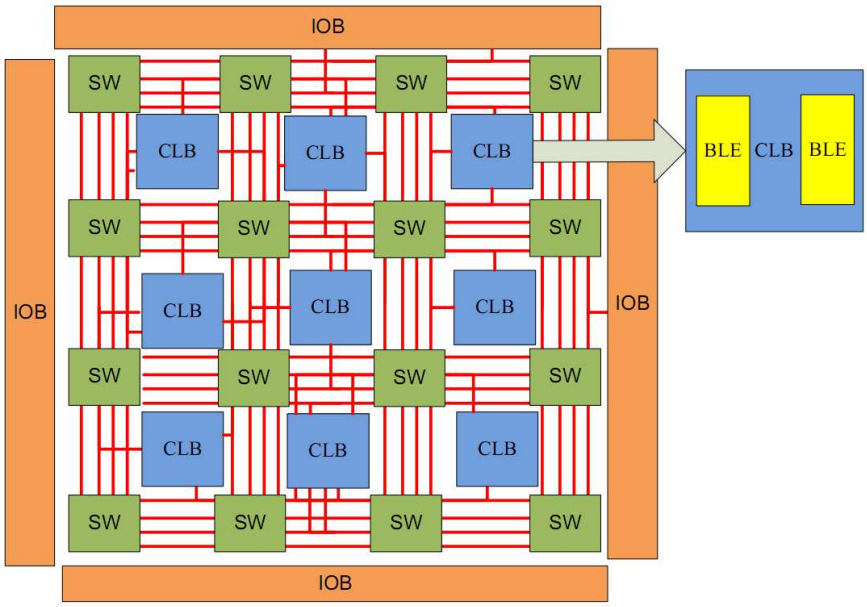
2.1.2.2. CLB (Clustered Logic Blok)

FPGA mimarisindeki mantıksal bloklar (CLB), BLE'lerin bir araya gelmesi ile Şekil 2-10'de görüldüğü gibi oluşur ([1], [2]).



Şekil 2-10-CLB Mimarisi

Mantıksal blokların (CLB) FPGA mimarisindeki temsili yerleşimi Şekil 2-11'de gösterilmiştir.

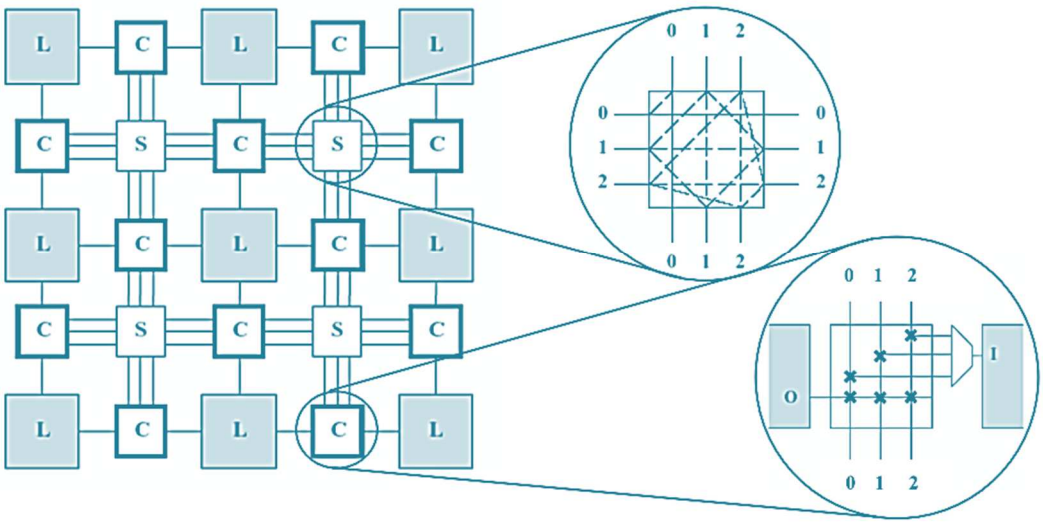


Şekil 2-11-FPGA Mimarisi ve Ada Bağlantı Yapısı

Modern mimaride CLB'ler Slice olarak isimlendirilir. Xilinx'in sağlamış olduğu mimarilerde iki adet Slice türü mevcuttur. Bu Slice türleri SRAM'i harici hafıza ve mantıksal fonksiyon olarak kullanılabilecek şekilde tasarlanmış Slice M türü bloklar ile SRAM'i sadece mantıksal fonksiyonların uygulanabileceği şekilde tasarlanmış Slice L türü bloklardır ([6]).

2.1.2.3. Ara Bağlantılar ve Komütatörler

I/O Blokları ve CLB'lerin kendi aralarında programlanabilen bağlantılar (komütatörler) vardır. Bağlantıların birleştiği noktalarda, Şekil 2-11 ve Şekil 2-12'da SW ve S ile gösterilen, programlanabilen komütatörler bulunur. Bu komütatörler kullanılarak CLB ve I/O blokları arasındaki bağlantılar düzenlenir ([1], [2], [7]).

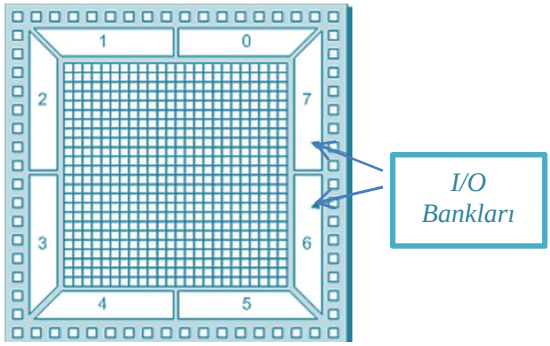


Şekil 2-12-FPGA Komütatör Yapısı ([7])

Şekil 2-12'da görüldüğü üzere komütatör (S) içinde komütatöre giren girişler arasındaki bağlantıları ayarlamak için matris şeklinde bağlantılar mevcuttur. Bu bağlantıların koparılması veya birleştirilmesi ile girişlerin birbirleri olan ilişkisi düzenlenir ([7]).

2.1.2.4. I/O Blokları (IOB)

I/O blokları FPGA'in dış ortamla olan bağlantısını sağlar. FPGA veri giriş çıkışları I/O blokları üzerinden gerçekleşir. I/O blokları kendi içinde banklara ayrılmıştır. Xilinx'in tasarımlarında her FPGA'de ortalama dokuz ile otuz arasında bank bulunabilir. Bir bank ortalama kırk I/O içerir ([6]). Şekil 2-13'de bankların FPGA üzerindeki yerleşimi gösterilmiştir ([5, p. 90]).

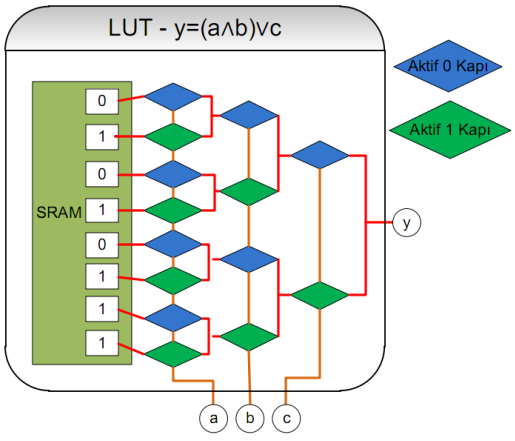


Şekil 2-13-FPGA I/O Blokları ve Banklar ([5, p. 90])

2.1.3. Programlanan Hücre Teknolojilerine Göre FPGA Türleri

2.1.3.1. LUT Tabanlı FPGA Mimarileri

LUT tabanlı mimarilerde güç verildiğinde aktif veya pasif olan kapılar yardımı ile akım yönlendirilerek, doğruluk tablosunun değerinin tutulduğu SRAM hücresinin sıfır veya bir olan bit değeri çıktı olarak verilir ([5]). Şekil 2-14'de gösterilen a, b ve c girişlerine göre SRAM hücresindeki değerler y fonksiyon çıktısı olarak verilir. Mavi olanlar güç verildiğinde kapalı olan kapılardır, yeşil olanlar ise güç verildiğinde açık olan kapılardır.



Şekil 2-14-Programlama Hücresi – LUT Tabanlı Mantık Hücresi

2.2. NPN DENKLİK VE HİPER ÇİZGE İZOMORFİZMİ

Çizge izomorfizmi, çizgelerin eş şekilli olup olmadığını inceler. Hiper çizge izomorfizmi ise n-boyutlu çizgelerin eş şekilli olup olmadığını inceler. Bu iki kuram bizim bu çalışmamızda kullandığımız NPN kuramının temelini oluşturmaktadır.

NPN iki veya daha çok mantıksal fonksiyonun doğruluk tabloları arasında ilişki kurarak farklı fonksiyonların doğruluk tablolarını veya fonksiyonları birbirleri cinsinden ifade etmeye olanak sağlayan matematiksel bir modeldir. NPN kuramı mantıksal devrelerin doğrulanması ve sentezlenmesinde devrelerin aynı fiziksel ağa sahip olduklarını incelemek için kullanılır ([8]). NPN kuramı aşağıda açıklanan geleneksel şekilde bütün permütasyonlar hesaplanarak yapılabilir. Bunun performans açısından uygun olmadığı aşikârdır. Bu sorunu çözmek için NPN kuramı Luks tarafından hiper çizge izomorfizm problemine dönüştürülüp çözülür ([1]–[3], [8]).

NPN ve alt denklikleri incelenirken karşılaştırılan fonksiyonların girdi sayıları eşittir. Eşit olmayan durumlarda sabit değer atama ve köprüleme işlemleri ile girdi sayıları eşitlenip denklik incelemesi yapılacak hale getirilir ([1]–[3], [8], [9]). Girdilerin eşit olmadığı durumda yapılan bu işlemler NPN denklik anlatımından sonra açıklanmıştır

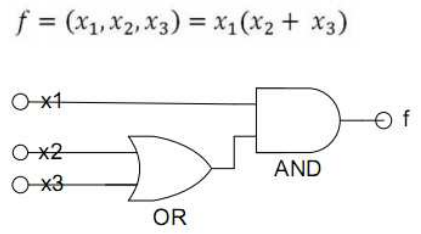
İki fonksiyonun f ve g NPN denk olup olmadığını araştırırken NPN incelemesini hiper çizge izomorfizm problemine dönüştürmeden bütün karşılaştırma olasılıklarını sınayarak araştırma yöntemi ve alt denklikleri aşağıda açıklanmıştır. Aşağıdaki açıklamalarda kullanılan sembollerin açıklamaları şunlardır:

1. φ girişlerin tersinin alındığı operatörü simgeler
2. א çıktıların tersinin alındığı operatörü simgeler
3. β ise girişlerin permütasyonlarının alındığı operatörü simgeler

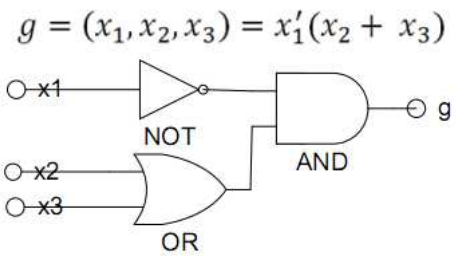
2.2.1. NI Denkliği

Eğer bir mantıksal fonksiyonun girdilerinin herhangi bir alt kümesinin tersini alarak başka bir fonksiyonun çıktılarını üretebiliyorsak bu iki fonksiyon birbirine NI denk denir ([2]).

Eğer $f = (x_1, x_2, \dots, x_n)$ fonksiyonumuz ve girdilerin tersi alınarak oluşturulan fonksiyon kümemiz $X^{\varphi}(x_1, x_2 \dots, x_n) = (x_1^{\varphi_1}, x_2^{\varphi_2}, \dots, x_n^{\varphi_n})$ ise ([9]); Herhangi bir g fonksiyonun NI denkliği için $f \equiv g \Leftrightarrow f = (go\ X^{\varphi})$ şartı sağlanmalıdır ([2], [9]). Örneğin; $f = (x_1, x_2, x_3) = x_1(x_2 + x_3)$ (Şekil 2-15) ile $g = (x_1, x_2, x_3) = x_1'(x_2 + x_3)$ (Şekil 2-16) fonksiyonları arasındaki ilişki incelendiğinde, f fonksiyonunda x_1 değerlerinin tersi alındığında g fonksiyonu çıktıları elde edildiği görülür. Aynı işlem g fonksiyonu için de yapıldığında sonuç aynıdır. Bu durumda f fonksiyonu ile g NI denktir denir. Bu örnekte sadece x_1 in tersini alarak istediğimiz sonucu bulduk. Girdi adedi n olan bir fonksiyonda bu işlem bütün alt kümeler için yapılabilir. Herhangi biri ile denklik sağlanırsa NI denktir denir.



Şekil 2-15Örnek Fonksiyon-1

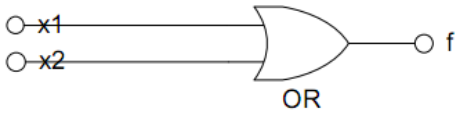


Şekil 2-16-Örnek Fonksiyon-1 Tersi

2.2.2. NO Denkliği

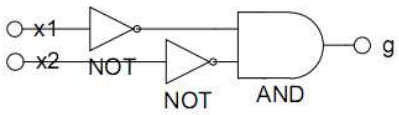
Eğer bir mantıksal fonksiyonun çıktıları tersini alarak başka bir fonksiyonun çıktılarını üretebiliyorsak bu iki fonksiyon birbirine NO denk denir ([2]). Eğer $f = (x_1, x_2, \dots, x_n)$ fonksiyonumuz ise herhangi bir g fonksiyonuna NO denkliği için $f \equiv g \Leftrightarrow f = (g)^{\aleph}$ şartı sağlanmalıdır ([9]). Örneğin;$f = (x_1, x_2) = x_1 + x_2$ (Şekil 2-17) ile $g = (x_1, x_2) = x_1'.x_2'$ (Şekil 2-18) fonksiyonları incelendiğinde $(x_1 + x_2)' = x_1'.x_2'$ olduğu görülür. Buna göre $f = (x_1, x_2)$ fonksiyonun çıktılarını tersi $g = (x_1, x_2)$ fonksiyonun çıktılarına denk olmaktadır. Öyleyse bu iki fonksiyon birbirine NO denktir denir. Burada NI denklikte girdilerde yapıldığı gibi herhangi bir alt kümesinin değil çıktıların tamamının tersi alınır.

$f = (x_1, x_2) = x_1 + x_2$



Şekil 2-17-Örnek Fonksiyon-2

$g = (x_1, x_2) = x_1'.x_2'$

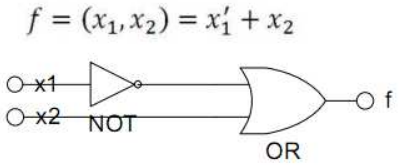


Şekil 2-18-Örnek Fonksiyon-2 Çıktısı Ters Çevrilmiş

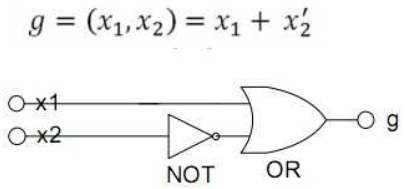
2.2.3. P Denkliği

Eğer bir mantıksal fonksiyonun girdilerinin permütasyonları ile başka bir fonksiyon türetebiliyorsak bu iki fonksiyon birbirine P denk denir ([1], [2]). Eğer $f = (x_1, x_2, \dots, x_n)$ fonksiyonumuz ve girdilerin permütasyonları ile oluşturulan fonksiyon kümemiz $X_\pi(x_1, x_2 \dots, x_n) = (x_{\pi(1)}, x_{\pi(2)}, \dots, x_{\pi(n)})$ ise; Herhangi bir g fonksiyonun P denkliği için $f \equiv g \Leftrightarrow f = (g \, o X_\pi)$ şartı sağlanmalıdır ([9]). $f = (x_1, x_2) = x_1' + x_2$ (Şekil 2-19) ile $g = (x_1, x_2) = x_1 + x_2'$ (Şekil 2-20) fonksiyonları incelendiğinde f fonksiyondaki girdilerin yerlerini değiştirince g fonksiyonu elde ettiğimiz görülür. Aynı

şekilde g fonksiyonunun da girdilerinin yerlerini değiştirince f fonksiyonunu elde ederiz. Buna göre f ve g fonksiyonları kendi aralarında P denktir.



Şekil 2-19-Örnek Fonksiyon-3



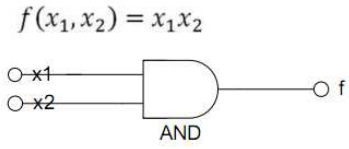
Şekil 2-20-Örnek Fonksiyon-3 Girdi Permutasyonu Alınmış

Bu operasyonlar ile fonksiyonlar birbiri cinsinden yazılabilir hale getirilir.

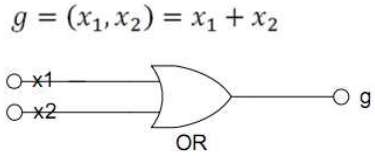
2.2.4. NPN Denklik

Yukarıdaki tanımlardan yola çıkarak NPN denklik şartını şöyle ifade edebiliriz.

$f \equiv g \Leftrightarrow f = (g^{\aleph} \mathrm{o} X_{\pi} \mathrm{o}\, X^{\varphi})$ şartını sağlayıp NO ,P ve NI denkliklerinden en az bir tanesini sağlayan her g fonksiyonu f fonksiyonuna NPN denktir ([1], [2], [9]). Örneğin; $f(x_1, x_2) = x_1 x_2$ (Şekil 2-21) ve $g = (x_1, x_2) = x_1 + x_2$ (Şekil 2-22) fonksiyonlarını inceleyelim.



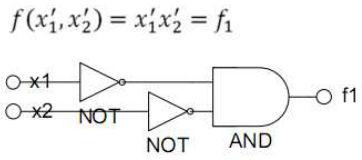
Şekil 2-21-Örnek Fonksiyon-4



Şekil 2-22-Örnek Fonksiyon-5

1. Fonksiyon f'in girdilerinin tersini aldığımızda;

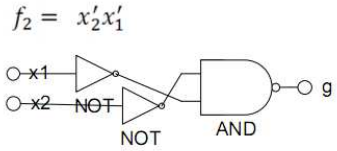
$f(x_1', x_2') = x_1'x_2' = f_1$ (Şekil 2-23) fonksiyonunu elde ederiz.



Şekil 2-23-Örnek Fonksiyon-4 Girdi Tersi Alınmış

2. Elde ettiğimiz fonksiyonun girdilerini permütasyon ile yer değiştirdiğimizde;

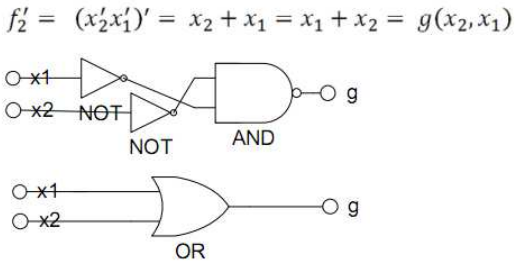
$f_2 = x_2'x_1'$ (Şekil 2-24) fonksiyonunu elde ederiz.



Şekil 2-24-Örnek Fonksiyon-4 Girdi Tersi Alınmış Permütasyonu

3. Elde ettiğimiz bu yeni fonksiyonun tersini aldığımızda;

$f_2' = (x_2'x_1')' = x_2 + x_1 = x_1 + x_2 = g(x_2, x_1)$ (Şekil 2-25) olduğu görülür. Buna göre f ve g fonksiyonları NPN denktir.



Şekil 2-25-Örnek Fonksiyon-4 ve 5 NPN Denkliği

2.2.5. NPN Kontrolü İçin Özel Durumlar

Girdiler eşit olmadığında denklik incelemesi yapılamaz. Bu durumda girdileri eşitlemek için çeşitli operasyonlar gerçekleştirilir. Bu operasyonlar, sabit değer atama ve köprüleme işlemleridir ([2]).

2.2.5.1. Sabit Atama

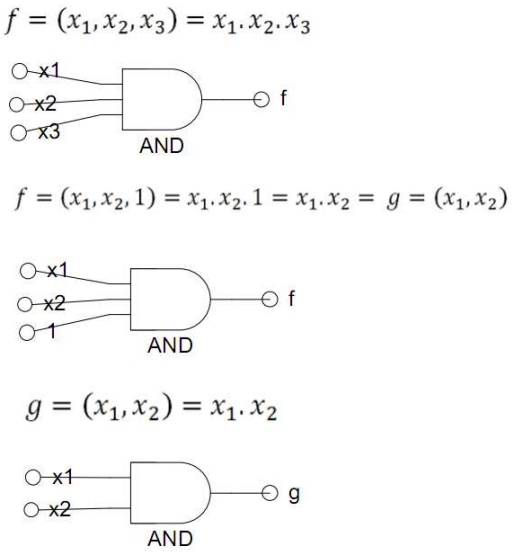
İki fonksiyon arasında girdi sayısı küçük olan fonksiyon analiz edilip diğer fonksiyonun pozitif veya negatif kofaktörü olup olmadığı tespit edilir ve böylelikle fonksiyonların birbirine denk olup olmadığı tespit edilir. Pozitif kofaktör için girdi sayısı fazla olan fonksiyonun fazla olan girdilerine bir verilir, negatif kofaktör için ise sıfır verilir. Matematiksel olarak ifade edersek ([10]);

$f = (x_1, x_2, \dots, x_i, \dots, x_n)$ olsun

Pozitif kofaktör : $h = f_{xi}$ ise $h = \{x | f(x_1, x_2, \dots, 1, \dots, x_n) = 1\}$ dir.

Negatif kofaktör: $h = f_{\overline{xi}}$ ise $h = \{x | f(x_1, x_2, \dots, 0, \dots, x_n) = 1\}$ dir.

Örnek: $f = (x_1, x_2, x_3) = x_1.x_2.x_3$ ve $g = (x_1, x_2) = x_1.x_2$ olsun f ve g fonksiyonlarını denkleştirmek için $x_3 = 1$ verirsek $f = (x_1, x_2, 1) = x_1.x_2.1 = x_1.x_2 = g = (x_1, x_2)$ olur. Buna göre g fonksiyonu f fonksiyonunun pozitif kofaktörü denir ve iki fonksiyon birbirine denktir (Şekil 2-26).



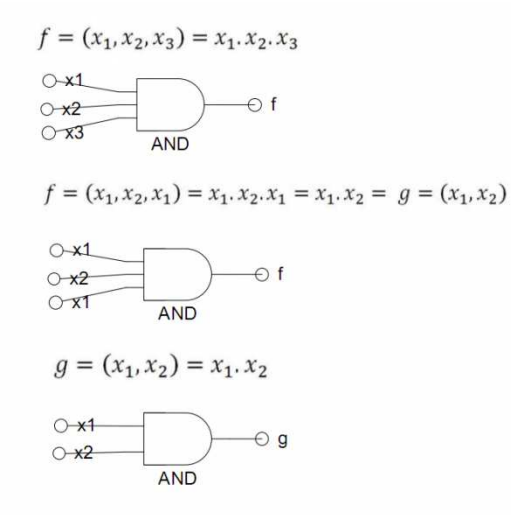
Şekil 2-26-Sabit Atama Örneği

2.2.5.2. Köprüleme

Girdi sayıları farklı iki fonksiyondan girdi sayısı büyük olan fonksiyonun girdilerini, girdi sayısı küçük olan fonksiyonun girdilerine eşitlemek için girdiler tekrarlanır ([2]).

Örnek-1;

$f = (x_1, x_2, x_3) = x_1.x_2.x_3$ ve $g = (x_1, x_2) = x_1.x_2$ olsun. F ve g fonksiyonlarını denkleştirmek için $x_3 = x_1$ verirsek $f = (x_1, x_2, x_1) = x_1.x_2.x_1 = x_1.x_2 = g = (x_1, x_2)$ olur. Buradan da denk oldukları görülür (Şekil 2-27).

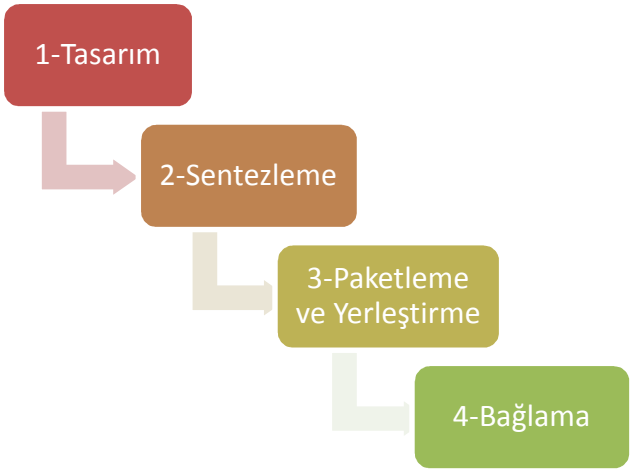


Şekil 2-27-Köprüleme Örneği

NPN denkliğe pratikte bir örnek vermek gerekirse; $f = (x_1, x_2, \dots, x_n)$ adında bizim devremizin bir kısmını ifade eden mantıksal bir fonksiyon olsun. Elimizde şu anki teknoloji ile gerçekleştirebileceğimiz devreler kümesi olsun $p = [g_1, \dots, g_n]$. NPN modelini kullanarak gerçekleştirmek istediğimiz devremizi (f), p kümesindeki devrelere uydurarak onlar ile gerçekleştirmiş oluruz ([2], [9]). Yani f fonksiyonumuzu p kümesinin elemanı olan fonksiyonlarla NPN modeline uygun bir şekilde ifade etmiş oluruz. Böylelikle mevcut donanımlarla farklı fonksiyonları gerçekleştirmiş oluruz veya mevcut bir donanımı birden fazla görev yapabilecek şekilde programlayıp programlanan alandan tasarruf sağlarız.

2.3. BİLGİSAYAR DESTEKLİ FPGA TASARIMI

FPGA tasarımı Şekil 2-28'de gösterilen adımlardan oluşmaktadır. FPGA tasarımında, performanslı tasarımlar elde etmek için ara adımlar konulabilir ve mevcut adımlarda değişiklikler yapılabilir.



Şekil 2-28- FPGA Tasarım Akışı

FPGA tasarımındaki temel unsurları tasarımın HDL (Hardware Description Language) ile ifade edilmiş biçimi, tasarım kısıtları ve programlanmak istenen FPGA özellikleri oluşturmaktadır ([11]).

FPGA tasarımında devreyi ifade etmek için kullanılan en yaygın programlama dilleri HDL, Verilog ve VHDL (Very High Speed Integrated Circuit HDL)'dir. Bunlar devreyi RTL (register transfer level)'de her bir saat sinyalinde operasyonların ne olacağı şeklinde ifade eder. Ayrıca C ve SystemC gibi üst seviye diller de HDL tanımlama için kullanılmaktadır. Bunların yanı sıra domain tabanlı tasarımlarda Matlab ve Simulink araçları da kullanılabilmektedir.

FPGA tasarımında bir başka unsur olan tasarım kısıtları, yani FPGA tasarlanırken beklenen çıktıyı sağlayacak kısıtlardan bazıları şunlardır:

a) FPGA işleme hazır hale gelene kadar geçen süre
b) Devredeki hızı etkileyecek olan gecikmeler
c) Tasarlanmış devrenin FPGA üzerinde ne kadar yer işgal edeceği
d) FPGA çalışırken ne kadar güç harcanacağı
e) Tasarımın gerçekleştirilmesinin maliyeti yani devrenin karmaşıklığı

f) Kritik ve kritik olmayan işlemlerin belirlenmesi

Yukarıda belirttiğimiz unsurlar FPGA seçimimizi doğrudan etkilemektedir. FPGA'lerde kullanılan blok yapısı, mimarisi ve kapasitesi gerçekleştireceğimiz devrenin maliyetini oluşturur ve tasarımın karmaşıklığını belirler. Eğer çok küçük kapasitede bir FPGA seçerseniz tasarım aşamasında çok fazla iyileştirme yapmanız gerekebilir aksi halde devreniz FPGA içine sığmayacaktır. Çok büyük kapasitede FPGA aldığınızda tasarım kolaylaşabilir ama bu sefer de maliyet yükselecektir. Bu gibi nedenlerden ötürü seçim yapılırken parametreler en uygun değerde seçilir ([11]).

Takip eden bölümlerde FPGA tasarım adımları anlatılacaktır.

2.3.1. Tasarım

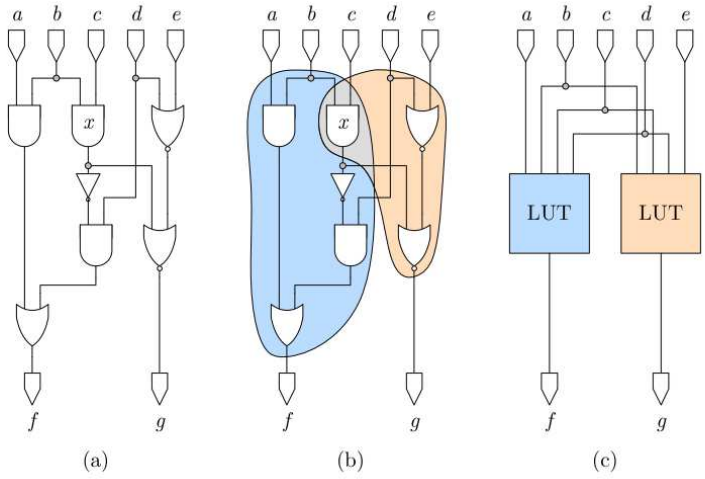
Tasarım aşaması, amacımıza hizmet edecek olan devrenin HDL ile oluşturulup tasarlandığı adımdır. Tasarım esnasında en çok kullanılan HDL dilleri Verilog ile VHDL'dir. Bu aşamada yazmaçlar ile veri yolları RTL olarak tasarlanır ve hafıza mimarisi ve mantıksal fonksiyon çizgeleri oluşturulur. Tasarım aşamasında platformdan bağımsız veri yollarında iyileştirme yapılarak daha verimli çıktı elde edilir ([11]).

2.3.2. Sentezleme

Sentezleme adımında HDL ile tanımlanan devre FPGA'in sahip olduğu en temel mantıksal elemanlar kullanılarak oluşturulur. Bu işlem yapılırken alan, gecikmeler ve güç, kullanılan teknolojiye göre iyileştirilerek Şekil 2-29'de gösterildiği gibi devre tekrardan oluşturulur ([12], [13]).

Şekil 2-29 – Mantıksal Devre Sentezi

İlk LUT tabanlı sentezleme Chortle ve Mis-FPGA ile gerçekleştirilmiştir ve sonrasında bunlarla ilgili birçok performans iyileştirme algoritması geliştirilmiştir ([12]). LUT sentezlemesine örnek olarak Şekil 2-30'de bir devrenin teknolojik elemanlarla eşleştirilmesi ve gerçekleştirilmesi, kapsama probleminin çözümü yaklaşımı ile verilmiştir ([14]).



Şekil 2-30 – (a)Orijinal Devre (b)Eşleme (c)Sentezlenmiş Devre[14]

Sentezleme aşamasında, tasarım aşamasındaki gösterimimiz, programlamak istediğimiz FPGA donanıma uygun bir şekilde devre bileşenlerinin bağlantı bilgilerini içeren netlist formatına dönüştürülür ([5]). Sentezleme yapılırken FPGA'in sahip olduğu fiziksel özelliklere bağlı çıktılar ancak sentezlendikten sonra incelenebilir ve uygun olup olmadığına o zaman karar verilir. Bu yüzden sentezlemede uygun çözüme belli hata oranında ulaşana kadar tekrar eden yöntemler uygulanır. Fiziksel sentezlemede kullanılan bazı önemli teknikler aşağıdaki bölümlerde açıklanmıştır.

2.3.2.1. Lojik Fonksiyonları Kümeleme

Fonksiyonların kümeler şeklinde sentezlenmesi alan yönetimi ve hız açısından performanslı tasarımlar sağlar. Kümeleme yaparken kullanılan kısıtlar; bir kümede kaç fonksiyon yer alacağı, kaç girişi olacağı ve kullanılacak olan saat sinyalidir. Kümeleme için geliştirilen etkili uygulamalardan ilki VPACK yazılımıdır ([11]). VPACK en çok kullanılan girişleri tespit edip bu fonksiyonları bir küme içine almayı hedefler böylelikle birbiri ile ilişkili fonksiyonlar aynı kümede olur ve bağlantı aşamasında daha iyi sonuçlar çıkar. Bu yöntem alan yönetimi ve gecikmelere de fayda sağlar. VPACK ihtiyaç olan küme sayısını azaltmaya çalışır ([11]). VPACK'in daha gelişmiş hali olan T-VPACK amaç

olarak gecikmeleri azaltmaya çalışır ([11]). T-VPACK'in azaltmaya çalıştığı gecikme değerleri şunlardır: Mantıksal hücre gecikmesi, mantıksal kümelerin içindeki gecikmeler ve mantıksal kümeler arasındaki gecikmelerdir. Kümeler arasındaki gecikmeler kümelerin içindeki gecikmelerden fazladır. Bu yüzden T-VPACK, FPGA üzerindeki kümeler arasındaki kritik bağlantıları azaltmayı hedefler. VPACK'den farkı yaptığı hesaplamalarda ve seçimlerde gecikme parametresini de kullanmasıdır.

Yapılan gözlemlerde T-VPACK sonuçları VPACK'e göre FPGA fonksiyonların yerleştirmesi ve bağlantıların tamamlanmasından sonra daha iyi alan yönetimi sağladığını göstermiştir. T-VPACK fonksiyonların çıkışlarını olabildiğince kümeler içine sokarak ara bağlantıların daha az olmasını sağlar. Bu bağlantı aşamasında kolaylık sağlar ve daha iyi sonuçlar çıkarttığı VPR (Verilog-Placement-Routing) ile görülür. Bu algoritmada kümelerin yerlerinin değiştirilmemesi bir dezavantaj olarak görülmektedir. Kümeleme işleminde kümeye yakın fonksiyonlar işleme tabi tutulabilir ama daha iyi sonuç alınabilecek alternatif fonksiyonlar incelenip kümeler FPGA üzerinde kaydırılmaz. FPGA tasarımında şu an en yoğun kullanılan teknik kümelemedir. Kümeleme işlemi bağlantıları düzgün yapılabilen tasarımların çıkmasına katkı sağlamıştır. Bir başka araç olan VPR önceliklendirmeye dayalı bir algoritma ile seçim yaparak bağlantı için ideal bir çıktı üretir. Bazı algoritmalarda ise üretilen netlist FPGA donanıma uyana kadar deneme yapar. Bu tarz yaklaşımda, bağlantıları oluşturma adımında performans sağlanır.

Bazı çalışmalarda kümeleme çalışması bir kaç seviye için gerçekleştirilir ve daha iyi sonuçlar ortaya çıkar. Örneğin kümeleme seviyesi iki ise her küme kendi içinde ayrıca bir kümeleme işlemine tabi tutulur ve kümelerin içindeki tasarım, gecikme ve alan yönetimleri de iyileştirilir ([11]).

2.3.2.2. Yerleştirmeye Dayalı Eşleştirme

Bu sentez tekniği yerleştirme ve teknoloji eşleştirmesinin birlikte gerçekleştirildiği tekniktir. Bu teknikte teknoloji eşleme ve yerleştirme beraber incelenir. Bu yöntemin kullanımına örnek olarak MIS-Pga algoritması verilebilir. MIS-Pga tekrarlanan performans artırma ve yerleştirme algoritması kullanarak sentezleme yapar ([11]). Bazı algoritmalar eşleştirmede ağırlığı yani devredeki yoğunluğu fazla olanı seçerek sentezi gerçekleştirir. Yukarıda bahsedilen MIS-Pga algoritmasının da kullandığı tekrarlı

iyileştirmeler ve yerleştirme işleminde tekrarlar, sentezlenen devrenin ilk işlem sonrasında istenen çözümü vermemesinden dolayı yapılmaktadır. Bu nedenle istenen çözüme yakın sonuçlar alınana kadar işlemler tekrarlanır. Bu konu ile ilgili olarak Şekil 2-31'de eşleştirme ve yerleştirme akışı verilmiştir.

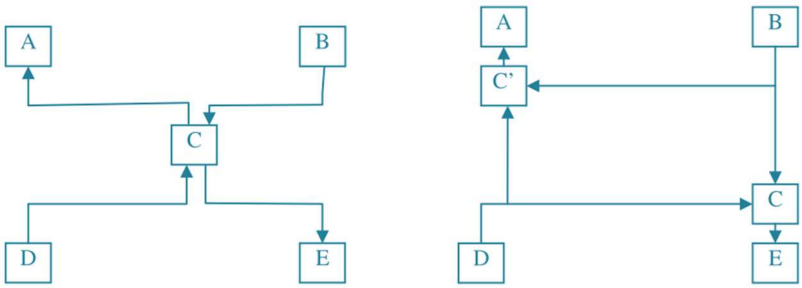
Şekil 2-31 – Eşleştirme ve Yerleştirme Akışı

Yukarıdaki akış tekrar edilir ve en iyi tasarımı bulmak için önceki işlemlerin FPGA tasarım alan miktarı, kritik yol gecikmesi, harcanan güç analizi gibi değerleri bir tabloda tutulur. Tablo değerlerine göre bir sonraki işleme gerek olup olmadığı analiz edilir ve iyileştirilmesi gereken kısımlar tablodaki değerlerden tespit edilip gerekli iyileştirme ve sentez işlemleri gerçekleştirilir. Önceki çalışmalarda sabit değerler ile işlemler yapılıyordu. İşlemler gerçekleştirilirken bir analiz yapılıp seçimler ağırlıklandırılmıyordu. Sabit değerler ile çalışılan önceki çalışmalara göre tekrarlanan ve geri beslemeli çalışan bu yeni teknikler daha iyi sonuçlar çıkarmaktadır. Yerleştirme ve eşleme algoritmalarının bazıları devreyi çizge şeklinde programlanabilen en küçük parçalara ayrıştırarak sentez işlemini gerçekleştirirler. Şekil 2-31'deki adımlarda bu işlemin benzerini görebilirsiniz.

2.3.2.3. Yerleştirmeye Dayalı Kopyalama

Gecikmelerin iyileştirilmesi için kullanılan etkili bir tekniktir. Kritik yolları izole eder, fonksiyon çıkışlardaki yoğunluğu azaltarak iyileştirme yapar. Kullanılmayan FPGA alanı mevcut ise kopyalama işlemi alan yönetiminde sorun teşkil etmez. Normal yapılan yerleştirme işleminden sonra kümeler arasındaki bağlantıları azaltmak için uygulanır. Örneğin A ve B isimli iki fonksiyon kümesi olsun ve B'deki bir fonksiyon A'dakine

bağlantılı olsun. Normal şartlarda devreyi yerleştirdiğimizde ilişki olduğu için A ve B kümesi arasında bağlantı oluşturmamız gerekmektedir. Eğer ki B'deki fonksiyonu A'ya kopyalarsak kümeler arasında bağlantı yapmamıza gerek kalmaz ve gecikmeler iyileşmiş olur. Şekil 2-32'de C fonksiyonu ilk aşamada bütün diğer kümelere eşit uzaklıkta olması için ortaya konulmuştur ama sonrasında iyileştirme için kopyalanarak yollarda iyileştirme yapılmıştır.



Şekil 2-32 – Fonksiyonların Kopyalanarak Gecikmelerin Azaltılması ([11])

2.3.2.4. Tekrarlayan Gecikme Hesaplaması ve Yerleştirme

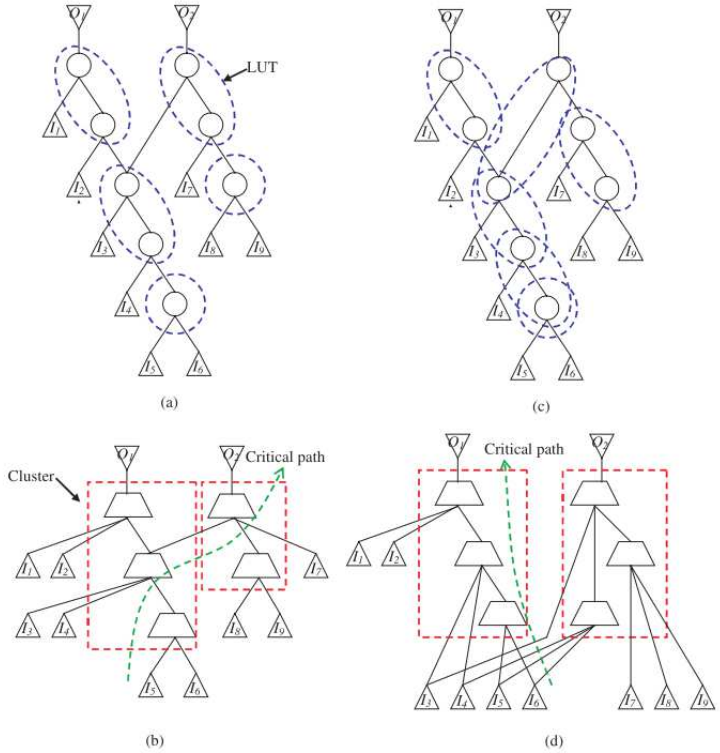
Bu yöntemde tekrarlayan şekilde, gecikmeler ve yerleştirme hesapları yapılarak uygun çözüme yakın çözüm aranır. Uygun çözüm bulununca işlem bitirilir.

2.3.2.5. SPFD Tabanlı Bağlantıların Tekrar Düzenlenmesi

SPFD (Set of Pairs of Functions to be Distinguished) yönteminde fonksiyonlara dokunmadan bağlantılar tekrarlı bir şekilde düzenlenir ve kabul edilebilir bir çözüm bulununca durulur. Buradaki çözümü belirleyen unsurların kabul edilebilirliği, kullanılan FPGA'in programlama sonrasındaki ölçülen, kullanılan programlama alanı, kullanılan güç ve gecikme gibi değişkenlerinin, devrenin kullanılacağı sistemlerin ihtiyacına uygunluğu ile ölçülür. Bu yöntemde LUT yapısının sağlamış olduğu dinamiklik kullanılmaz. Normalde k girişli bir LUT maksimum k adet girişe sahip herhangi bir fonksiyonu gerçekleştirebilir ama bu yöntemde sadece bağlantılar tekrar gözden geçirilip değiştirilir.

2.3.2.6. Eş-zamanlı Teknoloji Eşleme ve Kümeleme

Şekil 2-33'de gösterilen şekildeki gibi parçalanan çizge FPGA teknolojisine göre önce eşleştirilir ve sonra bu eşleştirilen yapılar kümelenir.



Şekil 2-33 – Eşleme ve Kümeleme ([11])

Yukarıda açıklanan yöntemler ile birlikte geçmişten bu güne kadar yapılan çalışmaları incelediğimizde sentezleme tekniklerini dört başlık altında inceleyebiliriz bunlar: (1) Yapısal Tabanlı Teknikler, (2) SAT (Boolean Satisfiability) Tabanlı Teknikler,(3) Sınıf Tabanlı Teknikler ve (4) Ayrıştırma Tabanlı Tekniklerdir ([13].)

2.3.2.7. Yapısal Tabanlı Teknikler

Yapısal tabanlı sentezleme tekniklerinin temeli lojik konilerin tekrar eşlenmesine dayanır ([12], [15]). Bu yaklaşım şu çalışmalar ile benzer eşleştirme yöntemini kullanır ([16], [17]). Yapısal tabanlı teknikler devreleri çizge eşliği üzerinden inceler. Devreler çizge olarak incelendiğinde etkili bir yöntemdir ancak LUT tabanlı mimarilerde yeteri kadar etkili değildir ([13]).

2.3.2.8. SAT (Boolean Satisfiability) Tabanlı Teknikler

SAT tabanlı teknikler mantıksal fonksiyonların bir LUT'un parçası olup olmadığını kontrol etmede etkilidirler. Bu yöntemin dezavantajı giriş sayısı ondan büyük olan fonksiyonlarda performansları düşük olmasıdır. SAT tabanlı teknikler eğer eşleştirilen

fonksiyona eş bir fonksiyon devrede varsa onu bulmayı garanti ederler ([13], [14], [18]–[20]).

2.3.2.9. Sınıflandırma Tabanlı Teknikler

Sınıflandırma tabanlı teknikler, sentezleme yaparken kullanılan sınıflandırma algoritmasının çıktısına göre farklı performanslarda çalışırlar ([21]). Sınıflandırmaya örnek olarak, NPN denkliğe göre sınıflandırma verilebilir ([8], [22]). Sınıflandırma tekniğinde karşılaştırma maliyetleri ve performans beklentisi kütüphaneyi oluşturan algoritmaya ve karşılaştırma algoritmasına bağlıdır. Bu yöntemdeki genel yaklaşıma göre sentez öncesinde genelleştirilmiş bir fonksiyon kütüphanesi oluşturulup sentezleme bu kütüphaneye göre yapılır. Ancak kütüphane büyüklüğü fonksiyon giriş sayısının artışına göre sorun oluşturacak şekilde büyümektedir. Kütüphane büyüdükçe arama ve eşleştirmede performans açısından zorluklar ortaya çıkar. Ayrıca sınıflandırma algoritması bir takım kısıtlamalar getirip istenilen fonksiyonların seçilip sentezin performanslı bir çıktı vermesini engelleyebilir ([13]). Kısıtların performansa etkisi şöyle olacaktır: Daha iyi sonuç veren bir fonksiyon kullanıcının kısıtına uymadığı için kullanılamayacaktır; kullanıcı da bütün olasılıkları deneyemeyeceğinden elde edeceği sonuçtan daha iyi sonuçlar olma olasılığı olacaktır.

2.3.2.10. Ayrıştırma Tabanlı Teknikler

Ayrıştırma tabanlı tekniklerde devre, kullanılan teknolojik alt yapıda kullanılabilecek en alt parçalara ayrıştırılır sonraki aşamada optimizasyon problemini “bin-packing” ve “set cover” problemlerine dönüştürüp ([12]) iyileştirme yapar ve sentezleme tamamlanır ([13], [21]).

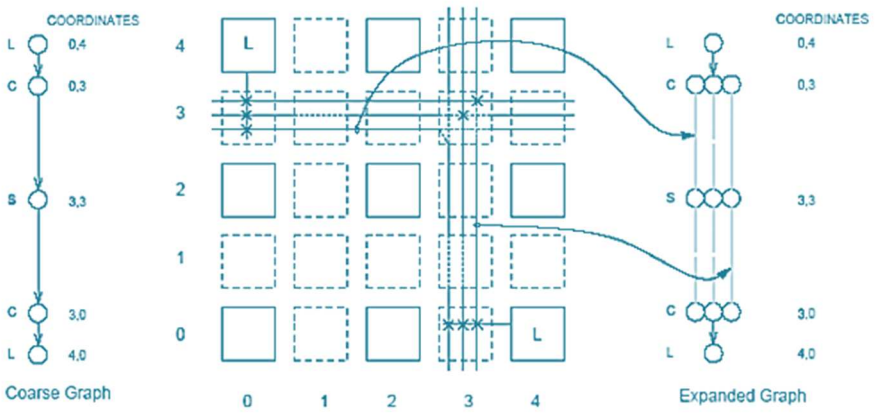
Sentezleme yaparken kullanılan tekniklerden sınıflandırma tabanlı ve ayrıştırma tabanlı teknikler etkili tekniklerdir ([13]). Özelliklede son dönemlerde sınıflandırma ile ilgili oldukça değişik çalışmalar yapılmaktadır. Bunlardan performans açısından avantajlı olduğu için birçok uygulaması olan NPN kuramı sınıflandırma tabanlı sentezlemelerde oldukça yoğun ve farklı şekillerde kullanılmaktadır ([1], [3], [13]).

2.3.3. Paketleme ve Yerleştirme

Devremizi kullandığımız FPGA teknolojisine eşleştirdikten sonra bağlamayı kolaylaştıracak ve performans beklentimizi giderecek şekilde fonksiyonları FPGA üzerine yerleştirmemiz gerekir. Bunu yaparken LUT tabanlı FPGA kullanıyorsak CLB'ler içerisine veya yakın bloklara aynı kümedeki fonksiyonlar koyulur. Yani bağlama adımında oluşturacağımız çizgede yakın düğümler FPGA üzerinde de birbirine yakın bir şekilde yerleştirilir. İlişkili fonksiyonları yakın şekilde yerleştirmek ve birbirine benzer veya yakın fonksiyonları aynı blok(CLB) içerisine koyma işlemi sonucunda son adım olan bağlama adımında daha az karmaşık bağlantı şemaları çıkacaktır ([11]). Paketleme ve yerleştirme için VPR ve TV-Pack etkili bir şekilde kullanılır.

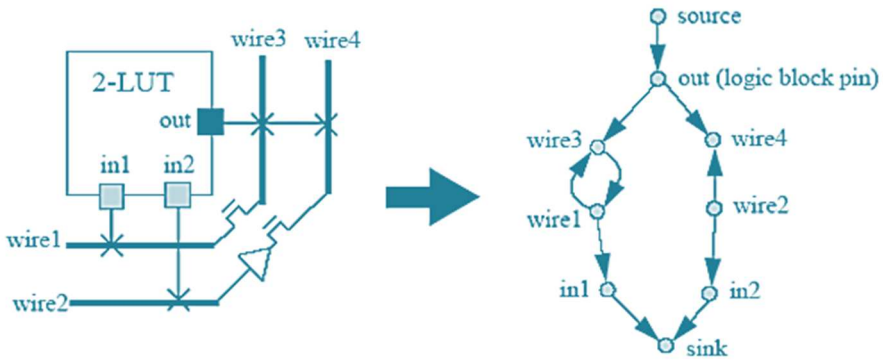
2.3.4. Bağlama

Bağlama aşaması FPGA tasarımda kritik yol üzerindeki gecikme ile ilgili belirlenen hedefleri etkileyen bir adımdır. Bir devrenin bütün elemanlarının başarılı bir şekilde bağlanabilmesi kullanılan FPGA donanımının bağlantı için ayrılan kısmıyla doğrudan ilişkilidir. Bunlar ara bağlantı kabloları, programlanabilen kondüktörler ve çoklayıcı (MUX) gibi elemanlardan oluşmaktadır. Devre tasarlanırken fonksiyonlar belli kümelere ayrıştırabilir. Eğer ki bir küme içindeki bağlantıları gerçekleştiriyorsak buna yerel bağlama denir. Kümeleri birbirine bağlıyorsak buna da global bağlama denilmektedir. Şekil 2-34'de global (sol) ve yerel (sağ) bağlamaya örnek çizgeler verilmiştir ([11]).



Şekil 2-34-Global ve Yerel Bağlama ([11])

FPGA bağlantı düzenleme işi Şekil 2-35'de gösterildiği gibi devrenin çizgesinin oluşturulması ile başlar ([11]).



Şekil 2-35-Bağlantı için Çizge Oluşturulması ([11])

Çizge üzerindeki düğümler mantıksal kapılara denk gelirken ara yollar bağlantılara ve programlanabilen kondüktörlere denk gelmektedir.

Devrelerin çizgelerini oluşturmak için VPR ve Pathfinder gibi yazılımlar kullanılmaktadır. Çizgeleri oluşturmak için şu parametrelere ihtiyaç duyulur:

a) Mantıksal blok girdi/çıktı sayıları
b) Mantıksal blok girdi ve çıktılarından aktif ve pasif olanların sayısı
c) Mantıksal blokların girdi ve çıktıları arasında eşleşme olanlar sayısı
d) Satır ve Sütun boyunca kullanılacak olan I/O blok sayısı
e) İletişim kanallarının yatay/dikey genişlikleri
f) Eğer ki FPGA melez ise farklı kısımlarındaki kanalların genişlikleri
g) Programlanabilen kondüktör yapısı ve konumlandırılmaları
h) Mantıksal blokların girdi ve çıktılarında pin başına düşen bağlantı sayısı
i) FPGA kablo kısımlarının özellikleri

Burada (a) ve (f) global bağlama için, (g) ve (i)'de yerel bağlama için gereklidir ([11]).

3. LUT SEVİYESİNDE DEVRELERDE NPN DENKLİĞİ ARTIRMAK

Bu çalışmamızda bir devre içerisindeki fonksiyonlar arasındaki NPN ilişkileri artırmak amaçlanmıştır. LUT tabanlı FPGA mimarilerinde SRAM paylaşımı kullanılarak daha az alan kullanıldığını ve bunun gecikmeler ile kullanılan güce olumlu fayda sağladığını yapılan çalışmalardan bilmekteyiz ([1], [3]).

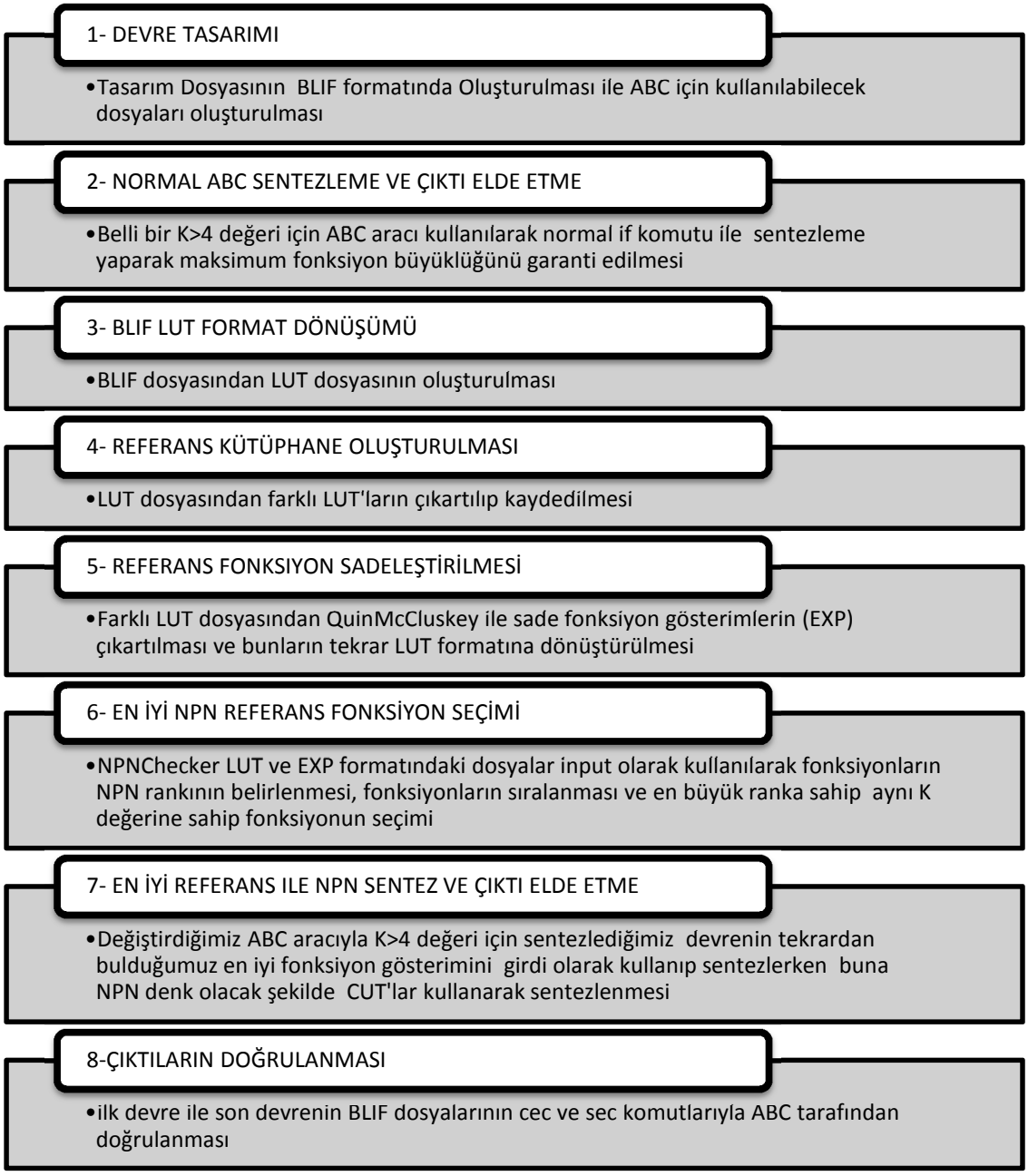
Geçmişte yapılan çalışmalarda gözlemlediğimiz yaklaşım belli bir örnek devre kümesinde kullanılan fonksiyonların NPN ilişkilerine ve kullanım yoğunluğuna göre ağırlıklandırılarak genel kullanıma yönelik fonksiyon blokları oluşturmak veya NPN ilişki kütüphaneleri oluşturularak fonksiyon seçimlerini bu kütüphane kümesinden sağlamaktır ([1], [3]). Bu yöntemler sonucunda SRAM paylaşımını artırıp iyileştirme yapılması sağlanmıştır.

Yapılan bu çalışmalar genel kullanıma yönelik çözümler sunarlar ve her bir devreye özel iyileştirme yaparken güç kullanımı, alan kullanımı ve gecikme açısından kısıtlı sonuçlar verirler. Ayrıca yapılan ölçümler belli bir kümeyi temel aldığı için daha geniş bir devre kümesinde yeterli performansı sağlayıp sağlayamadıkları bilinmemektedir.

Yapılan bu çalışmalar deneysel ölçümler sonucu oluşturulan kütüphanelerin FPGA'lere uygulanmış halidir. Bizim çalışmamız devreyi analiz eder ve o devre için sentezlemede kullanılabilecek NPN fonksiyon sınıflarını devreden oluşturup kullanır. Bu yöntem önceki NPN tabanlı yapılan çalışmalarda uygulanmış olan genelleme yaparak en çok kullanılan NPN fonksiyonların belirlenip bütün devrelere uygulanmasından daha etkili sonuçlar vermektedir.

Çalışmamız sentezleme için kullanılan ABC aracına eklenti yazarak gerçekleştirilmiştir. Ayrıca format dönüşümleri için de yazılımsal araçlar tarafımızca geliştirilmiştir. ABC aracı önceliklendirme ile sentezleme gerçekleştirirken girdi olarak aldığı devreyi ufak parçalara bölerek oluşturduğu CUT (Circuit Partitions) diye isimlendirilen devre kesitlerini kullanıcıların yazdıkları fonksiyon kısıtlarına göre kullanışlı ve kullanışsız şeklinde ayırır. Kullanışlı olarak seçilenleri devreyi oluştururken kullanır. Devre bütünlüğünü bozmamak için kullanışsız olanları da kullanabilir ama yoğun olarak kullanıcının seçtiği şarta uygun CUT'ları kullanır. Bizim geliştirdiğimiz eklenti ABC'nin sentezlemeyeceği devreden oluşturduğu ve doğruluk tablosu ile ifade ettiği CUT'ları

verdiğimiz referans fonksiyonları ile NPN denk olup olmadığını bütün özel durumlarını dikkate alarak inceler ve referans fonksiyonu ile ABC aracından gelen CUT NPN denk ise ABC'ye oluşturduğu CUT kullanışlıdır şeklinde cevap döner. Bunu yaparken çizge izomorfizm eşitlik kontrolünü yapabilen Nauty kütüphanesini kullanarak hiper çizge izomorfizm ile gerçekleştirir. Bu eklenti sayesinde devreyi LUT seviyesinde parçalara ayırıp en küçük parçaların bizim verdiğimiz devreye NPN olduğunu garanti ederiz. Yaptığımız sentezleme çalışmasının adımları Şekil 3-1'de verilmiştir.

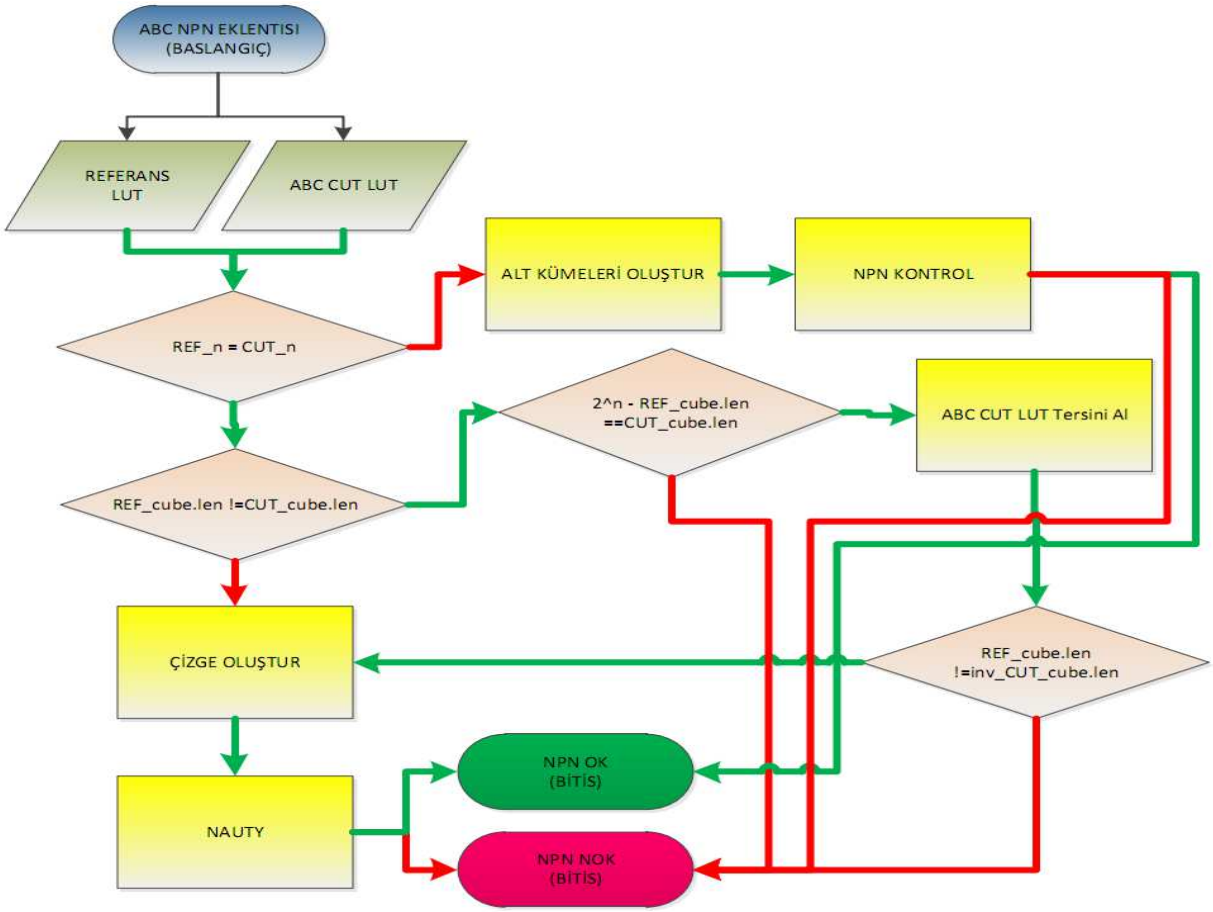


Şekil 3-1- NPN Tabanlı Sentezleme Akışı

Yukarıdaki Şekil 3-1'de verilen adımlardan birinci, ikinci ve sekizinci adım dışındaki adımlarda geliştirdiğimiz yazılımsal araçlar ile işlemler gerçekleştirilir. Birinci adımda girdi olarak BLIF(Berkeley Logic Interchange Format) formatındaki MCNC referans devre dosyaları kullanılır. İkinci adımda mevcut ABC aracının if komutu değiştirilmeden kullanılır. Son adım olan sekizinci adımda ise ABC aracındaki mevcut doğrulama komutlarından cec(combinational equality checking) ve sec(sequential equality checking) komutları kullanılarak işlemler gerçekleştirilir. Adımlarla ilgili detaylı açıklamalar ilerleyen sayfalarda mevcuttur.

Çalışmamızdaki fonksiyon seçim aşaması diğer çalışmalara göre oldukça dar bir kümeyi incelediği için daha az eşitlik karşılaştırması yapar ve daha hızlı çalışır. Çünkü incelemeye başlamadan önce BLIF dosyasından çıkarılan farklı fonksiyonlar Quin-McCluskey algoritması ile sadeleştirme işlemine sokulur ve eğer benzer fonksiyonlar varsa eşleri elenir. Böylelikle daha az farklı fonksiyon ile işlem yapılır. Bu dar kümedeki fonksiyonlar içinden kendi aralarında en çok NPN denklik bağı olanı sentezlemede referans fonksiyon olarak seçilir.

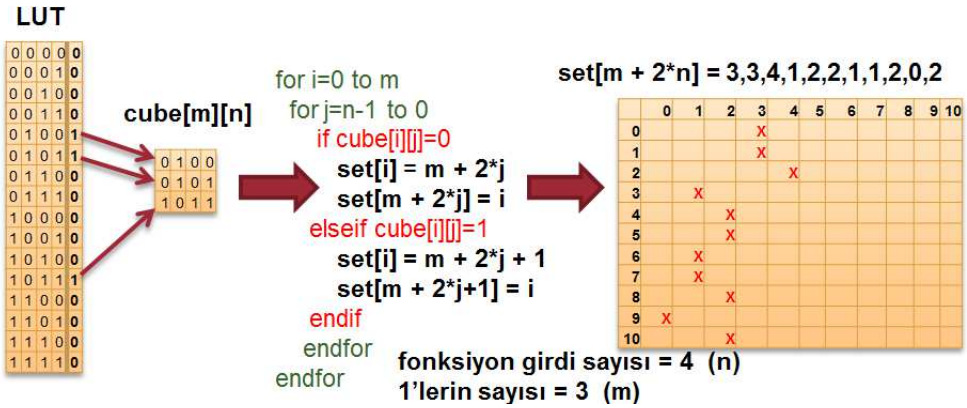
Bu aşamadan sonra yaptığımız geliştirme, ABC'ye ek kontrol fonksiyonu yazarak seçtiğimiz referans fonksiyona NPN denk olan CUT'ların kullanışlı CUT olarak işaretlenmesi sağlamaktır. Geliştirdiğimiz eklentinin fonksiyonel akış diyagramı Şekil 3-2'de gösterilmiştir. ABC eklentisine referans fonksiyonu ve ABC'den gelen fonksiyonların doğruluk tabloları (LUT) girdi olarak gelir ve bu fonksiyonların NPN olup olmadığı ile ilgili sıfır veya bir cevabı dönülerek işlem tamamlanır. Bu sıfır ve bir cevabı ABC içerisinde ABC tarafından üretilen (CUT) fonksiyonun devre sentezlemesinde kullanışlı olup olmadığına karar vermek için kullanılır. Dönen cevap bir ise bu fonksiyon (CUT) devre içerisinde kullanılabilir demektir ve devreye eklenir. Bunun dışında sıfır ise dönen cevap ABC bu fonksiyonu devreyi oluştururken kullanmaz.



Şekil 3-2-ABC Eklentisi If_CutPerformCheckNpn, Check_NPN_Equailty Fonksiyonu Akış Diyagramı

Buradaki işlemde fonksiyonları karşılaştırmak için öncelikle değişken sayılarına bakılır. Eğer ki değişken sayıları farklı ise değişken sayılarını eşitlemek için NPN yönteminde bahsettiğimiz özel durumlardan sabit atama ve köprüleme işlemleri yapılarak değişken sayıları eşitlenir. Bu şekilde oluşabilecek bütün alt kümeler için referans fonksiyonu ile NPN kontrolü yapılır ve sonuç dönülür.

Bunun dışında Şekil 3-3'deki gösterimde fonksiyon akışındaki devrelerin küp (cube) denilen doğruluk tablosundaki birlerin oluşturduğu matrise bakılarak Nauty'in kullanacağı formata dönüştürülür. Bu format her farklı fonksiyon için farklı olan çizge gösterimleri oluşturulur.



Şekil 3-3- Küp Oluşturma ve Nauty Çizge Oluşturma

Şekil 3-2'de ve Şekil 3-3'deki ifadelere bakılarak fonksiyonların karşılaştırılması için küp gösterimlerine, değişken sayılarına ve gerekirse doğruluk tablosunun tersine ihtiyaç vardır. Bu girdiler ile Nauty için çizgeler oluşturulur ve oluşturulan bu çizgeler Nauty ile karşılaştırılarak NPN denklikleri araştırılır. Yukarıdaki ifadeler Algoritma 1'de de ayrıca detaylı olarak verilmiştir.

Geliştirmemizde fonksiyonu ABC'de etkinleştirecek bir parametre tanımlanmıştır. if komutuna eklediğimiz –U parametresi ile NPN denklik fonksiyonunu kullanmasını sağlar ve –B parametresi ile de kullanacağı referans fonksiyonlarının tutulduğu dosyayı girdi olarak veririz.

Geliştirdiğimiz If_CutPerformCheckNpn fonksiyonu aşağıda görüldüğü gibi kullanılmak üzere fonksiyon referansı ile atama yapılır.

```
if ( pPars->fEnableCheckNPN )
  {
                if ( pPars->nLutSize < 2 || pPars->nLutSize > 15 )
    {
      Abc_Print( -1, "This feature only works for {2,3,4,5,6,7,8,9,10,11,12,13,14,15}-
LUTs.\n" );
return 1;
    }

pPars->pFuncCell = If_CutPerformCheckNpn;
    pPars->fCutMin = 1;
  }
```

Atama yaptığımız fonksiyon ABC içerisinde aşağıdaki şekilde görülen pFuncCell referansı ile çağrılır ve aşağıda Algoritma 1'de gösterilen eklentinin çalıştırılması sağlanır.

```
// compute the truth table
      pCut->fCompl = 0;
if ( p->pPars->fTruth )
      {
int RetValue = If_CutComputeTruth( p, pCut, pCut0, pCut1, pObj->fCompl0, pObj->fCompl1 );
         pCut->fUseless = 0;
if ( p->pPars->pFuncCell && RetValue < 2 )
         {
            assert( pCut->nLimit >= 4 && pCut->nLimit <= 16 );
pCut->fUseless = !p->pPars->pFuncCell( p, If_CutTruth(pCut), pCut->nLimit, pCut->nLeaves, p->pPars->pLutStruct );
            p->nCutsUselessAll += pCut->fUseless;
            p->nCutsUseless[pCut->nLeaves] += pCut->fUseless;
            p->nCutsCountAll++;
            p->nCutsCount[pCut->nLeaves]++;
         }

      }
```

Algoritma 1 NPN denklik kontrolü (Check_NPN_Equailty)

for i=0 to num in referans function **do**

gen.ref.functions truth tables and variable counts

gen. ref. function and abc cut truth table cube

if (cut and ref variable size not equal)

Check_Special_Case with same Nauty NPN check functionality

if (any of special case provides NPN equality)

return NPN equal

else

try another ref. function in list

```
        end if
end if
if(cut and ref cube size not equal)
İf(cut lut length – cut cube length != cube ref length)
        try another ref. function in list
else
        invert cut lut and calc. cut cube again
        if(ref. func. cube length != inverted cut lut cube length )
                try  another ref. function in list
        else
        run Nauty = true
        endif
endif
else
        run Nauty = true
end if
if(run Nauty == true)
        initiate Nauty with default options
        set get canon true
        set write automs false
        set write markers false
        create Nauty graph with ref function cube(Create_Graph)
        create Nauty graph with cut function cube(Create_Graph)
```

if (canonical representations equal)

return NPN equal

endif

endif

end for

Algoritma 1'de bahsedilen özel durumların hesaplanması Algoritma 2'de açıklanmıştır. Yukarıda gösterilen NPN denklik algoritması doğruluk tablolarından küp gösterimlerini oluşturur. Bu küp gösterimlerini çizgelere dönüştürüp çizgeler arasındaki ilişkileri Nauty ([23], [24]) ile inceleyerek benzer olup olmadıklarını araştırır.

Algoritma 2 NPN denklik kontrolü (Check_Special_Case)

---

get alphabetical variable list from ref. function expression

calc variable difference from cut truth table

for i= 0 num combination for this alphabetical representation **do**

keep static part i^{th}combination and change dynamic

calc. all dynamic part combination for bridging and same for 0 and 1 for constant assigment together

map new variables to expression

calc lut cube and to same Nauty check (Check_NPN_Equailty) without special case check control

if(Nauty check success)

return NPN equal

else

continue to try another combination

---

endif

end for

Algoritma 2'de aynı değişken sayısına sahip olmayan fonksiyonları karşılaştırma için yapılan köprüleme ve sabit atama işlemlerine yer verilmiştir. Bütün olasılıklar hesaplanıp her biri için Nauty ile denk olup olmadığına bakılıp cevap dönülür.

İlerleyen başlıklarda yukarıdaki Şekil 3-1'deki çalışmamızın adımları detaylıca anlatılarak geliştirdiğimiz uygulamalara ve algoritmalara yer verilmiştir.

3.1. Devre Tasarımı

Çalışmalarımızda MCNC referans devrelerinin BLIF ([25]) formatındaki hazır dosyaları kullanılmıştır. Bu adımda herhangi bir geliştirme yapılmamıştır.

3.2. Normal ABC Sentezleme ve Çıktı Elde Etme

Bu adımda ABC sentezleme aracının sağlamış olduğu komutlar kullanılarak BLIF ([25]) formatındaki dosyalar okunmuştur. ABC'nin –if komutunun maksimum fonksiyon giriş sayısı ile ilgili parametresi (K) kullanacağımız teknolojiye göre girdi olarak verilmiştir. Kullandığımız K parametresi ile ABC'nin devreyi, en fazla K girişli fonksiyonlar ile sentezlemesi sağlanmış ve sonuçları kaydedilmiştir. Bu adımda herhangi bir geliştirme yapmadan hali hazırda mevcut ABC özellikleri kullanılarak normal sentezleme yapılmıştır. Örnek sentezleme işlemi sırası ile aşağıda gösterildiği şekildedir:

- ABC> read_blif xxx.blif (okuma)
- ABC> if –m –K 5 (sentezleme)
- ABC> write_blif xxx_k5.blif (yazma)

ABC sentezleme yaparken devreyi en küçük parçalara ayırıp gecikme ve alan gibi unsurlara göre parçaları karşılaştırıp önceliklendirir ve K ile verdiğimiz giriş parametresine kadar eşleşme sağlanan devre kesitleriyle devreyi tekrardan oluşturur.

3.3. BLIF-LUT Format Dönüşümü

Bu adım algoritmamızda kullanmak üzere BLIF dosyalarının işlenmesiyle ilgilidir ve geliştirdiğimiz yazılımsal araçlar ile format dönüşümü içerir.

ABC sentezi sonrası oluşturduğumuz maksimum beş girişli (K=5) fonksiyonlar içeren BLIF dosyasını okuyarak uygulamamızda kullanabileceğimiz LUT-1 formatına dönüştürürüz. LUT-1 formatı aşağıda verildiği şekliyle ilerleyen kısımlarda anılacaktır. LUT-1 formatına bakarak doğruluk tablosu ve fonksiyonun BLIF dosyasında hangi kısmına denk geldiğini görülür. Bu format geliştirdiğimiz araçlara özel kullandığımız bir formattır. Herhangi özel bir standardı yoktur. Sadece doğruluk tabloları ile ilgili yazılım geliştirirken daha kolay işlem yapmak için tercih edilmiştir.

Örnek BLIF formatı gösterimi (girdi):

```
.names n25 n27 i_1_ n24

01- 1

0-0 1
```

Yukarıdaki BLIF formatından dönüştürülmüş LUT-1 formatı gösterimi (çıktı):

```
lut_name:[lut_1] output_name:[n24] var_count:[3] lut_data:[00001100]

lut_name:[lut_1] output_name:[n24] var_count:[3] lut_data:[00001101]
```

3.4. Referans Kütüphane Oluşturulması

Üçüncü adımda format dönüşümü yaptığımız ve oluşturduğumuz LUT formatındaki dosya analiz edilerek bütün farklı LUT değerleri geliştirdiğimiz yazılım ile alınmış ve yeni bir dosyaya <lut>_<giriş sayısı> formatında kaydedilmiştir. Bu yeni format LUT-2 olarak isimlendirilmiştir. Bu formatta sadece doğruluk tablosu ve fonksiyon giriş sayısı mevcuttur. Geliştirdiğimiz yazılımlarda kolaylık olması açısından bu şekilde basit bir formata dönüştürülmüştür. Bundan sonraki kısımlarda LUT-2 olarak anılan format aşağıda gösterilmiştir. LUT-2 formatında ilk kısım ikilik sistemde doğruluk tablosunu ve sonundaki rakam ise değişken sayısını ifade eder.

Örnek LUT-1 format gösterimi (girdi):

```
lut_name:[lut_1] output_name:[n24] var_count:[3] lut_data:[00001100]
```

Yukarıdaki LUT-1 formatından dönüştürülmüş LUT-2 formatı gösterimi (çıktı):

```
00001100_3
```

3.5. Referans Fonksiyon Sadeleştirilmesi

LUT-2 formatında oluşturduğumuz dosyamızı bu adımda geliştirdiğimiz yazılım ile okumuştur ve Quine-McCluskey algoritması ([26]) kullanarak geliştirdiğimiz yazılım ile sadeleştirilmiştir. Bu operasyon ile fonksiyondaki gereksiz girişler elenmiştir. Bu sadeleştirme fonksiyon seçimi ve sentez aşamasında daha az karşılaştırma yapmayı sağlar. Sadeleştirme sonrası sonrası elde ettiğimiz fonksiyonların girişi daha az olduğu için en fazla beş girişli fonksiyonlar olacak şekilde (K=5) sentezlediğimiz devredeki fonksiyonlar ile NPN denkliğine bakıldığında özel durumları (köprüleme ve sabit atama) da incelemek gerekir. Bu durum sadeleştirme sonrası oluşturulan fonksiyonlar ile ABC'nin ürettiği devre kesitlerinin/fonksiyonların (CUT) NPN denk olma olasılıklarını artırır. Bu adımda kullanılan EXPression formatında mantıksal fonksiyon gösterimi kullanılmıştır. EXP formatının örneği aşağıda gösterilmiştir.

Örnek bir fonksiyonun LUT-2 Formatı gösterimi (girdi):

```
00001100_3
```

Yukarıda LUT-2 formatındaki fonksiyonun sadeleştirilmiş EXP formatı gösterimi (ara çıktı):

```
((a&(!b)))
```

Yukarıda EXP formatı ile gösterilen sadeleştirilmiş fonksiyonun LUT-2 formatı ile gösterimi (çıktı):

```
0010_2
```

3.6. En İyi Referans NPN Fonksiyon Seçimi

Beşinci kısımda verilen NPN denklik algoritması bu adımda en iyi referans fonksiyon seçiminde kullanılmıştır. Geliştirdiğimiz NPN denklik kontrolü yapan yazılım beşinci adımda elde edilen EXP ve LUT-2 formatındaki n adet fonksiyon barından dosyaları girdi olarak alıp nxn boyutlarındaki ilişki matrisini doldurur.

İki fonksiyon arasındaki $(nx(n-1))/2$ adet ilişki NPN denklik algoritması ile incelenirken aynı zamanda geliştirdiğimiz yazılımla oluşturulan bu nxn boyutlarındaki matris analiz edilerek fonksiyonlar NPN denklik rankına göre sıralanır. Referans fonksiyonları içerisinden kendi aralarında NPN olanlardan rankı küçük olanlar elenir ve en büyük ranka sahip, mimaride kullanılan K girişli fonksiyon referans fonksiyonu olarak seçilir.

Sentezlemede tek bir referans fonksiyonun girdi olarak seçilmesindeki amaç devrede bulunan NPN denklik küme sayısını azaltmaktır. NPN denklik küme sayısı azalırsa devre homojenliği artar ve SRAM paylaşımı da bunun paralelinde artar. Yaptığımız çalışmada sentezleme adımında sadece bir referans fonksiyonu girdi olarak kullanıldığında MCNC referans devrelerinin genelinde büyüme görülmüştür. Bu durumda devrelerin büyümesini engellemek için birden fazla fonksiyonu referans fonksiyon olarak sentezlemede kullanmak gerekmiştir. Buna karşın birden fazla fonksiyonu sentezlemede referans fonksiyonu olarak kullandığımızda devrede birbirine NPN denk fonksiyon sayısının genelde azaldığı görülmüştür. Bu şekilde en uygun sonucu veren referans fonksiyonları sayısı tespit edilip devre sentezlemesi yapılmıştır.

Bu adımdaki çıktı dosyasında fonksiyonlar EXP formatındadır. ABC'de yaptığımız eklenti EXP formatında parametre aldığı için bu şekilde kullanılmaktadır.

3.7. En İyi Referans İle NPN Sentezleme ve Çıktı Elde Etme

ABC if komutu ile sentezleme yaparken oluşturduğu CUT'ların (kullanılan teknolojiye eşleşen en küçük devre kesiti) kullanışlı olup olmadığı tespitinde ek kontrol fonksiyonları kullanabilmektedir. Bu fonksiyonları kullanmak için parametre ile aktif hale getirmek gerekmektedir. Bunu kullanarak çalışmamızın amacına uygun parametre ile aktif hale gelen bir eklenti tarafımızca geliştirilmiştir. Bu eklenti ABC'den girdi olarak gelen CUT'ların girdi olarak verdiğimiz dosyadaki EXP formatındaki referans fonksiyonuna NPN denk olup olmadığını inceler ve girdiler birbirine NPN denk ise kullanışlı değilse kullanışsız şeklinde cevap döner.

Bu adımdaki sentezleme işlemi şu şekilde sırası ile gerçekleştirilir. Öncelikle ABC ile normal sentez ile sentezlediğimiz devremizi okuruz:

```
ABC> read_blif  xxx_k5.blif
```

Sonrasında sentezleme için geliştirdiğimiz eklentiyi aktif edecek parametreleri if komutu ile kullanırız:

```
ABC> if –m -v –K 5 –U –B exp_file.txt
```

Exp_file.txt içeriğinde ((a&(!b))) gibi EXP formatında ifadeler barındırmaktadır.ABC if komutuna –U parametresi eklenmesi NPN denklik kontrolü aktif hale gelir. Sonrasındaki –B parametresi ile de kullandığımız referans fonksiyonu barındıran dosyayı yolu girdi olarak verilir.

Sentezleme tamamlandıktan sonra write_blif ile yeni sentezlenmiş devre BLIF formatında kaydedilir.

```
ABC>write_blif  xxx_k5_npn.blif
```

Bu aşamaya kadar ki işlemler tamamlandığında sentezleme işlemini tamamlamış olur. Bundan sonraki adımda sentezlenen devrenin doğrulu kontrol edilir.

3.8. Çıktıların Doğrulanması

ABC'nin barındırdığı cec ve sec komutlarını ile sentezlediğimiz devrenin orjinali ile aynı olup olmadığını aşağıda gösterilen örnekteki gibi kontrol ederiz.

```
ABC>cec xxx_k5_npn.blif   xxx.blif
```

veya

```
ABC>sec xxx_k5_npn.blif   xxx.blif
```

Doğrulamadan başarılı bir şekilde geçen yeni devrel ile sentezlemeyi başarılı bir şekilde bitirmiş oluruz.

4. DENEY SONUÇLARI

Yaptığımız sentezleme çalışması sonucunda beklenilen sonuç SRAM paylaşımının artması ve devrenin büyüklüğünün korunup devredeki kullanılan alan, gecikme ve güçte kazanç sağlamaktır. Bunu incelemek için elde etmek istediğimiz veriler şunlardır.

1. Normal ve NPN sentez arasındaki NPN yoğunluk farkı ile NPN ilişkili çift artışının analizi.
2. Normal ve NPN sentez arasındaki devrelerdeki fonksiyon artış farkı ile devrelerin sentez sonrası alan olarak ne kadar büyüdüğünün kontrolü.
3. Sentezlemede kullanılan devre ile ilişkili referans fonksiyon sayısının NPN yoğunluğa etkisi.
4. Çalışmamızda gerçekleştirdiğimiz normal ve NPN sentezleme ile elde ettiğimiz devrelerdeki fonksiyonlardan NPN ilişkili çiftler gruplanır. Bu şekilde gruplandığında elde edilen NPN küme sayılarının değişimi.
5. Sentez sonucu elde edilen NPN küme elemanlarının girişlerine göre dağılımları. Dağılımların analizi kullanılacak FPGA teknolojisiyle ilgilidir.
6. Devrelerdeki fonksiyonların girişlerine göre dağılımları.
7. Sentezleme için geçen sürelerin analizi.

4.1. Tanımlar

2.4.1. NPN Rank

Tablolarda ve deney sonuçlarında yer alan “NPNRank” ifadesi NPN denklik ilişki sayısını ifade etmektedir. Bir fonksiyonun rank değeri o fonksiyonun bulunduğu devrede kaç adet fonksiyonla NPN denklik ilişkisi olduğunu gösterir.

2.4.2. NPN Yoğunluk

Çalışmamızda devredeki NPN ilişki sayısının devredeki fonksiyon çiftleri sayısına oranını NPN yoğunluğu diye isimlendirilmiştir. NPN Yoğunluğu şu şekilde hesaplanır:

C = Devredeki Toplam Fonksiyon Sayısı

D = NPN Rank (Fonksiyonların kendi aralarındaki ilişkiler de dahil olmak üzere devredeki toplam NPN ilişki sayısıdır.)

NPN Yoğunluk $= \frac{(D-C)}{(\mathrm{C}*(\mathrm{C}-1))}$

2.4.3. Referans Fonksiyonlar

NPN sentezlemede kullanılan girdilerden biridir. Sentezlenecek devre içerisinde diğer fonksiyonların sadeleştirme sonrasında kendi aralarındaki NPN ilişkiler incelenir ve NPN ilişki sayısına göre bu fonksiyonlar sıralanır. En çok NPN ilişkisine sahip fonksiyon referans fonksiyon olarak yeniden sentezleme işleminde kullanılır. Çalışmamızda en çok ilişkiye sahip fonksiyon dışında iki adet devre (s298_k5 ve alu4_k5) için sırası ile birden fazla referans fonksiyonu da denenmiştir.

2.4.4. Normal ve NPN Sentez

Normal ve NPN sentez diye söz konusu olan durumlar şunlardır. Normal Sentez: ABC sentez aracının –if komutu normal parametreleri ile en fazla beş girişli fonksiyonlar ile devreyi oluşturacak şekilde (K=5) kullanıldığında ortaya çıkan sentezlenmiş devreden hesaplanan değerlerdir. NPN Sentez: ABC için geliştirdiğimiz NPN eklentisi kullanılarak yapılan sentezleme sonucu hesaplanan değerlerdir.

4.2. Sonuç Verileri ve Açıklamaları

Tablo 4-1'de BLIF formatındaki devrelerin kaç girişi ve kaç çıkışı olduğu bilgisi ile devrede kaç adet doğruluk tablosu gösterilmiştir. Tablo 4-1'de verilen değerler sentez öncesi orijinal devre parametreleridir.

Tablo 4-1-Referans Devre Listesi

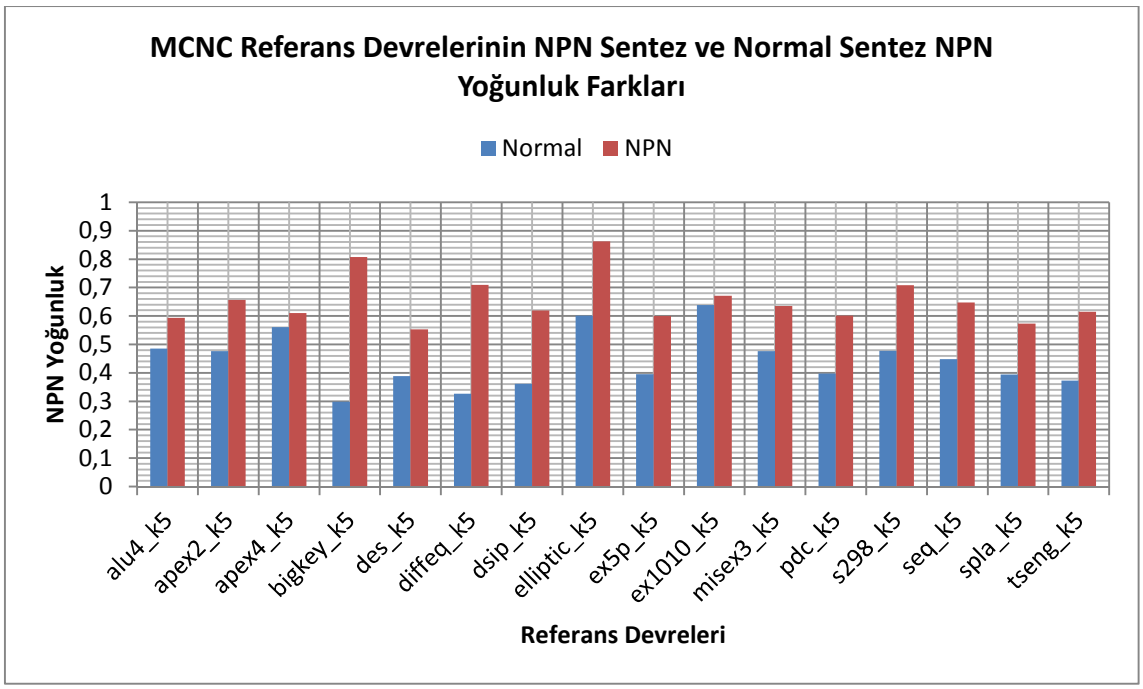
	BLIF Dosyası	Girdi Sayısı	Çıktı Sayısı	Fonksiyon Sayısı
1	alu4	14	8	1522
2	apex2	39	3	1878
3	apex4	9	19	1262
4	bigkey	263	197	1707
5	clma	383	82	8381
6	des	256	245	1591
7	diffeq	64	39	1494
8	dsip	229	197	1370
9	elliptic	131	114	3602
10	ex5p	8	63	1064
11	ex1010	10	10	4598
12	frisc	20	116	3539
13	misex3	14	14	1397
14	pdc	16	40	4575
15	s298	4	6	1930
16	s38417	29	106	6096
17	s38584.1	39	304	6281
18	seq	41	35	1750
19	spla	16	46	3690
20	tseng	52	122	1046

Tablo 4-2'deki alu4_k5 devresi $n = 1041$ adet fonksiyon içerir ve $(nx(n-1))/2$ den 541320 adet toplam fonksiyon ilişkisi olduğu görülür. Bunların içerisinde NPN denk olan ilişkilerin sayısı hesaplamalar sonucu Tablo 4-2'de bulunan 527007 değeridir. Sonrasında ilişki yoğunluğunu hesaplayıp NPN sentezleme öncesi ve sonrasında karşılaştırma yapılmıştır. Tablo 4-2'deki NPN artış miktarı Normal ve NPN Sentez sonucunda hesaplanan NPN yoğunluk miktarı ile hesaplanmaktadır. Şekil 4-1'de NPN yoğunluk miktarları normal ve NPN sentez işleminden geçen devreler için verilmiştir. Devamındaki Şekil 4-2'de ise NPN sentezin normal senteze göre NPN yoğunluğunun ne kadar artırdığı grafiksel olarak gösterilmiştir. Analizi pratik olarak tamamlanamayan devre verileri Tablo 4-2'de çarpı ile ifade edilmiştir. Yaptığımız çalışmada MCNC

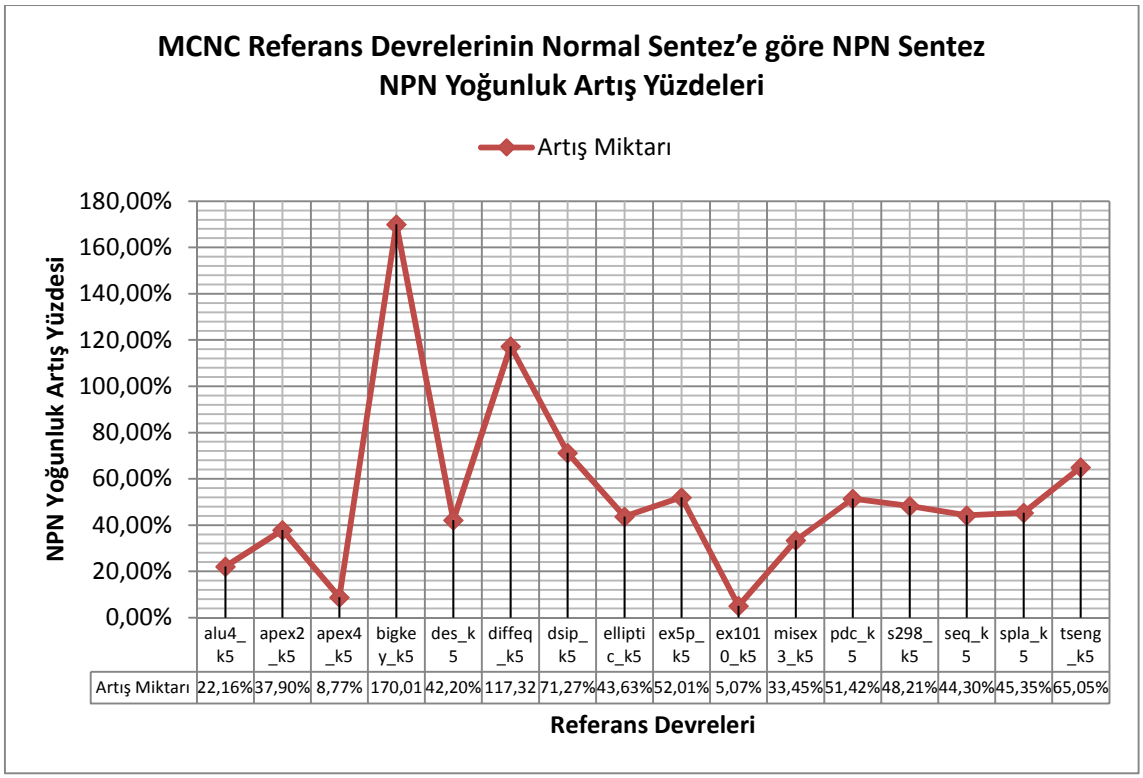
referans devrelerinde NPN yoğunluğunu başarılı bir artırılmıştır. Her bir devre için Tablo 4-2'deki NPN Artış kolonunda bilgiler verilmiştir.

Tablo 4-2- MCNC Referans Devreleri NPN Yoğunluk Tablosu (K=5)

	NORMAL Sentez		NPN Sentez		NORMAL	NPN	
Devre Adı	Fonksiyon Sayısı	NPN Rank	Fonksiyon Sayısı	NPN Rank	NPN Yoğunluk	NPN Yoğunluk	NPN Artış (%)
alu4_k5	1041	527007	1160	799037	0,485818	0,593464	22,15772012
apex2_k5	1332	845678	1481	1441057	0,476253	0,656777	37,90488605
apex4_k5	1067	639143	1313	1052484	0,560984	0,610204	8,773730704
bigkey_k5	2511	1887163	2409	4686113	0,299027	0,807414	170,0134654
clma_k5							
des_k5	2217	1912873	2892	4626507	0,388908	0,553013	42,19627851
diffeq_k5	1523	757944	2027	2914391	0,326324	0,709173	117,3218747
dsip_k5	2020	1478006	2696	4506152	0,361905	0,619822	71,26661602
elliptic_k5	551	182653	639	352494	0,600898	0,863062	43,62887631
ex5p_k5	700	193889	879	464081	0,394827	0,600188	52,01269417
ex1010_k5	1307	1090939	1646	1817772	0,638353	0,670734	5,072522423
frisc_k5	4829	8618329	5557		0,36945		
misex3_k5	985	462182	1089	753439	0,475834	0,634985	33,44675128
pdc_k5	3194	4054118	3929	9286092	0,39721	0,601445	51,41727205
s298_k5	62	1868	75	4003	0,477525	0,707748	48,21162691
s38417_k5							
s38584.1_k5							
seq_k5	1406	887718	1602	1662098	0,448668	0,647417	44,29756605
spla_k5	3115	3825367	3705	7863469	0,394042	0,572731	45,34753081
tseng_k5	1865	1297342	2149	2841308	0,372653	0,615062	65,04944167



Şekil 4-1 -MCNC Referans Devrelerinin NPN Sentez ve Normal Sentez NPN Yoğunluk Farkları

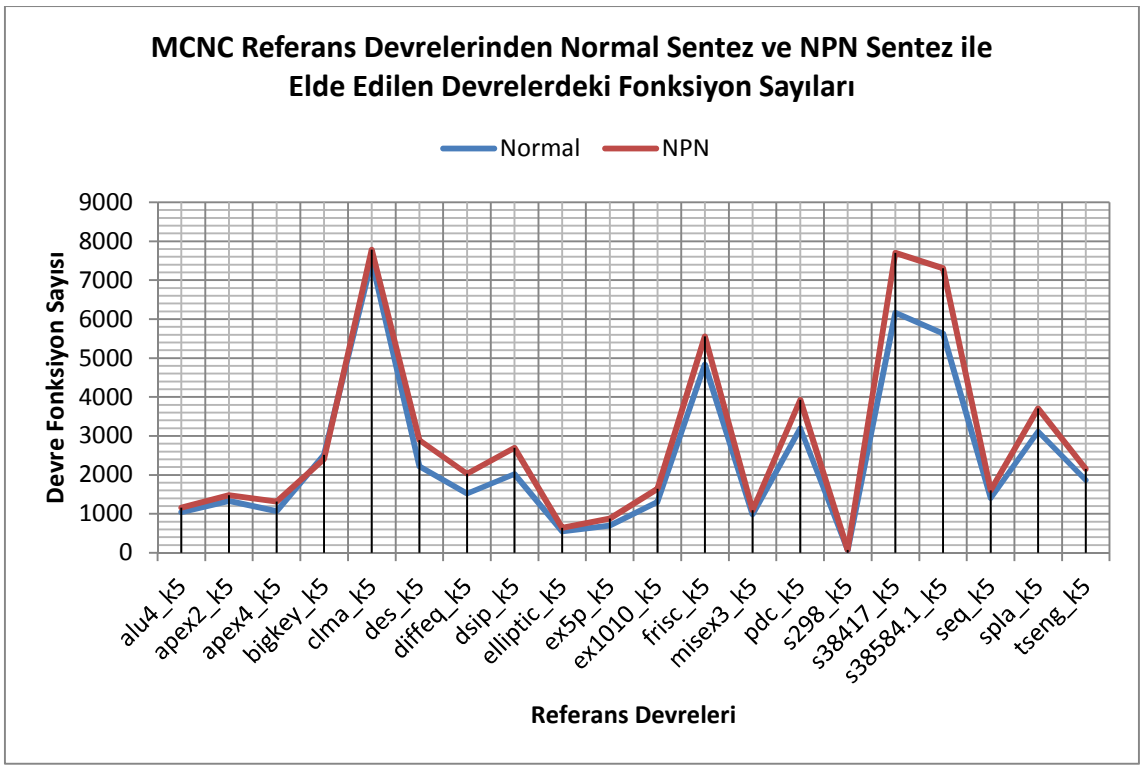


Şekil 4-2 - MCNC Referans Devrelerinin Normal Senteze göre NPN Sentez NPN Yoğunluk Artış Yüzdeleri

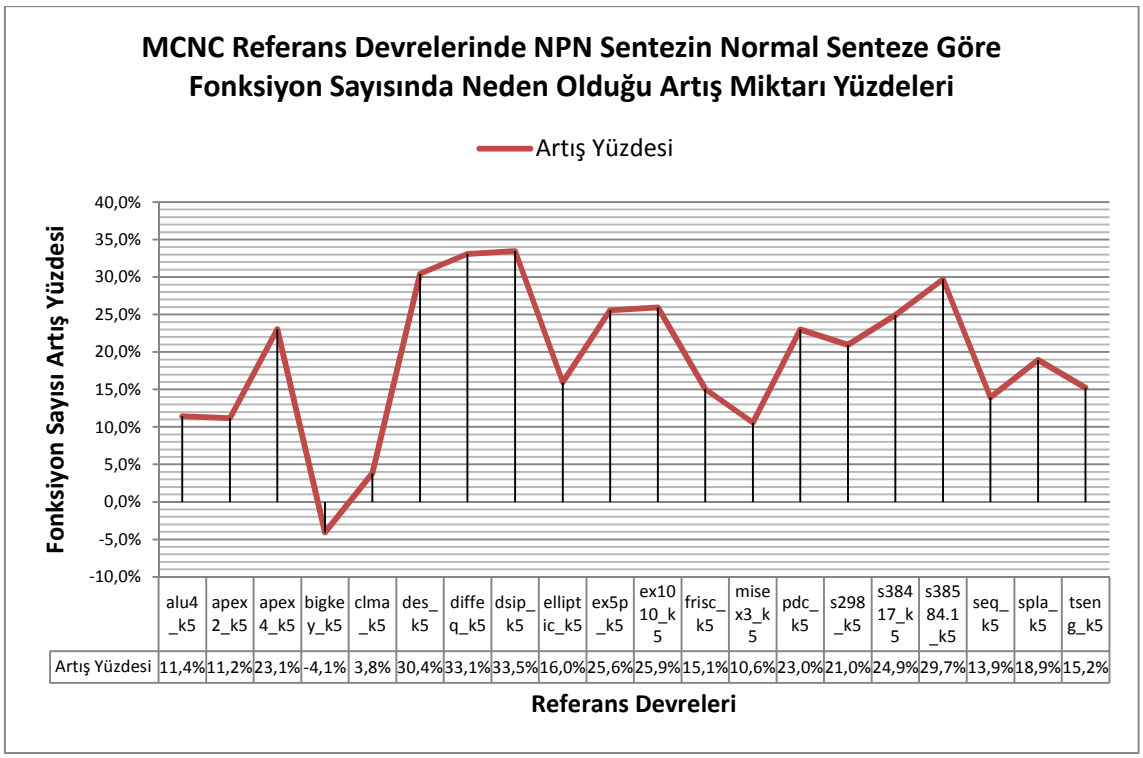
NPN yoğunluğu artırılırken fonksiyon sayısındaki artışa da dikkat etmek gerekir. Tablo 4-3'de sentezleme ile devrelerdeki fonksiyon sayılarındaki değişiklikler verilmiştir ve Şekil 4-3 ile Şekil 4-4'de grafiksel olarak ifade edilmiştir. Burada orijinal devreler ilk aşamada normal bir şekilde ABC ile K=5 için sentezlendikten sonra NPN eklentisi ile tekrar ABC ile sentezlenmiştir ve elde edilen fonksiyon sayıları, normal ve NPN sentezleme için tabloya yazılmıştır. Aşağıdaki MCNC referans devreleri için yapılan analiz sonuçlarından elde edilen tablo ve şekillerde görüldüğü üzere fonksiyon sayısında artış vardır. Her bir devre için artış miktarı Tablo 4-3'de verilmiştir. Tablo 4-3'de gösterilen devrelerdeki fonksiyon sayılarının artış miktarları çok gibi görünse de SRAM paylaşım seviyesi artırılarak kazanç sağlamak mümkündür. Bu sonuçlar en iyi referans fonksiyonu senteze girdi olarak verildiğinde elde edilmektedir.

Tablo 4-3-MCNC Referans Devreleri NPN Sentez Lut Artış Miktarları (K=5)

Devre Adı	Normal Sentez Fonksiyon Sayısı	NPN Sentez Fonksiyon Sayısı	Fonksiyon Sayısı Artış Yüzdesi (%)
alu4_k5	1041	1160	11,43131604
apex2_k5	1332	1481	11,18618619
apex4_k5	1067	1313	23,05529522
bigkey_k5	2511	2409	-4,062126643
clma_k5	7500	7786	3,813333333
des_k5	2217	2892	30,44654939
diffeq_k5	1523	2027	33,09258043
dsip_k5	2020	2696	33,46534653
elliptic_k5	551	639	15,97096189
ex5p_k5	700	879	25,57142857
ex1010_k5	1307	1646	25,9372609
frisc_k5	4829	5557	15,07558501
misex3_k5	985	1089	10,55837563
pdc_k5	3194	3929	23,01189731
s298_k5	62	75	20,96774194
s38417_k5	6166	7702	24,91080117
s38584.1_k5	5630	7303	29,71580817
seq_k5	1406	1602	13,94025605
spla_k5	3115	3705	18,94060995
tseng_k5	1865	2149	15,22788204



Şekil 4-3-MCNC Referans Devrelerinden Normal Sentez ve NPN Sentez ile Elde Edilen Devrelerdeki Fonksiyon Sayıları



Şekil 4-4- MCNC Referans Devrelerinde NPN Sentezin Normal Senteze Göre Fonksiyon Sayısında Neden Olduğu Artış Miktarı Yüzdeleri

Tablo 4-3'de tek bir referans fonksiyonu verildiğinde elde edilen sonuçlar gösterilmiştir. Kullanılan referans fonksiyonu sayısının sentezleme sonucunda çıkan devredeki fonksiyon sayısındaki değişime etkisini incelemek için bulduğumuz referans fonksiyonları farklı sayıda NPN sentezlemede kullanılarak sonuçları iki devre için Tablo 4-4 ve Tablo 4-5'de gösterilmiştir. NPN Sentez öncesi fonksiyon sayısı örnek olarak "normal = 62" şeklinde tabloda verilmiştir. Yukarıdaki Tablo 4-3 sonuçlarının dışında birden fazla NPN referans fonksiyonu verildiğinde sonuçlarda fazla bir değişiklik olmadığı Tablo 4-4 ve Tablo 4-5'de gösterilmiştir.

Tablo 4-4 - Farklı Referans Fonksiyon Sayısı ile Sentez (s298_k5)

Devre Adı	Kullanılan Ref. Fonk. Sayısı	Sentez Sonrası Fonk. Sayısı
s298_k5	1	75
Normal = 62	2	72
	3	75
	4	72
	5	72
	6	72
	7	72
	8	72
	21	61

s298_k5 devresi için toplam yirmi bir adet beş girişli referans fonksiyon mevcuttur. Tablo 4-4'deki yirmi bir değeri bunu ifade etmektedir.

Tablo 4-5 - Farklı Referans Fonk. Sayısı ile Sentez (alu4_k5)

Devre Adı	Kullanılan Ref. Fonk. Sayısı	Sentez Sonrası Fonk. Sayısı
alu4_k5	1	1160
Normal = 1041	2	1174
	3	1146
	4	1142
	5	1140
	6	1144

Alu4_k5 devresi için toplam altı adet beş girişli referans fonksiyon mevcuttur. Tablo 4-5'deki altı değeri bunu ifade etmektedir.

Tablo 4-3'deki NPN yoğunluğundaki artışı elde etmemizi sağlayan referans fonksiyonları da Tablo 4-6'de verilmiştir. Bu referans fonksiyonların nasıl bulunduğu 3.2, 3.3, 3.4, 3.5 ve 3.6 numaralı bölümlerde sırası ile anlatılmıştır.

Tablo 4-6'ya bakıldığında beş girişli fonksiyonlar (K=5) için sentezlenmiş MCNC referans devrelerin listesini ve sentez sonucu yapılan analizde bulunan en iyi referans fonksiyonları görebilirsiniz. Devredeki farklı fonksiyonları alıp sadeleştirme işlemine tabi tuttuğumuzda elde ettiğimiz yeni farklı fonksiyonlar "hesaplanan küme sayısı" olarak verilmiştir. Burada NPN rank değeri de bu referans fonksiyonunun bu kümede kaç fonksiyonun NPN denk olduğunu göstermektedir. Referans fonksiyon hesaplama süresi de ayrıca her bir devre için verilmiştir.

Tablo 4-6-NPN Referans Fonksiyonları (K=5)

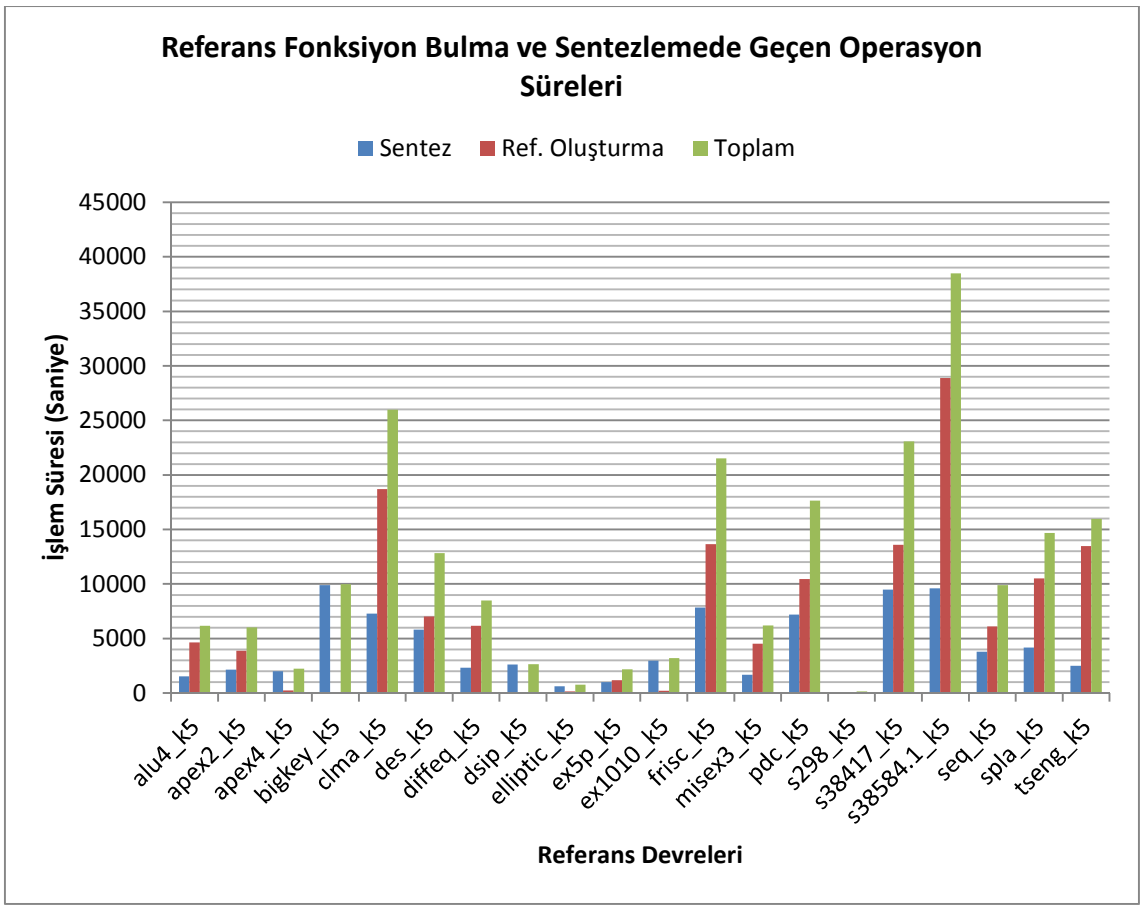
	Dosya İsmi (K=5)	En iyi NPN Referans Fonksiyon (K=5)	NPN Rank	Hesaplanan Küme Sayısı	Hesaplama Süresi Gün/Saat: Dakika: Saniye
1	alu4	(((!a)&(!b)&(!c)&(!d))|((!a)&(!d)&(!e))|((!a)&b&c&(!e)))	162	330	0/1:17:18
2	apex2	((a&(!b)&e)|(a&(!b)&c)|(a&d&e))	189	433	0/1:4:40
3	apex4	(((!a)&(!e))|((!a)&c&(!d))|((!a)&b&d))	36	129	0/0:3:51
4	bigkey	(((!a)&(!b)&(!c)&(!d)&e)|((!a)&b&(!c)&(!d)&(!e))|((!a)&b&c&d&e))	13	44	0/0:1:2
5	clma	((b&(!e))|(a&(!b)&(!c)&(!d)&e)|(a&d&(!e))|(a&c&(!e)))	431	937	0/5:11:38
6	des	(((!a)&(!b)&(!c)&(!d)&(!e))|((!a)&c&d&(!e))|(a&(!b)&c&(!d)&(!e)))	155	430	0/1:56:48
7	diffeq	((a&d&(!e))|(a&(!b)&c&d))	103	305	0/1:42:39
8	dsip	(((!a)&b&(!d)&e)|(a&(!b)&(!c)&(!d)&e)|(a&b&c&d))	15	37	0/0:0:31
9	elliptic	(((!b)&c&d&(!e))|(a&c&(!d)&(!e)))	43	83	0/0:2:29
10	ex5p	(((!c)&(!d)&(!e))|(b&c)|a)	59	175	0/0:19:31
11	ex1010	((c&d&e)|(a&c&d)|(a&b&c))	54	113	0/0:3:33
12	frisc	((a&(!c)&e)|(a&(!b)&c&(!e))|(a&c&(!d)&(!e))|(a&b&d&e))	174	468	0/3:47:37
13	misex3	((a&(!b)&(!c))|(a&(!b)&d)|(a&d&e))	139	297	0/1:15:25
14	pdc	(((!a)&c&d&e)|((!a)&b&(!d))|((!a)&b&e)|((!a)&b&c))	199	549	0/2:54:11
15	s298	(((!a)&(!b)&c&d)|((!c)&(!d)&(!e))|((!b)&(!d)&(!e)))	21	51	0/0:1:16
16	s38417	((a&(!c)&(!d)&e)|(a&b&c&(!d))|(a&b&c&e))	163	463	0/3:46:31
17	s38584.1	(((!a)&b&(!e))|(a&(!b)&(!d))|(a&b&(!c)))	253	1068	0/8:1:18
18	seq	(((!a)&c&(!e))|((!a)&(!b)&c&d)|((!a)&b&(!d)&(!e))|((!a)&b&c&(!d)))	178	415	0/1:41:53
19	spla	((b&c&d)|(a&e)|(a&b&c))	221	549	0/2:55:5
20	tseng	((a&c&e)|(a&b&(!c)&(!e))|(a&b&d))	120	469	0/3:44:30

Tablo 4-7'de NPN sentezleme işleminin ne kadar sürdüğü, referans fonksiyon seçiminin ne kadar sürdüğü ve toplamda ne kadar sürede işlemi tamamladığımıza dair veriler ve ortalama süreler verilmektedir.

MCNC referans devrelerinin sentezlemesinin geneline bakıldığında sentezleme için gerekli referans fonksiyon bulunması ve sonra bu referans fonksiyonlar devrenin sentezlemesi yirmi devre temel alındığında ortalama 3,1684 saat sürmektedir. Bu sürenin 1,1752 saatini sentezleme alırken, 1,9931 saatini de referans fonksiyon bulunması almaktadır. Diğer işlemler bir kaç saniyede gerçekleştiği için işlem süreleri hesaba katılmamıştır. Her bir devrenin sentezleme süresi ve fonksiyon bulma süresi Tablo 4-7'de verilmiştir ve Şekil 4-5'de grafiksel olarak gösterilmiştir.

Tablo 4-7 –MCNC Referans Devreleri için Sentez ve Referans Fonksiyon Bulma Süreleri (K=5)

Devre Adı	Sentez Süresi (saniye)	Ref.Fonk. Hesap Süresi (saniye)	Toplam Sentez Süresi (saniye)
alu4_k5	1528,41	4638	6166,41
apex2_k5	2141,16	3880	6021,16
apex4_k5	2006,75	231	2237,75
bigkey_k5	9910,15	62	9972,15
clma_k5	7293,04	18698	25991,04
des_k5	5810,55	7008	12818,55
diffeq_k5	2330,23	6159	8489,23
dsip_k5	2604,26	31	2635,26
elliptic_k5	626,38	149	775,38
ex5p_k5	1007,58	1171	2178,58
ex1010_k5	2981,72	213	3194,72
frisc_k5	7846,98	13657	21503,98
misex3_k5	1677,09	4525	6202,09
pdc_k5	7193,23	10451	17644,23
s298_k5	79,49	76	155,49
s38417_k5	9493,59	13591	23084,59
s38584.1_k5	9615,82	28878	38493,82
seq_k5	3783,36	6113	9896,36
spla_k5	4181,21	10505	14686,21
tseng_k5	2507,73	13470	15977,73
Ortalama Değerler	4230,94	7175,3	11406,24



Şekil 4-5-Referans Fonksiyon Bulma ve Sentezlemede Geçen Operasyon Süreleri

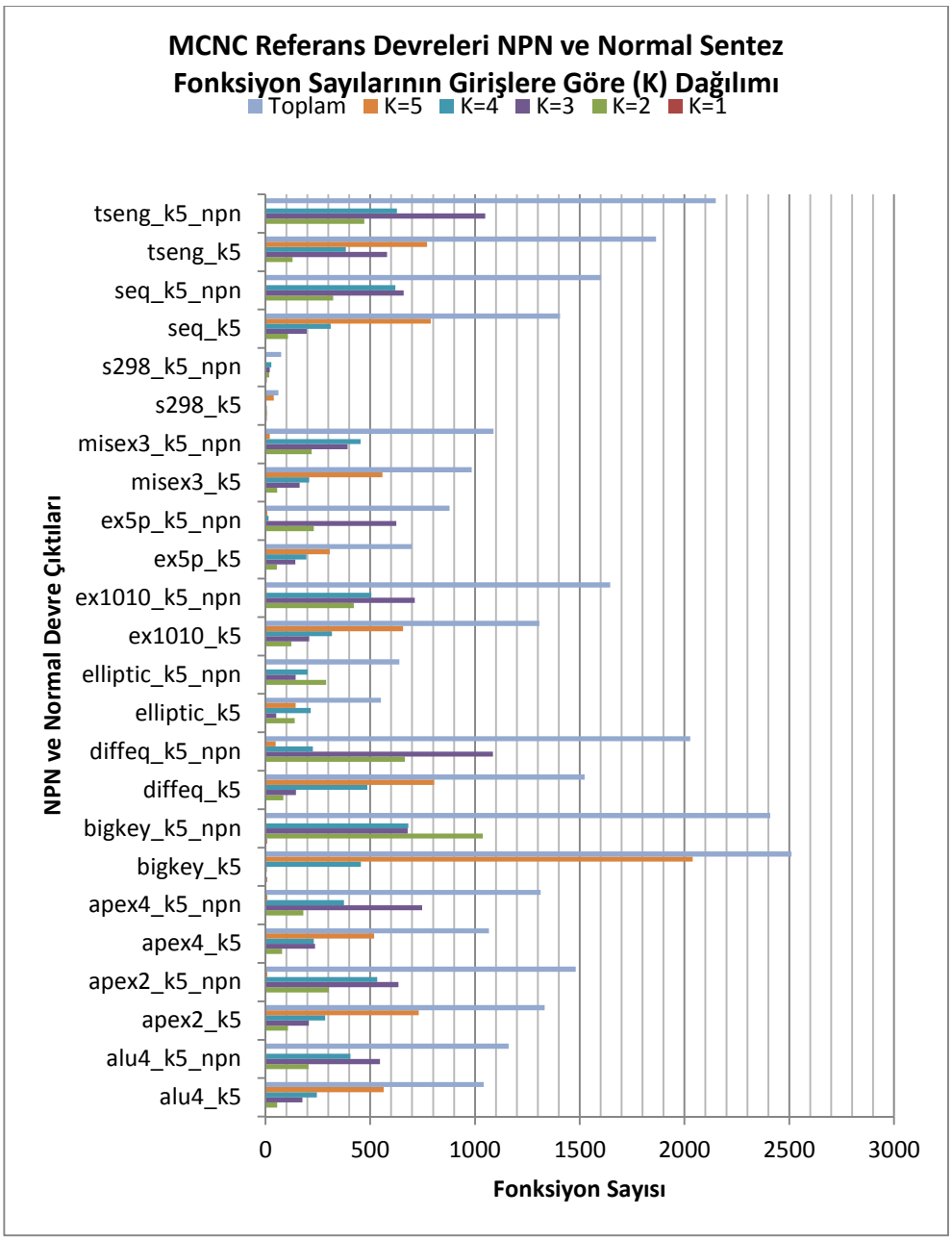
Sentez operasyonlarındaki 3,16 saatlik süre tek seferlik programlanıp fabrikasyonu yapılacak bir devrenin bir kereye mahsus analizi için kabul edilebilir bir süredir.

MCNC referans devreleri için Tablo 4-8'de Normal sentezleme ve NPN sentezleme ile elde edilen devrelerdeki fonksiyonların girişlerine göre dağılımları verilmiştir. Örneğin alu4_k5 için iki girişli elli altı adet fonksiyon üç girişli yüz yetmiş altı adet fonksiyon, dört girişli iki yüz kırk beş adet fonksiyon ve beş girişli beş yüz atmış dört adet fonksiyon vardır. Hemen altında NPN sentezi sonucu elde edilen değerler gösterilmiştir. Alu4_k5_npn için iki girişli iki yüz beş adet fonksiyon mevcuttur ve fonksiyon sayısı elli altıdan ikiyüz beşe çıkmıştır. Aynı devrede işlem öncesinde üç girişli yüz yetmiş altı adet fonksiyon mevcuttur. İşlem sonrasında bu rakam beş yüz kırk altıya yükselmiştir. Dört girişli fonksiyon sayısına baktığımızda ise iki yüz kırk beşten dört yüz altıya çıktığı tablodan görülmektedir. Bu sonuçlardan sadece beş girişli fonksiyon sayısı azalmış ve beş girişli fonksiyon sayısı beş yüz atmış dörtten üçe düşmüştür.

Tabloya bakıldığında sentez öncesi ve sonrası toplam fonksiyon sayısı da görülmektedir. Bu şekilde devrelerdeki fonksiyon değişimleri Tablo 4-8'de bütün kullanılan devreler için verilmiştir ve Şekil 4-6'da grafiksel olarak gösterilmiştir.

Tablo 4-8 –MCNC Referans Devreleri Fonksiyon Sayıları ve Giriş Sayılarına Göre Dağılımları(K=5)

Devre Adı	K=0	K=1	K=2	K=3	K=4	K=5	Toplam Fonksiyon Sayısı
alu4_k5	0	0	56	176	245	564	1041
alu4_k5_npn	0	0	205	546	406	3	1160
apex2_k5	0	0	108	207	285	732	1332
apex2_k5_npn	0	0	303	635	534	9	1481
apex4_k5	1	0	80	237	231	518	1067
apex4_k5_npn	1	0	181	748	374	9	1313
bigkey_k5	0	8	2	6	456	2039	2511
bigkey_k5_npn	0	8	1037	679	682	3	2409
diffeq_k5	0	0	85	146	487	805	1523
diffeq_k5_npn	0	0	666	1086	227	48	2027
elliptic_k5	0	0	139	52	216	144	551
elliptic_k5_npn	0	0	289	144	202	4	639
ex1010_k5	0	0	124	208	318	657	1307
ex1010_k5_npn	0	0	421	713	506	6	1646
ex5p_k5	0	0	55	142	196	307	700
ex5p_k5_npn	0	0	231	624	15	9	879
misex3_k5	0	0	56	163	208	558	985
misex3_k5_npn	0	0	221	393	454	21	1089
s298_k5	0	6	7	6	4	39	62
s298_k5_npn	0	6	17	21	28	3	75
seq_k5	0	0	108	197	312	789	1406
seq_k5_npn	0	0	323	659	620	0	1602
tseng_k5	1	0	129	581	383	771	1865
tseng_k5_npn	1	0	471	1049	628	0	2149



Şekil 4-6-MCNC Referans Devreleri NPN ve Normal Sentez Fonksiyon Sayılarının Girişlere Göre (K) Dağılımı

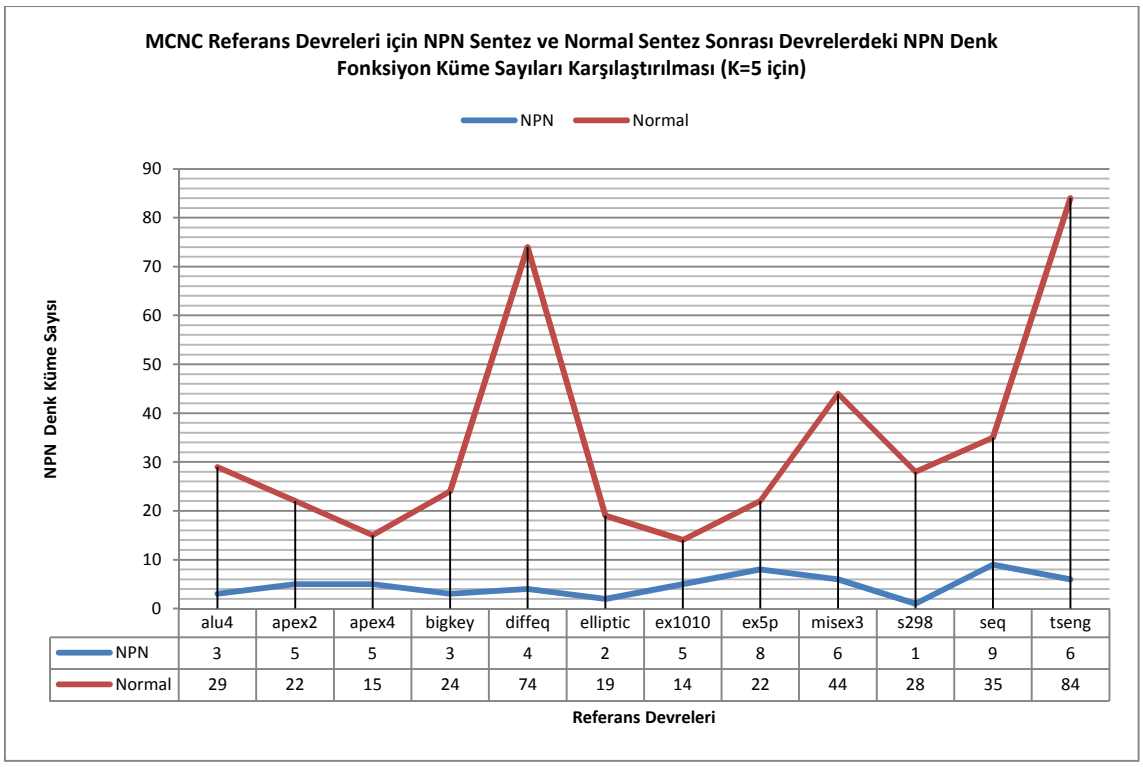
Tablo 4-9'da ve Tablo 4-8'de verdiğimiz fonksiyonlar incelendiğinde, NPN ilişkisine göre gruplandığında NPN sentez öncesi ve sonrası kaç farklı grup oluştuğu ve grubu temsil eden fonksiyonun giriş sayısı görülür. Bu değişim Şekil 4-7 ve Şekil 4-8'de grafiksel olarak ifade edilmiştir.

Örneğin alu4_k5 incelendiğinde yirmi dokuz farklı NPN fonksiyon kümesi olduğu görülür. Bu yirmi dokuz adet fonksiyon kümesi içerisinde iki, üç, dört ve beş girişli fonksiyonları barındırabilir. Burada beş girişli (K=5) fonksiyonların yirmi dokuz adet olmasının nedeni kümeyi oluşturan elemanların içerisinde en çok girişli olanını temsilci

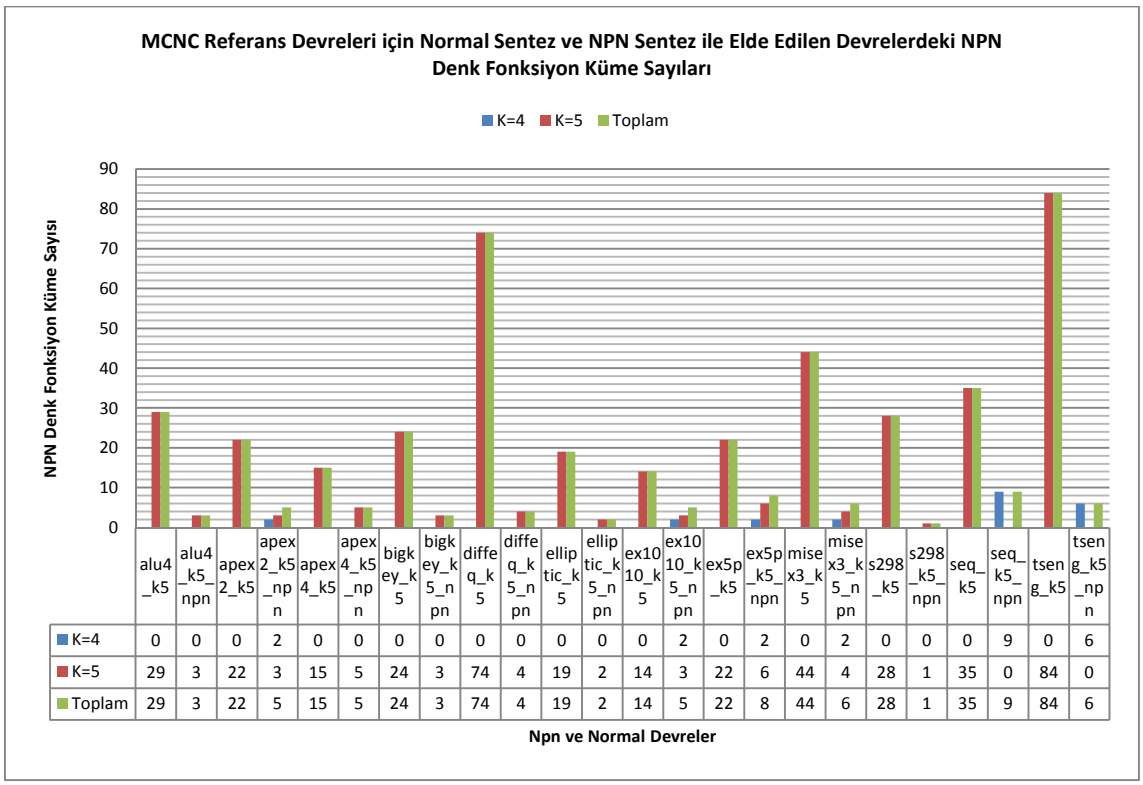
fonksiyon seçmemizdir. Tablo 4-9'daki başka bir örneğe baktığımızda apex2_k5_npn devresinin analizi sonucu dört girişli temsil fonksiyonları olduğu görülür. Yaptığımız iyileştirmedeki asıl amaç fonksiyon kümelerini azaltmaktır. MCNC referans devrelerinden elde ettiğimiz Şekil 4-7'de ve Tablo 4-9'daki sonuçlara baktığımızda fonksiyon küme sayısındaki azalma her bir devre için görülebilir. Tablo 4-9'da NPN küme sayılarını gösterilmiştir. Ayrıca Şekil 4-8'de kümeyi oluşturan fonksiyonların referans fonksiyon girişlerine göre dağılımı mevcuttur.

Tablo 4-9-MCNC Referans Devreleri NPN Fonksiyon Küme Sayıları

Devre Adı	K=0	K=1	K=2	K=3	K=4	K=5	Toplam Fonksiyon Küme Sayısı
alu4_k5	0	0	0	0	0	29	29
alu4_k5_npn	0	0	0	0	0	3	3
apex2_k5	0	0	0	0	0	22	22
apex2_k5_npn	0	0	0	0	2	3	5
apex4_k5	0	0	0	0	0	15	15
apex4_k5_npn	0	0	0	0	0	5	5
bigkey_k5	0	0	0	0	0	24	24
bigkey_k5_npn	0	0	0	0	0	3	3
diffeq_k5	0	0	0	0	0	74	74
diffeq_k5_npn	0	0	0	0	0	4	4
elliptic_k5	0	0	0	0	0	19	19
elliptic_k5_npn	0	0	0	0	0	2	2
ex1010_k5	0	0	0	0	0	14	14
ex1010_k5_npn	0	0	0	0	2	3	5
ex5p_k5	0	0	0	0	0	22	22
ex5p_k5_npn	0	0	0	0	2	6	8
misex3_k5	0	0	0	0	0	44	44
misex3_k5_npn	0	0	0	0	2	4	6
s298_k5	0	0	0	0	0	28	28
s298_k5_npn	0	0	0	0	0	1	1
seq_k5	0	0	0	0	0	35	35
seq_k5_npn	0	0	0	0	9	0	9
tseng_k5	0	0	0	0	0	84	84
tseng_k5_npn	0	0	0	0	6	0	6



Şekil 4-7-MCNC Referans Devreleri için NPN Sentez ve Normal Sentez Sonrası Devrelerdeki NPN Denk Fonksiyon Küme Sayıları Karşılaştırılması (K=5 için)



Şekil 4-8- MCNC Referans Devreleri için Normal Sentez ve NPN Sentez ile Elde Edilen Devrelerdeki NPN Denk Fonksiyon Küme Sayıları

Yukarıda verilen fonksiyon küme sayılarındaki azalma, devreyi oluşturan fonksiyonların birbirlerine olan benzerliği artırır. Benzerliğin artması SRAM paylaşımını artırır.

Tablo 4-9'da verilen fonksiyon kümelerini vermiştik, ilerleyen tablolarda ise bu kümedeki fonksiyon sayılarını, temsili fonksiyonları ve her bir kümedeki farklı fonksiyonları girişlerine göre tablolar halinde gösterilmiştir. Tablolardaki kümeyi temsil eden fonksiyonlar için 05 - 0000B8FC şeklinde gösterim kullanılmıştır. İlk bayt (05) fonksiyon giriş sayısını, geri kalan kısım ise doğruluk tablosunun on altılık sistemdeki gösterimidir (0000B8FC). Bu şekilde fonksiyonlar için farklı gösterim şekilleri elde edilmiştir. Satırlar temsilci fonksiyon detaylarını göstermektedir. Aşağıda sonraki bölümlerde yer alan tabloların nasıl yorumlanacağı ile ilgili örnek verilmiştir.

Toplam Lut Sayısı	1041						
alu4_k5	K=0	K=1	K=2	K=3	K=4	K=5	Toplam
K=5							
050000B8FC	0	0	4	28	18	0	50

Yukarıdaki tablo örneğinde temsilci fonksiyon 050000B8FC ile gösterilmiştir. Devre toplam bin kırk bir adet LUT (fonksiyon) vardır ve 050000B8FC temsilci fonksiyonuna sahip küme elli adet farklı fonksiyon barındırmaktır. Bu elli adet fonksiyonun kaç girişli fonksiyonlardan oluştuğu bilgisi yanında verilmektedir: İki girişli dört adet farklı fonksiyon ve üç girişli yirmi sekiz adet farklı fonksiyon bulundurmaktadır. Beş girişli bir adet fonksiyon vardır ve kendisini de temsilci fonksiyon olarak aldığımız için sıfır olarak görünmektedir. Yani kendisi dışında ilişkisi olan beş girişli fonksiyon yoktur. İlerleyen tablolarda benzer şekilde gösterimler yapılmıştır.

Tablo 4-10 - Alu4_k5 ve Alu4_k5_npn Fonksiyon Küme Detayları

Toplam Lut Sayısı	1041							Toplam Lut Sayısı	1160						
alu4_k5	K=0	K=1	K=2	K=3	K=4	K=5	Toplam	alu4_k5_npn	K=0	K=1	K=2	K=3	K=4	K=5	Toplam
K=5								K=5							
050000B8FC	0	0	4	28	18	0	50	05000A0000	0	0	4	9	35	0	48
050000F000	0	0	4	8	13	29	54	05C00A0000	0	0	4	9	16	0	29
050000F700	0	0	4	16	14	10	44	05C88A0000	0	0	4	41	118	0	163
0500553377	0	0	4	20	8	0	32	K=4							
0500000000	0	0	4	32	61	0	97	K=3							
0500050003	0	0	4	16	31	7	58	K=2							
05C00A0000	0	0	4	8	11	0	23	K=1							
05000C0000	0	0	4	8	11	29	52	K=0							
05AA08AAAA	0	0	4	28	25	10	67								
05D8880000	0	0	4	16	36	2	58								
05AA00AAAA	0	0	4	20	9	3	36								
050055AAFF	0	0	4	12	7	0	23								
05004F0000	0	0	4	16	20	15	55								
05000030FC	0	0	4	16	1	3	24								
05CFCC0000	0	0	4	28	16	10	58								
050F5F3F7F	0	0	4	28	24	0	56								
05000037BF	0	0	4	20	17	3	44								
05C88A0000	0	0	4	28	47	0	79								
05000D0000	0	0	4	8	24	17	53								
050000FF00	0	0	4	8	7	5	24								
052A2F0000	0	0	4	28	38	1	71								
054040EA40	0	0	4	20	22	0	46								
050000FF80	0	0	4	20	23	8	55								
0500008000	0	0	4	8	12	17	41								
050000F300	0	0	4	16	7	18	45								
050000FFFF	0	0	0	0	0	0	0								
05EC00A000	0	0	4	28	36	7	75								
05DFDD5555	0	0	4	28	32	12	76								
050455AEFF	0	0	4	24	26	0	54								
K=4															
K=3															
K=2															
K=1															
K=0															

Tablo 4-11 - Apex2_k5 ve Apex2_k5_npn Fonksiyon Küme Detayları

apex2_k5	K=0	K=1	K=2	K=3	K=4	K=5	Toplam
K=5							
05070F0000	0	0	4	13	16	17	50
0511000000	0	0	4	13	28	39	84
0500000000	0	0	4	29	67	0	100
0500001000	0	0	4	8	16	17	45
0511100000	0	0	4	23	26	31	84
0500005F11	0	0	4	29	45	0	78
05FFFF0000	0	0	0	1	0	0	1
050808080C	0	0	4	14	31	26	75
05CCCCCCCF	0	0	4	23	0	1	28
0503FF0000	0	0	4	20	23	24	71
05035703FF	0	0	4	21	0	0	25
0547000000	0	0	4	13	37	0	54
05030F0000	0	0	4	13	10	27	54
0555555500	0	0	4	20	7	5	36
05AAA0AAA8	0	0	4	29	11	23	67
05FEEEFCCC	0	0	4	21	33	0	58
05F4F00000	0	0	4	20	29	23	76
05FAAAF000	0	0	4	20	6	0	30
0500FF0000	0	0	4	5	6	12	27
05BA300000	0	0	4	28	45	15	92
0507FF0000	0	0	4	20	23	22	69
05000F0000	0	0	4	8	12	31	55
K=4							
K=3							
K=2							
K=1							
K=0							

apex2_k5_npn	K=0	K=1	K=2	K=3	K=4	K=5	Toplam
K=5							
05050F0000	0	0	4	16	72	2	94
05151F0000	0	0	4	40	137	2	183
05000F0000	0	0	4	8	21	2	35
K=4							
040000D800	0	0	4	39	0	0	43
0400000000	0	0	4	40	0	0	44
K=3							
K=2							
K=1							
K=0							

Tablo 4-12- Apex4_k5 ve Apex4_k5_npn Fonksiyon Küme Detayları

apex4_k5	K=0	K=1	K=2	K=3	K=4	K=5	Toplam
K=5							
0500000000	0	0	4	16	27	0	47
0501010103	0	0	4	11	16	2	33
050C3F0000	0	0	4	15	0	0	19
050000333F	0	0	4	10	0	3	17
0500000400	0	0	4	9	11	18	42
0503000000	0	0	4	9	10	8	31
050000007F	0	0	4	10	10	5	29
05F4004400	0	0	4	15	6	1	26
050000003F	0	0	4	10	10	7	31
050000000F	0	0	4	8	6	7	25
050000777F	0	0	4	15	0	1	20
050000FF00	0	0	4	2	0	1	7
05AEBF0000	0	0	4	15	0	0	19
05A800FF00	0	0	4	7	0	5	16
0513000000	0	0	4	11	16	12	43
K=4							
K=3							
K=2							
K=1							
K=0							

apex4_k5_npn	K=0	K=1	K=2	K=3	K=4	K=5	Toplam
K=5							
050AFF0000	0	0	4	22	0	0	26
05333337BF	0	0	5	24	0	1	30
05000005AF	0	0	5	30	8	0	43
05000000AA	0	0	5	17	5	0	27
0500FF0000	0	0	4	8	0	0	12
K=4							
K=3							
K=2							
K=1							
K=0							

Tablo 4-13- Bigkey_k5 ve Bigkey_k5_npn Fonksiyon Küme Detayları

bigkey_k5	K=0	K=1	K=2	K=3	K=4	K=5	Toplam
K=5							
05000001AA	0	1	1	4	2	0	8
0500FF7821	0	1	1	4	1	0	7
05805F01FA	0	1	1	0	0	0	2
0500000000	0	1	1	6	6	0	14
050000000E	0	1	1	2	1	0	5
0509080100	0	1	1	4	1	0	7
058800FFFF	0	1	1	2	0	1	5
05807F01FE	0	1	1	0	0	0	2
05F8F0FFFF	0	1	1	4	0	1	7
0500FF7820	0	1	1	4	1	0	7
05005501AA	0	1	1	4	1	0	7
0540810000	0	1	1	4	2	0	8
0500FF78E1	0	1	1	2	1	0	5
0500007820	0	1	1	4	1	0	7
05000C0001	0	1	1	4	2	1	9
050000000C	0	1	1	2	1	1	6
0500000001	0	1	1	2	2	2	8
05805501AA	0	1	1	4	0	0	6
05000000AA	0	1	1	4	1	1	8
050000FFFF	0	1	0	0	0	0	1
05807F01FA	0	1	1	0	0	0	2
05000E0001	0	1	1	4	1	1	8
0500810000	0	1	1	4	2	0	8
05805F01AA	0	1	1	2	0	0	4
K=4							
K=3							
K=2							
K=1							
K=0							

bigkey_k5_npn	K=0	K=1	K=2	K=3	K=4	K=5	Toplam
K=5							
0540810000	0	1	5	11	9	0	26
0500810000	0	1	5	6	8	0	20
0500010000	0	1	5	6	7	0	19
K=4							
K=3							
K=2							
K=1							
K=0							

Tablo 4-14- Diffeq_k5 ve Diffeq _k5_npn Fonksiyon Küme Detayları

diffeq_k5	K=0	K=1	K=2	K=3	K=4	K=5	Toplam
K=5							
05AEEA8448	0	0	6	24	0	1	31
05FFFF0000	0	0	0	1	1	0	2
0500000030	0	0	5	8	5	12	30
05AABB2033	0	0	6	31	4	0	41
050000ECA0	0	0	5	28	21	1	55
050000A815	0	0	5	30	9	0	44
0522203210	0	0	5	28	21	0	54
05F0020000	0	0	5	9	11	0	25
05FCE8E8C0	0	0	5	27	0	9	41
05FCA80000	0	0	5	28	11	10	54
05AEEA0448	0	0	6	24	4	0	34
05000FEAC0	0	0	6	11	2	0	19
0585555555	0	0	5	21	17	0	43
05502AA995	0	0	6	28	2	0	36
053C2941C3	0	0	6	37	0	2	45
05FFAAFF00	0	0	4	21	3	1	29
05FC000000	0	0	5	14	9	12	40
0595555555	0	0	5	21	6	0	32
05AAAE0884	0	0	5	31	11	0	47
05AAEA0048	0	0	5	29	16	1	51
05CC00135F	0	0	6	12	0	1	19
05AAAE0004	0	0	5	29	15	0	49
053C2841C3	0	0	6	29	0	2	37
05000000C3	0	0	5	14	1	4	24
05F0000000	0	0	5	7	10	11	33
05FEEAA000	0	0	5	34	2	1	42
0502000100	0	0	5	15	15	1	36
05562AA995	0	0	6	28	4	3	41
05002AA995	0	0	6	42	3	0	51
0500AA88AA	0	0	5	29	15	0	49
05FEEAA800	0	0	5	34	4	1	44
05F0C20000	0	0	5	29	10	0	44
0520002030	0	0	5	28	15	2	50
0500002322	0	0	5	29	26	0	60
05004551A2	0	0	6	20	11	0	37
05ECA0135F	0	0	6	19	0	2	27
05AAAA0008	0	0	5	22	7	0	34
05FFEAFF48	0	0	5	28	11	2	46
05FEEA0000	0	0	5	28	4	1	38
050000A915	0	0	5	30	10	0	45
05FCC00000	0	0	5	27	1	1	34
05BBAB3303	0	0	6	31	3	0	40

diffeq_k5_npn	K=0	K=1	K=2	K=3	K=4	K=5	Toplam
K=5							
0500110000	0	0	5	16	21	6	48
0511110001	0	0	5	40	28	11	84
0511110000	0	0	5	18	12	4	39
0500000000	0	0	5	42	39	0	86
K=4							
K=3							
K=2							
K=1							
K=0							

Tablo 4-15-Diffeq_k5 ve Diffeq _k5_npn Fonksiyon Küme Detayları (devamı)

diffeq_k5	K=0	K=1	K=2	K=3	K=4	K=5	Toplam	diffeq_k5_npn	K=0	K=1	K=2	K=3	K=4	K=5	Toplam
K=5															
050000A995	0	0	5	30	1	0	36								
05FFAAFF08	0	0	5	28	14	1	48								
050000A8AA	0	0	5	29	13	0	47								
05FFECFF20	0	0	5	28	4	0	37								
05FFFF00C0	0	0	4	14	1	0	19								
0580555555	0	0	5	30	9	0	44								
0500000000	0	0	6	45	50	0	101								
05566AA995	0	0	6	11	2	3	22								
050000D1C0	0	0	5	28	13	2	48								
05000AA995	0	0	6	40	1	0	47								
05542AA995	0	0	6	28	2	1	37								
0500000015	0	0	5	15	20	1	41								
053C0041C3	0	0	6	27	0	1	34								
050AAA8AAA	0	0	5	28	13	1	47								
050000A015	0	0	5	29	20	0	54								
05000041C3	0	0	5	30	2	2	39								
050000F1E0	0	0	5	27	15	2	49								
0500005555	0	0	5	9	4	2	20								
0500000100	0	0	5	8	6	3	22								
053C6941C3	0	0	6	35	0	1	42								
05AABB0033	0	0	6	22	2	2	32								
05000001C3	0	0	5	27	2	2	36								
05FCA00000	0	0	5	27	12	10	54								
05053FEAC0	0	0	6	31	2	0	39								
05FCA8A000	0	0	5	34	11	7	57								
050C0041C3	0	0	5	34	0	2	41								
053C0841C3	0	0	6	27	0	2	35								
0580005555	0	0	6	24	8	0	38								
0502323202	0	0	6	31	2	1	40								
05F0D20000	0	0	5	29	12	0	46								
05003FEAC0	0	0	6	18	0	0	24								
05FCA8A800	0	0	5	34	9	7	55								
K=4															
K=3															
K=2															
K=1															
K=0															

Tablo 4-16- Elliptic_k5 ve Elliptic_k5_npn Fonksiyon Küme Detayları

elliptic_k5	K=0	K=1	K=2	K=3	K=4	K=5	Toplam
K=5							
0502000A08	0	0	5	12	16	0	33
0513331311	0	0	5	18	1	0	24
0500000007	0	0	5	8	10	0	23
05F0020000	0	0	5	9	10	1	25
050000C0CA	0	0	5	18	12	0	35
0500000003	0	0	5	8	3	2	18
050000F000	0	0	5	7	7	4	23
05F0C20000	0	0	5	17	8	1	31
05AAAA000F	0	0	5	11	7	0	23
050000C0CC	0	0	5	16	5	1	27
0503330300	0	0	5	17	0	0	22
050000C4CC	0	0	5	17	6	0	28
05EEEECCCF	0	0	5	14	5	0	24
0500000000	0	0	5	20	25	0	50
050000C0F5	0	0	5	15	6	0	26
0500000200	0	0	5	8	4	5	22
0502010000	0	0	5	12	10	3	30
05F0D20000	0	0	5	17	9	1	32
050000C4CE	0	0	5	19	6	0	30
K=4							
K=3							
K=2							
K=2							
K=2							

elliptic_k5_npn	K=0	K=1	K=2	K=3	K=4	K=5	Toplam
K=5							
0502000A08	0	0	5	8	39	1	53
0500000808	0	0	5	8	21	1	35
K=4							
K=3							
K=2							
K=2							
K=2							

Tablo 4-17- Ex5p_k5 ve Ex5p_k5_npn Fonksiyon Küme Detayları

ex5p_k5	K=0	K=1	K=2	K=3	K=4	K=5	Toplam
K=5							
0500000002	0	0	5	9	10	20	44
050003FFFF	0	0	4	10	9	1	24
0500000202	0	0	5	14	17	9	45
05020F22AF	0	0	5	15	13	0	33
0570000000	0	0	5	9	16	4	34
05101211FF	0	0	5	15	9	0	29
05EEEF0000	0	0	5	18	18	8	49
050000FFFF	0	0	0	1	1	0	2
05000FFFFF	0	0	4	9	1	5	19
05000211FF	0	0	4	15	14	0	33
050001FFFF	0	0	4	10	10	4	28
05808FFFFF	0	0	5	17	1	0	23
05000000DD	0	0	5	14	15	3	37
050F1F0000	0	0	5	14	13	6	38
05000F88FF	0	0	4	9	2	2	17
0500000000	0	0	5	24	45	0	74
05000200AA	0	0	5	23	25	3	56
05000011FF	0	0	4	15	14	4	37
05000F0000	0	0	5	8	7	7	27
05000F22AF	0	0	5	14	14	0	33
05000000FF	0	0	4	5	5	2	16
05000B00FF	0	0	4	14	9	5	32
K=4							
K=3							
K=2							
K=1							
K=0							

ex5p_k5_npn	K=0	K=1	K=2	K=3	K=4	K=5	Toplam
K=5							
050F0F0FAF	0	0	5	32	0	0	37
05AAAA0000	0	0	5	11	1	0	17
05000003CF	0	0	5	46	3	0	54
05000000CC	0	0	5	17	4	1	27
05000047CF	0	0	5	48	3	2	58
05BBAA1100	0	0	5	41	0	0	46
K=4							
0400007C00	0	0	5	32	0	0	37
0400003C00	0	0	5	2	0	0	7
K=3							
K=2							
K=1							
K=0							

Tablo 4-18- Ex1010_k5 ve Ex1010_k5_npn Fonksiyon Küme Detayları

ex1010_k5	K=0	K=1	K=2	K=3	K=4	K=5	Toplam	ex1010_k5_npn	K=0	K=1	K=2	K=3	K=4	K=5	Toplam
K=5								K=5							
050F3F0000	0	0	3	6	3	0	12	05030F0000	0	0	5	14	18	1	38
0503000000	0	0	3	9	16	6	34	05000F0000	0	0	5	8	12	1	26
0557000000	0	0	3	11	24	3	41	05131F0000	0	0	5	15	32	1	53
05A8A08800	0	0	3	7	4	3	17	K=4							
050055007F	0	0	3	14	13	1	31	040000000F	0	0	4	6	4	2	16
055F7F0000	0	0	3	6	3	0	12	0400000000	0	0	5	28	0	3	36
0507000000	0	0	3	9	24	7	43	K=3							
05007F0000	0	0	3	10	10	3	26	K=2							
0511111113	0	0	3	11	16	0	30	K=1							
0500000000	0	0	3	15	38	0	56	K=0							
0580000000	0	0	3	9	17	10	39								
05000F0000	0	0	3	8	8	8	27								
050101030F	0	0	3	14	13	0	30								
05003F0000	0	0	3	10	10	5	28								
K=4															
K=3															
K=2															
K=1															
K=0															

Tablo 4-19- Misex3_k5 ve Misex3_k5_npn Fonksiyon Küme Detayları

misex3_k5	K=0	K=1	K=2	K=3	K=4	K=5	Toplam
K=5							
050000000F	0	0	4	8	13	19	44
0500F000B0	0	0	4	14	17	12	47
0500000CFF	0	0	4	18	15	7	44
0507000500	0	0	4	15	41	5	65
05EAAFAAAF	0	0	4	19	9	0	32
050000C0AF	0	0	4	26	15	0	45
050000BB20	0	0	4	27	35	0	66
050000FB60	0	0	4	26	22	0	52
0500F080F0	0	0	4	18	22	7	51
05800033FF	0	0	4	18	1	0	23
0500001DFF	0	0	4	18	16	0	38
05025222FF	0	0	4	19	19	0	42
05FF030003	0	0	4	7	1	0	12
058301114B	0	0	4	27	0	0	31
050000114A	0	0	4	15	24	0	43
050000FFFF	0	0	0	0	0	0	0
05CF0F0F0F	0	0	4	12	0	1	17
0500007000	0	0	4	9	24	20	57
050000004A	0	0	4	9	25	0	38
053003CCFF	0	0	4	19	0	0	23
05A0A80088	0	0	4	27	29	4	64
05005022FF	0	0	4	19	24	0	47
058300114A	0	0	4	27	0	0	31
05830333FF	0	0	4	18	1	0	23
053000CCFF	0	0	4	18	0	0	22
058000C040	0	0	4	27	23	2	56
0500020000	0	0	4	9	12	14	39
050000C1AF	0	0	4	26	15	0	45
05B003CCFF	0	0	4	19	0	0	23
05C0000000	0	0	4	9	11	22	46
050C4CCCCC	0	0	4	27	1	10	42
0500000000	0	0	4	28	64	0	96
05000044B0	0	0	4	27	24	0	55
0533A00000	0	0	4	27	13	1	45
050133CDFF	0	0	4	19	1	0	24
050300114A	0	0	4	27	13	0	44
05FFABAAAB	0	0	4	28	3	0	35

misex3_k5_npn	K=0	K=1	K=2	K=3	K=4	K=5	Toplam
K=5							
053F220000	0	0	5	38	99	6	148
050C5D0000	0	0	5	34	121	0	160
053F000000	0	0	5	15	41	5	66
050F000000	0	0	5	8	22	6	41
K=4							
040000AA00	0	0	5	9	8	10	32
0400000000	0	0	5	39	0	4	48
K=3							
K=2							
K=1							
K=0							

Tablo 4-20-Misex3_k5 ve Misex3_k5_npn Fonksiyon Küme Detayları (devamı)

misex3_k5	K=0	K=1	K=2	K=3	K=4	K=5	Toplam	misex3_k5_npn	K=0	K=1	K=2	K=3	K=4	K=5	Toplam
K=5															
050000F311	0	0	4	26	25	0	55								
05000000FF	0	0	4	6	7	7	24								
05C00F000F	0	0	4	6	0	0	10								
050033CCFF	0	0	4	7	0	0	11								
05000000AF	0	0	4	14	30	18	66								
0500DF00CC	0	0	4	26	16	5	51								
05000033FF	0	0	4	18	1	6	29								
K=4															
K=3															
K=2															
K=1															
K=0															

Tablo 4-21- S298_k5 ve S298_k5_npn Fonksiyon Küme Detayları

s298_k5	K=0	K=1	K=2	K=3	K=4	K=5	Toplam
K=5							
05333300AF	0	1	4	3	0	0	8
058B808880	0	1	4	2	1	0	8
0500CECCCC	0	1	4	3	0	0	8
058B8F8B83	0	1	4	3	0	0	8
05888F8B83	0	1	4	3	0	0	8
0500050000	0	1	4	1	2	3	11
05000040AA	0	1	4	2	2	0	9
05070F0703	0	1	4	2	0	0	7
05000048AA	0	1	4	2	2	0	9
0500000DF0	0	1	4	2	1	0	8
05000F0303	0	1	4	2	0	1	8
0500070002	0	1	4	2	2	0	9
05880F8800	0	1	4	2	1	0	8
050000B8B0	0	1	4	2	1	0	8
05000068AA	0	1	4	2	1	0	8
050000B830	0	1	4	2	1	2	10
058880888C	0	1	4	2	2	0	9
05C0CECCCC	0	1	4	1	0	0	6
05000000F0	0	1	4	1	1	4	11
05C8CECCCC	0	1	4	1	0	0	6
0500070006	0	1	4	2	2	0	9
0500000CF0	0	1	4	2	1	0	8
058B0F8B03	0	1	4	3	0	0	8
050000CCCC	0	1	4	1	0	0	6
05880F8B03	0	1	4	3	0	0	8
0500080000	0	1	4	1	1	0	7
05000000AF	0	1	4	2	1	0	8
05000ACCCC	0	1	4	2	0	0	7
K=4							
K=3							
K=2							
K=1							
K=0							

s298_k5_npn	K=0	K=1	K=2	K=3	K=4	K=5	Toplam
K=5							
0500000000	0	1	4	14	28	0	47
K=4							
K=3							
K=2							
K=1							
K=0							

Tablo 4-22- Seq_k5 ve Seq_k5_npn Fonksiyon Küme Detayları

seq_k5	K=0	K=1	K=2	K=3	K=4	K=5	Toplam
K=5							
050000000F	0	0	5	8	15	20	48
050000FF00	0	0	4	7	6	13	30
0530F00000	0	0	5	15	13	20	53
050000FF80	0	0	4	25	24	17	70
050B8E0000	0	0	5	33	35	0	73
053421FFFF	0	0	5	35	2	0	42
05038C0000	0	0	5	15	2	0	22
050000FFFF	0	0	0	2	2	0	4
050380FFFF	0	0	4	20	6	0	30
057F007700	0	0	5	35	26	17	83
05C8000000	0	0	5	15	30	27	77
050A8A5FDF	0	0	5	36	17	1	59
05D1D00000	0	0	5	33	46	1	85
0500170000	0	0	5	15	43	1	64
0570600000	0	0	5	15	45	0	65
0500FF05FF	0	0	4	20	8	3	35
0500280000	0	0	5	8	18	2	33
05C0000000	0	0	5	8	15	28	56
05373F0000	0	0	5	35	20	11	71
05000FF0FF	0	0	4	1	0	0	5
050000010F	0	0	5	15	28	12	60
05008855DD	0	0	5	28	10	0	43
05004700FF	0	0	4	27	24	0	55
050401FFFF	0	0	4	27	6	0	37
05D1C00000	0	0	5	33	17	2	57
0500000000	0	0	5	36	96	0	137
0500000FFF	0	0	4	18	2	3	27
0570F00000	0	0	5	15	19	22	61
050200FFFF	0	0	4	20	6	4	34
053401FFFF	0	0	5	27	4	0	36
053F003300	0	0	5	35	18	16	74
0570200000	0	0	5	15	30	0	50
0501000000	0	0	5	8	16	24	53
0500000383	0	0	5	33	15	0	53
05E0A0C000	0	0	5	33	46	2	86
K=4							
K=3							
K=2							
K=1							
K=0							

seq_k5_npn	K=0	K=1	K=2	K=3	K=4	K=5	Toplam
K=5							
K=4							
0400000B0A	0	0	5	38	47	0	90
0400000600	0	0	5	9	2	0	16
0400000300	0	0	5	8	22	0	35
0400002021	0	0	5	40	0	0	45
0400000700	0	0	5	16	45	0	66
0400000000	0	0	5	42	0	0	47
0400000200	0	0	5	9	3	0	17
040000F000	0	0	4	8	10	0	22
040000E400	0	0	5	37	4	0	46
K=3							
K=2							
K=1							
K=0							

Tablo 4-23- Tseng_k5 ve Tseng _k5_npn Fonksiyon Küme Detayları

tseng_k5	K=0	K=1	K=2	K=3	K=4	K=5	Toplam
K=5							
053C9686C3	0	0	6	38	6	5	55
05AAAE000C	0	0	5	34	27	7	73
052C00ECCC	0	0	5	38	8	0	51
0500000001	0	0	5	8	13	6	32
050000139F	0	0	5	33	12	2	52
05008F8F70	0	0	6	4	0	0	10
05F7755000	0	0	5	33	4	2	44
05000AA665	0	0	7	42	5	1	55
05F5313100	0	0	5	42	2	1	50
05308F8F70	0	0	6	6	3	0	15
050000038F	0	0	5	32	11	1	49
05F5F5A0E4	0	0	5	32	14	1	52
053C9682C3	0	0	7	38	3	2	50
0500FF00F0	0	0	4	17	0	3	24
0555550044	0	0	5	34	30	3	72
05518AA665	0	0	7	33	6	2	48
05000F8F70	0	0	6	6	2	0	14
0500AACCEE	0	0	7	33	7	0	47
05ECCECEEC	0	0	7	26	4	0	37
0533630000	0	0	5	34	18	1	58
052E0C2200	0	0	7	25	20	1	53
05CFC00000	0	0	5	32	2	0	39
055050A0A0	0	0	5	14	10	0	29
05708F8F70	0	0	6	6	3	0	15
0500000F70	0	0	5	12	5	0	22
0538840000	0	0	5	21	4	0	30
057250A2A0	0	0	7	48	32	2	89
05000000C3	0	0	5	15	6	3	29
05F0000000	0	0	5	7	15	13	40
05CCCC0001	0	0	7	36	8	0	51
05CCCCCCC9	0	0	5	26	4	0	35
0503222222	0	0	5	17	25	8	55
050C0C0D0C	0	0	5	34	14	0	53
053C0082C3	0	0	7	35	0	2	44
0550000000	0	0	5	8	28	20	61
05F7750000	0	0	5	33	12	3	53
0500FFA0FF	0	0	4	18	2	1	25
050AAACEEE	0	0	7	42	19	1	69

tseng_k5_npn	K=0	K=1	K=2	K=3	K=4	K=5	Toplam
K=5							
K=4							
0400000F01	0	0	4	38	60	0	102
0400000F00	0	0	4	8	11	0	23
0400000C00	0	0	5	8	22	0	35
0400008B00	0	0	5	38	15	0	58
0400000000	0	0	5	49	0	0	54
0400000C08	0	0	5	16	24	0	45
K=3							
K=2							
K=1							
K=0							

Tablo 4-24 - Tseng_k5 ve Tseng _k5_npn Fonksiyon Küme Detayları (devamı-1)

tseng_k5	K=0	K=1	K=2	K=3	K=4	K=5	Toplam	tseng_k5_npn	K=0	K=1	K=2	K=3	K=4	K=5	Toplam
K=5															
05000082C3	0	0	5	38	11	3	57								
05008AA665	0	0	7	48	8	1	64								
053B000000	0	0	5	16	24	8	53								
0508840000	0	0	5	21	33	0	59								
05000002C3	0	0	5	32	8	3	48								
05000A0AA0	0	0	7	10	18	0	35								
05D0008000	0	0	5	16	36	3	60								
050C0082C3	0	0	5	42	0	3	50								
0522A200A0	0	0	5	39	24	0	68								
053C9696C3	0	0	4	17	3	5	29								
05F5313000	0	0	5	38	3	2	48								
05000005B7	0	0	5	32	19	0	56								
0500008F70	0	0	5	30	6	0	41								
0500840000	0	0	5	16	27	0	48								
05CCCCCC01	0	0	5	36	2	0	43								
05F7755110	0	0	5	37	2	5	49								
05F7755100	0	0	5	37	11	3	56								
053300FFCC	0	0	4	15	1	0	20								
05C0C0C4DD	0	0	7	35	8	1	51								
05F3000000	0	0	5	15	9	10	39								
05CCCCCCC1	0	0	5	26	3	0	34								
0500000045	0	0	5	16	37	3	61								
0500000000	0	0	7	57	137	0	201								
053C1482C3	0	0	7	37	0	3	47								
050000A045	0	0	5	37	27	1	70								
050000A245	0	0	5	38	12	1	56								
053C1682C3	0	0	7	39	3	3	52								
0533430000	0	0	5	33	12	1	51								
0538940000	0	0	5	21	10	0	36								
0533400000	0	0	5	21	17	1	44								
05F2F30000	0	0	5	26	8	2	41								
05A00A0AA0	0	0	7	8	6	0	21								
05F3300000	0	0	5	32	2	4	43								
05D0808080	0	0	5	17	29	8	59								
0555550000	0	0	5	10	8	7	30								
0500FF00F4	0	0	4	18	1	1	24								
053B220000	0	0	5	34	32	1	72								
05508AA665	0	0	7	33	8	1	49								
053C0482C3	0	0	7	35	0	3	45								
05000000BF	0	0	5	15	15	2	37								

Tablo 4-25-Tseng_k5 ve Tseng _k5_npn Fonksiyon Küme Detayları (devamı-2)

tseng_k5	K=0	K=1	K=2	K=3	K=4	K=5	Toplam	tseng_k5_npn	K=0	K=1	K=2	K=3	K=4	K=5	Toplam
K=5															
05DFD50000	0	0	5	33	4	0	42								
0500070EE0	0	0	7	15	0	0	22								
050000A665	0	0	5	38	2	1	46								
050000A645	0	0	5	38	11	1	55								
05F5310000	0	0	5	38	19	3	65								
05F5300000	0	0	5	32	19	3	59								
K=4															
K=3															
K=2															
K=1															
K=0															

5. SONUÇ VE İLERİDE YAPILACAKLAR

Yapmış olduğumuz çalışma, bu alanda yapılmış ilk dinamik çalışmadır. Bugüne kadar yapılan diğer çalışmalarda ya statik kütüphane ya da statik CLB yapıları kullanılmıştır. [1] ve [3]'de yapılan çalışmalardan farklı olarak bizim çalışmamızda devreden referans fonksiyon oluşturma ve sentez aşamaları birbirinden bağımsız olduğu için kullanılacak referans fonksiyonu devreden bağımsız bir şekilde verilebilir. Bu çalışmamızda, SRAM tabanlı FPGA mimarilerinde kullanılan alan miktarını azaltmaya yönelik, mimariden bağımsız, devreye özel çözüm sunan yeni bir yöntem geliştirmeye çalışılmıştır. Kullandığımız bu yöntemde fonksiyonları birbirleri cinsinden ifade edebilecek şekle getirerek daha az fonksiyon çeşidi ile devreyi oluşturmak hedeflenmektedir. Bu şekilde daha az fonksiyon ile ifade edilen devre, FPGA üzerinde daha az SRAM kullanmamızı sağlamaktadır. Geliştirdiğimiz bu yöntem, fonksiyonların NPN denk olup olmadıklarını inceleyerek sentezleme işlemini gerçekleştirmektedir. NPN denklik kuramını kullanarak sentez yapan çalışmalardaki amaç devredeki NPN denklik sayısını artırmaktır. Geliştirdiğimiz yöntem ABC yapısındaki mevcut sentezleme aracından Tablo 4-2'da verilen bilgilere göre MCNC referans devreleri için normal ABC sentezine göre daha fazla NPN denklik artışı sağlamıştır. Her bir devre için NPN denklik artış oranları Tablo 4-2'de verilmiştir. Devrelerdeki bu NPN denklik artışı beraberinde Tablo 4-3'da verilen detaylı bilgilere göre fonksiyon sayısında da normal ABC sentezine göre artış oluşturmuştur. Her bir devrenin fonksiyon sayısındaki artış Tablo 4-3'de verilmiştir. Tablo 4-9'da verilen bilgilerde NPN benzerliğin artışının bir diğer ölçüsü olan NPN denk

küme sayısı normal ABC sentezine göre MCNC referans devreleri için azalmıştır ve devre daha az fonksiyon çeşidi ile ifade edilebilmektedir. Her bir devre için NPN denk küme sayısı değişimi Tablo 4-9'da verilmiştir. Çalışmamızın bir değer katkısı ise karşılaştırma sayısındaki azalma ile sürede yaptığımız iyileştirmedir. Ortalama bir devre sentezi MCNC referans devreleri için 11406,24 saniye sürmektedir. Bunun ortalama 7175,3 saniyesi referans fonksiyon hesaplanması için kullanılmaktadır. Hesaplanan referans fonksiyonları ile değiştirilmiş ABC aracı ile elde edilen NPN sentez süreleri ortalama 4230,94 saniye sürmektedir. Sentez süreleri ortalama yaklaşık 3,16 saat sürdüğü için tek seferlik programlanıp kullanılacak FPGA'lerin fabrikasyon sürecinde analiz için oldukça kabul edilebilir bir süredir.

Çalışmamız sonucunda NPN yoğunluğunu normal ABC sentezlemesine göre MCNC referans devreleri için Tablo 4-2'deki verilere göre başarılı bir şekilde artırmış bulunmaktayız. Çalışmanın ilerleyen aşamalarında NPN karşılaştırmasını sağlayan yapının paralel çalışması sağlanıp daha kısa sürede sonuçların alınması ve analiz edilmesi planlanmaktadır.

Çalışmamızı gerçekleştirirken devredeki NPN denk fonksiyonların birbirine olan uzaklıklarını dikkate alınmadan işlemler yapılmıştır. Bağlama adımında fonksiyonların birbirine uzaklığı sorun teşkil etmektedir. Bu nedenle bundan sonraki çalışmada NPN denkliğe bakıp fonksiyonları ağırlıklandırırken uzaklığın da parametre olarak kullanılması düşünülmektedir. Ayrıca daha performanslı bir tasarım yapmak için sentezleme yaptıktan sonra paketleme adımında birbirine NPN denk olan fonksiyonları aynı SRAM blokları içerisine koymak gerekmektedir. Bu işlemi, bundan sonraki çalışmada fonksiyonları paketleyen TV-PACK yazılımının ilgili kısımlarını değiştirerek yapmayı düşünmekteyiz. Bunlara ek olarak birden fazla referans fonksiyonu ile testler yapılması planlanmaktadır. Yapılan bu çalışmaların altıdan fazla girişe sahip olan fonksiyonlar için de gerçekleştirilmesi planlanmaktadır.

Sonuç olarak çalışmamızda NPN denklik kuramını kullanan çalışmaların ana temelini oluşturan devredeki NPN denklik sayısını artırmayı başarmış olduk. Çalışmamıza etkili bir şekilde kazanç sağlayacak paketleme ve bağlama adımlarında yukarıda açıkladığımız geliştirmeleri yaparak etkili bir sentezleme yöntemi sunmayı hedeflemekteyiz.

KAYNAKÇA

[1] J. Meyer and F. Kocan, “Sharing of SRAM Tables Among NPN-Equivalent LUTs in SRAM-Based FPGAs,” *Ieee Trans. Very Large Scale Integr. Vlsi Syst.*, vol. 15, no. 2, pp. 182–195, 2007.

[2] J. Meyer and F. Kocan, “SRAM Sharing Among NPN Equivalence. PhD Dissertation,” Southern Methodist University, 2007.

[3] Y. Okamoto, Y. Ichinomiya, M. Amagasaki, M. Iida, and T. Sueyoshi, “COGRE: A Configuration Memory Reduced Reconfigurable Logic Cell Architecture for Area Minimization,” in *2010 International Conference on Field Programmable Logic and Applications*, 2010, pp. 304–309.

[4] S. Kimura, T. Horiyama, M. Nakanishi, and H. Kajihara, “Folding of logic functions and its application to look up table compaction,” in *IEEE/ACM International Conference on Computer Aided Design, 2002. ICCAD 2002.*, 2002.

[5] C. Maxfield, *The Design Warrior’s Guide to FPGAs*. Elsevier, 2004, pp. 1–560.

[6] Xilinx, *Virtex-6 Family Overview, Product Specification*, DS150 (v2. ed. Xilinx, 2012, pp. 1–11.

[7] D. Gomez-Prado and M. Ciesielski, “A tutorial on fpga routing,” *Dep. Electr. Comput. E ngineering, Univ. Massachusetts, Amherst, USA*, pp. 1–16, 2006.

[8] E. M. Luks, “Hypergraph isomorphism and structural equivalence of Boolean functions,” in *Proceedings of the thirty-first annual ACM symposium on Theory of computing - STOC ’99*, 1999, pp. 652–658.

[9] U. Hinsberger and R. Kolla, “Boolean matching for large libraries,” in *DAC ’98 Proceedings of the 35th annual Design Automation Conference*, 1998, pp. 206–211.

[10] I. Addison Wesley Longman, “Actel ACT,” in *Application-Specific Integrated Circuits*, 1997, p. Section 5.

[11] D. Chen, J. Cong, and P. Pan, “FPGA Design Automation: A Survey,” *Found. Trends® Electron. Des. Autom.*, vol. 1, no. 3, pp. 195–334, 2006.

[12] R. J. Francis, “Technology Mapping for Lookup-Table Based Field-Programmable Gate Arrays. PhD Dissertation,” University of Toronto, 1993.

[13] A. Kennings, A. Mishchenko, K. Vorwerk, V. Pevzner, and A. Kundu, “Efficient FPGA Resynthesis Using Precomputed LUT Structures,” in *2010 International Conference on Field Programmable Logic and Applications*, 2010, pp. 532–537.

[14] a. Ling, D. P. Singh, and S. D. Brown, “FPGA technology mapping: a study of optimality,” *Proc. 42nd Des. Autom. Conf. 2005*, pp. 427–432, 2005.

[15] K. Hosung, Jim Lillis, “A framework for layout-level logic restructuring,” in *Proceeding ISPD ’08 Proceedings of the 2008 international symposium on Physical design*, 2008, pp. 87–94.

[16] D. Chen and J. Cong, “DAOmap: a depth-optimal area optimization mapping algorithm for FPGA designs,” in *IEEE/ACM International Conference on Computer Aided Design, 2004. ICCAD-2004.*, 2004, pp. 752–759.

[17] J. Cong and Y. D. Y. Ding, *FlowMap: an optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs*, vol. 13, no. 1. IEEE, 1994, pp. 1–12.

[18] J. Cong and K. Minkovich, “Improved SAT-based Boolean matching using implicants for LUT-based FPGAs,” in *Proceedings of the 2007 ACM/SIGDA 15th international symposium on Field programmable gate arrays - FPGA ’07*, 2007, p. 139.

[19] V. Shih, “Exploiting symmetry in SAT-based boolean matching for heterogeneous FPGA technology mapping,” in *2007 IEEE/ACM International Conference on Computer-Aided Design*, 2007, pp. 350–353.

[20] S. Safarpour, A. Veneris, G. Baeckler, and R. Yuan, “Efficient SAT-based Boolean matching for FPGA technology mapping,” *Proc. 43rd Annu. Conf. Des. Autom. - DAC ’06*, p. 466, 2006.

[21] A. Mishchenko, S. Chatterjee, and R. Brayton, “Fast Boolean matching for LUT structures,” 2007.

[22] A. Abdollahi, “A New Canonical Form for Fast Boolean Matching in Logic Synthesis and Verification,” *Proceeding DAC ’05 Proc. 42nd Annu. Des. Autom. Conf.*, pp. 379–384, 2005.

[23] S. G. Hartke and A. J. Radcliffe, “McKay ’ s Canonical Graph Labeling Algorithm,” *Contemp. Math.*, vol. 479, pp. 99–111, 2009.

[24] B. D. McKay and A. Piperno, “Practical graph isomorphism,II,” *J. Symb. Comput.*, vol. 60, pp. 94–112, Jan. 2014.

[25] U. Berkeley, “Berkeley logic interchange format (BLIF),” *Oct Tools Distrib.*, vol. 0, pp. 1–11, 1992.

[26] O. Coudert, “Two-level logic minimization: an overview,” *Integr. VLSI J.*, vol. 17, no. 2, pp. 97–140, Oct. 1994.

EK A

ABC NPN KONTROL EKLENTİSİ

ABC için geliştirdiğimiz eklenti fonksiyonu içeriği aşağıdaki gibidir. Check_NPN_Equailty dışarıdan referans fonksiyon listesini, sayısını, ileride kullanılmak üzere kullanılan fonksiyonların gösterim format bilgisini, ABC tarafından oluşturulan CUT'ların doğruluk tablosu ile değişken sayısını girdi olarak alır. Sonrasında ise NPN denklik kontrolü sağlayan fonksiyonun kullanacağı girdilere dönüştürülen parametre kullanılır.

```
/**Function*************************************************************

 Synopsis    [Performs additional check.]

 Description [Normal flow of NPN Check]

 SideEffects []

 SeeAlso     []

***********************************************************************/
int If_CutPerformCheckNpn( If_Man_t * p, unsigned * pTruth, int nVars, int nLeaves, char * pStr )
{
        int result = -1;
        int i,k,ind,funcIndex;

        char*  abcFuncLut;

        char** funcList                 = p->pPars->pNpnCheckFuncs;
        char*  fileFormat               = p->pPars->npnFileFormat;
        int    funcCount                = p->pPars->NpnCheckFuncsCount;
        int   abcFuncVarCnt    = nLeaves;

        size_t lenAbcFuncLut = 0;
        funcIndex = 0;

        lenAbcFuncLut = (unsignedint) ((powl(2,(double)nVars) )*2)+2;

        abcFuncLut = (char*)malloc(lenAbcFuncLut);

        for(i=0; i < p->nTruthWords ; i++)
_itoa((int)pTruth[i],&abcFuncLut[i*32],2);

        ind = (int)powl(2,(long)abcFuncVarCnt);
        abcFuncLut[ind] = '\0';

        result = CheckNPNEquailty(funcList,fileFormat,funcCount, abcFuncVarCnt,abcFuncLut);

        free(abcFuncLut);
        abcFuncLut = 0;

return result;
}
```

EK B

CONVERT EXP TO LUT ALGORİTMASI

Algoritma 3 Convert_EXP_To_LUT – Part1 : Infix_To_Polish_Notation

```
create a stack object
initialize stack object
while exp[i]!=null do
    if(exp[i] == '(')
        stack.push('(')
    elseif(exp[i] == ')')
        add all stack items to polish notation buffer untill '(' character
    elseif(exp[i] == '!')
        stack.push('!')
    elseif(exp[i] == '|' || exp[i] == '&' || exp[i] == '@')
        pop a stack item
        if it's operand
        push exp[i] item to stack
        else
        pop stack item and add to polish notation buffer
        endif
    else
        untill stack empty add all items to polish notation buffer
    endif
```

end while

calculate variable count from polish notation and return in results

---

Algoritma 4 Convert_EXP_To_LUT – Part2 : Polish_To_Lut

---

calculate LUT length from polish notation variable count

allocate LUT array from length information

evaluate polish notation with stack

copy result to allocated array and return

EK C

CONVERT LUT TO CUBE ALGORİTMASI

Algoritma 4 Convert_LUT_To_CUBE

```
count ones in LUT
allocate ones array of array number of variables
j = 0
for i = 0 to LUT length do
    if  LUT[i]==1 && j< count_of_ones
        copy(cubeList[j], binary representation of i);
        j++;
    end if
end for
return j and cubelist
```

EK D

CREATE NAUTY GRAPH FROM CUBE ALGORİTMASI

Algoritma 5 Create_Graph

```
Set Max Graph Size m

Get empty graph as g from parameters

Allocate set matrix matrixVar

Set Matrix length cubelist.Length+2*cubelist.numberOfVariables

for i = 0 to length of matrix do

        matrixVar [i]= GRAPHROW(g,i,m) ;

        EMPTYSET(matrixVar [i], m) ;

end for

for i=0 to cubeList.length do

for j=cubeList.numberOfVariables-1 to 0 do

        copy(&switchChar,(char*)&cubeList.cubeList[i][j],1);

        switch(cubeList.cubeList[i][j]){

        case '0':

            //add vertex node to edge node

        ADDELEMENT(matrixVar [i], cubeList.len+2*j) ;

            //add edge node to vertex node

        ADDELEMENT(matrixVar [cubeList.len+2*j], i) ;

        break;

        case '1':
```

```
            //add vertex node to edge node
        ADDELEMENT(matrixVar [i], cubeList.len+2*j+1) ;
            //add edge node to vertex node
        ADDELEMENT(matrixVar [cubeList.len+2*j+1], i) ;
        break;
        default: exit break;
          end switch
        end for
end for
deallocate matrixVar
return graph as output g
```