İSTANBUL ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

YÜKSEK LİSANS TEZİ

ETKİN BİT İNDİRGEME METODU KULLANARAK ARİTMETİK İŞLEM DEVRELERİ TASARIMI

Bilg.Müh. Sarmad MOHAMMED
Bilgisayar Mühendisliği Anabilim Dalı
Yüksek Lisans Programı

Danışman
Prof.Dr. Ahmet SERTBAŞ

Haziran, 2011

İSTANBUL

Bu çalışma / /2011 tarihinde aşağıdaki jüri tarafından Bilgisayar Mühendisliği Anabilim Dalı Yüksek Lisans programında Yüksek Lisans Tezi olarak kabul edilmiştir.

Tez Jürisi

Prof.Dr. Ahmet SERTBAŞ (Danışman)
İstanbul Üniversitesi
Mühendislik Fakültesi

Prof.Dr. Sabri ARIK
İstanbul Üniversitesi
Mühendislik Fakültesi

Yrd.Doç.Dr. Muhammed Ali AYDIN
İstanbul Üniversitesi
Mühendislik Fakültesi

Yrd.Doç.Dr. Hakan DOĞAN
İstanbul Üniversitesi
Mühendislik Fakültesi

Yrd.Doç.Dr.Oğuzhan ÖZTAŞ
İstanbul Üniversitesi
Mühendislik Fakültesi

ÖNSÖZ

Yüksek lisans öğrenimim sırasında ve tez çalışmalarım boyunca gösterdiği her türlü destek ve yardımından dolayı çok değerli hocam Prof.Dr. Ahmet SERTBAŞ'a en içten dileklerimle teşekkür ederim.

Sadece bu çalışma boyunca değil, desteğini ve ilgisini hiç bir zaman esirgemeyen çok değerli arkadaşım Muhammet İSENKUL'a teşekkürü borç bilirim.

Bugüne kadar maddi ve manevi hiçbir desteğini benden esirgemeyen, bugünlere gelmemi sağlayan, haklarını hiç bir zaman ödeyemeyeceğim canım annem ve babama sonsuz teşekkürlerimi sunarım.

Haziran, 2011 **Sarmad MOHAMMED**

İÇİNDEKİLER

ŞEKİL LİSTESİ

TABLO LİSTESİ

SEMBOL LİSTESİ

HDL : Donanım Tanımlama Dili
VHDL : Yüksek Hızlı Tümleşik Devreler için Donanım Tanımlama Dili
FPGA : Sahada Programlanabilir Kapı Dizileri
DSP : Sayısal İşaret İşlemi
ALU : Aritmetik Mantık Birimi
GUI : Grafiksel Kullanıcı arayüzü
PP : Kısmi sonuç
FP : Final sonuç
Co : Çıkış eldesi
Cin : Giriş eldesi
LSB : En düşük anlamlı bit
MSB : En yüksek anlamlı bit
RCA : Elde Dalgalı Toplama
N : Toplam bit sayısı
RHS : Sayının sağ yarısı
LHS : Sayının sol yarısı
SD : İşarestli Sayılar
MUX : Çoklayıcı
ADD : Toplayıcı devresi
SUB : Çıkarma devresi
IEEE : Elektrik ve Elektronik Mühendisleri Enstitüsü
RTL : Saklayıcı İletişim Seviyesi
ASIC : Uygulamaya Özel Tümleşik Devre
SPLD : Basit Programlanabilir Mantık Dizisi
CPLD : Karmaşık Programlanabilir Mantık Dizisi
PLD : Programlanabilir Mantık Dizisi
PAL : Programlanabilir Dizi Mantıkları
PLA : Programlanabilir Mantık Dizisi
I/O : Giriş/Çıkış
RAM : Rastgele Erişimli Hafiza
ROM : Sadece Okunabilir Bellek
ns : Nanosaniye

ÖZET

ETKİN BİT İNDİRGEME METODU KULLANARAK ARİTMETİK İŞLEM DEVRELERİ TASARIMI

Bilgisayar işlemcilerinin gün geçtikçe artan hızları aritmetik devre tasarımlarının da yüksek performanslı olarak gerçeklenmelerini gerektirmektedir.Bu gereksinim bilgisayar aritmetiğinin yeniden incelenmesine, daha hızlı algoritmaların ortaya çıkmasına ve teknolojinin sağladığı imkanlar ölçüsünde donanımsal gerçeklemelerine imkan tanıdı. Bilgisayar aritmetiğinin temel amacı, sayısal işlemlerin hızını arttıracak devrelerin ve algoritmaların tasarımıdır. Bu amaçla, tezde hızlı çarpma metodu olarak bilinen " etkin bit indirgeme metodu "na dayanan daha hızlı ve daha yüksek bit uzunluklu aritmetik çarpma devrelerinin tasarımı sunulmaktadır.

Bu tezde, etkin bit indirgeme metodu kullanarak aritmetik çarpma işlemleri için geliştirilmiş hızlı ve etkin algoritmalar incelenmiştir.Vedic matematiğine dayanan bazı çarpma yöntemlerinin üzerinde değişiklik yaparak, ayrıştırma ve bit öteleme gibi bazı temel özellikleri kullanarak geliştirilen 4 bitlik çarpma devrelerinin daha büyük bit uzunluklu devreler (8 ve 16 bit) için geliştirilmesi hedeflenmiştir. Bu algoritmalara dayanarak geliştirilen aritmetik çarpma devrelerinin tasarımı yapılmıştır.Bu amaçla, Nikhilam Sutra yöntemine dayanan yeni bir yöntem (Önerilen yöntem) geliştirilerek donanımsal çarpma devreleri tasarlanmış ve performans analizleri yapılmıştır.Tezde Önerilen yöntem, Urdhva Tiryakbhyam Sutra (Vedic matematiğine dayanan diğer algoritma) ve klasik en hızlı algoritma olan Booth yöntemi ile performans karşılaştırmaları MATLAB yazılımı aracığıyla elde edilmiştir.

Ayrıca, donanım devrelerinin tasarım ve analizlerinde etkin bir araç olarak kullanılan VHDL (VHSIC (Very High Speed Integrated Circuit) Hardware Description Language) Donanım Tanımlama Dili detaylı olarak tanıtılmıştır. Aritmetik çarpma devrelerin performans analizleri, VHDL simülasyonları aracılığıyla fonksiyonel olarak doğrulama, çıkış işareti dalga formu eldesi ve gecikme sürelerinin ölçümleri yapılarak gerçekleştirilmiştir.İncelenen bütün donanımsal devreler VHDL yardımıyla tanımlanmış ve FPGA Kullanarak sentezlenen çarpma devrelerinin performansları sunulmuştur.

Bu çalışmada, incelenen tüm aritmetik işlem devrelerinin performanslarını değerlendirmek için kullanılan kriterler olarak, (girişten çıkışa en uzun yol)gecikme süresi, kullanılan toplam standart kapı sayısı (çip alanı) ve çipte sarfolunan enerji (Verimlilik) seçilmiştir. Aritmetik devrelerin, gecikme ve alan hesaplamaları grafikler halinde verilmiştir. Sonuçların geliştirilen VHDL simülasyon programları ile uygunluk arzettiği görülmüştür.

SUMMARY

ARITHMETIC OPERATION CIRCUITS DESIGN USING EFFICIENT BIT REDUCTION METHOD

The increasing speed of computer processors with each passing day has required the design of arithmetic circuits to be verified as high performanced. For this reason; by being observed the computer artihmetic, it enabled faster algorithms to come out and verifications of hardwares in terms of the facilities that technology provides. The main aim of the computer arithmetic is the design of the circuits and algorithm that will increase the speed of numerical process. To this end, the design of arithmetic multiplication circuits with a faster and higher bit length is presented through the efficient bit reduction method in thesis.

In this thesis, the developed fast and efficient algorithms have been observed for arithmetic multiplication process by using the efficient bit reduction method. By making changes in some multiplication methods that are based on Vedic maths, the higher bit length circuits of multiplication circuits in literature which are 4 bits have been developed by using some basic properties of multiplication like decomposition and bit shifting.The design of developed arithmetic multiplication circuits has been implemented by relying on these algorithms. To this end, arithmetical multiplication circuits have been developed as fast multiplication method by being based on Urdhva Tiryakbhyam Sutra method, Nikhilam Sutra method, Booth algorithm and Proposed method. In MATLAB language, simulations of multiplication algorithms are realized and the result of the performances have been obtained.

Besides, VHDL (VHSIC (Very High Speed Integrated Circuit)) Hardware Description Language which is used a tool nowadays for hardware circuits analysis and design, is introduced in detail. Analysis of arithmetic circuits are implemented by verifying functionally with VHDL simulations, getting output signal wave form and measurements of delay time.All the circuits of hardware that are observed have been described via VHDL and the performances of multiplication circuits that are synthesized have been presented via FPGA.

The criteria which is used for examining performances of all arithmetic process circuits that are investigated in this work, are chosen as delay time, used gate number (chip area) and the energy that consumed in chip (Efficiency). Delay times and area calculations of arithmetic circuits, are depicted as graphics. It is seen that the obtained results are consistent with developed VHDL simulation programs.

1. GİRİŞ

Aritmetik, matematiğin en eski ve en basit dalıdır.Aritmetik sözcüğü Grek kökenli bir sözcük olan **ἀριθμός** (arithmos)'tan gelmektedir.Aritmetik, günlük basit işlerden ileri düzeydeki bilimsel işlem ve hesaplamalara kadar uzanan pek çok eylemin gerçekleştirilmesinde hemen herkes tarafından kullanılmaktadır. Sonuç olarak, bilgisayarlarda daha hızlı ve daha verimli bir Aritmetik Birimi'ne duyulan ihtiyaç uzun yıllardır ilgi konusu olmuştur.Temel aritmetikte en çok kullanılan dört işlem toplama, çıkarma, çarpma ve bölmedir.Çarpma temel olarak bir sayıyı başka bir sayı ile arttırmak için kullanılan matematiksel bir işlemdir.Bugünün mühendislik dünyasından bahsetmek gerekirse, çarpmaya dayalı işlemler; çok kullanılan fonksiyonlardan bir kaçıdır ve son zamanlarda Convolution, Fast Fourier Transform, Filtreleme gibi pek çok Sayısal İşaret İşleme (DSP) uygulamalarında ve Mikroişlemcilerin Aritmetik Mantık Birimi'nde (ALU) kullanılmıştır.En çok kullanılan işlem olarak çarpma işlem hızı ve düşük güçlü çarpma devresi tasarımı son yıllarda ilgi odağı olan bir konu olmuştur.

Güç tüketimini ve dijital sistemlerdeki gecikmeyi en az seviyeye indirmek icin tasarımın her aşamasında optimizasyon gerekmektedir.Bu optimizasyon (eniyileştirme), durum için en iyi algoritmayı seçmek demektir, o da tasarımın en yüksek seviyesi, daha sonra devre şekli, topoloji ve son olarak da dijital devrelerinin uygulamasında kullanılan teknoloji anlamına gelir.Bu bileşenleri esas alan farklı türlerde kullanılabilir çarpma devreleri tasarlanmıştır.

Çarpma yöntemlerinin; Mısır, Babil, Hint ve Çin uygarlıklarında kullanıldığı belgelenmiştir [1]. Bilgisayarların ortaya çıktığı ilk döneminde, çarpma genel olarak bir dizi toplama, çıkarma ve öteleme işlemiyle uygulanmıştır.Çarpma işlemini gerçekleştirmek için, literatürde önerilen pek çok algoritma vardır, bunların her biri farklı avantajlar sunmaktadır ve gecikme, devre karmaşıklığı, çipte kaplanan alan ve güç tüketimi açısından farklı performans gösterirler.

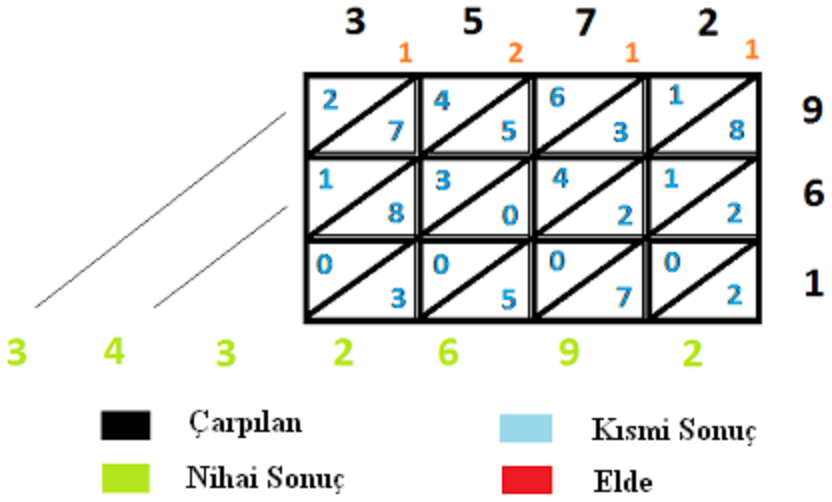
Bu çalışmada; etkin bit indirgeme metodu kullanarak aritmetik çarpma işlemleri için geliştirilmiş hızlı ve etkin algoritmalar incelenerek, algoritmaların gerçeklendiği çeşitli tasarım örnekleri ve bu algoritmaların MATLAB dilinde grafiksel kullanıcı arayüzü GUI (Graphic User Interfaces) kütüphanesi kullanılarak elde edilmiş simülasyonları sunulmaktadır.Ayrıca, incelenen bütün donanımsal devreler VHDL (VHSIC (Very High Speed Integrated Circuit) Hardware Description Language) donanım tanımlama dili yardımıyla tanımlanmış ve FPGA (Field Programing Gate Array) kullanarak sentezlenen çarpma devrelerinin simülasyonları ve performansları sunulmaktadır. Tezin 2. bölümünde, ele alacağımız çarpma yöntemleri Hint matematiğine dayandığı için eski Hint ve Vedic matematiğinden bahsedilmiştir.Ayrıca, Vedic matematiğinin formüllerine dayanan bazı klasik çarpma algoritmaları tanımlanarak örnekler üzerinde temel prensipleri verilmiştir.Tezin 3. bölümünde, İncelenen çarpma algoritmaları tanımlanarak donanımsal tasarımları ve performansları ele alınmıştır.Tezde incelenen tüm algoritmalar MATLAB ortamında yazılımları geliştirilmiş, VHDL aracılığıyla donanımsal gerçeklemeleri simüle edilmiş ve FPGA yardımıyla sentezlemeleri yapılmıştır.Bu nedenle, bu bölümde MATLAB, VHDL ve FPGA temel özellikleri verilmiştir.Tezin 4. bölümünde, 4 bit önerilen algoritmaya dayanarak 8 bit ve 16 bit uzunluklu olarak geliştirilen çarpma algoritmaları, donanımsal gerçekleme devreleri ve performans özellikleri sunulmuştur.Ayrıca, MATLAB, VHDL ve FPGA kullanılarak, tezde incelenen tüm aritmetik çarpma devrelerinin simülasyon sonuçları ve fonksiyonel olarak doğrulanmalarının yanısıra, performans analizleri sunulmaktadır. Son bölümde ise, elde edilen sonuçlar tartışılmıştır ve geleceğe dönük yapılması gereken bazı işlemler bir çalışma olarak önerilmiştir.

2. GENEL KISIMLAR

Bu bölümde Vedic matematiğinden kısaca bahsedilmiştir.Ayrıca , Vedic matematiğine dayanan çarpma işlemlerinin çeşitli algoritmaları incelenmiştir. İncelenen aritmetik işlemler olarak, yaygın kullanılan klasik Vedic çarpma aritmetik algoritmalar seçilmiştir. Bununla birlikte algoritmaların çalışma mantığının anlaşılabilmesi için çarpma algoritmaları örneklerle sunulmaya çalışılmıştır.

2.1. ESKİ HİNT MATEMATİĞİ

Indus Vadisi Uygarlığı'nın eski Hint matematikçileri çarpma işlemini gerçekleştirmek için çeşitli sezgisel numaralar kullanmıştır. Çoğu hesaplama, küçük yazı taşları üzerinde tebeşir kullanılarak yapılıyordu. Kafes çarpma, kullanılan yöntemlerden biriydi. Bu yöntemde çarpanlarla belirtilmiş satır ve sütunlardan oluşan bir tablo çiziliyordu. Tablonun her bir kutusu çapraz bir şekilde ikiye bölünüyordu. Böylece üçgen bir kafes oluşurdu. Tablonun girdileri, ondalık sayılar hâlinde yazılan kısmi çarpımlardan oluşuyordu. Daha sonra çarpım, kafesin köşegenlerindeki sayılar bir araya getirilerek oluşturulabiliyordu [2]. Bu, aşağıda Şekil 2.1'de gösterilmiştir.



Şekil 2.1 : Eski çarpma tekniği örneği

2.2. VEDİC MATEMATİĞİNİN TARİHİ

Vedic matematiği dört Veda'nın (bilgelik kitapları) bir parçasıdır. Vedic matematiği, Atharva Veda'nın bir tamamlayıcısı (upa-veda) olan Sthapatya Veda'nın (inşaat mühendisliği ve mimarlık üzerine olan kitap) bir parçasıdır. Aritmetik, geometri (düzlem, koordinat), trigonometri, ikinci dereceden denklemler, çarpanlara ayırma ve hatta hesaplarla ilgili açıklamalar vermektedir.

Kutsal Jagadguru Shankaracharya Bharati Krishna Teerthaji Maharaja (1884- 1960) bütün bu çalışmayı bir araya topladı ve çeşitli uygulamaları tartıştığı matematiksel açıklamalara da yer verdi. Swamiji, Atharva Veda'da yaptığı kapsamlı araştırmadan sonra 16 sutra (formül) ve 16 Upa sutra (alt formül) oluşturdu. Açıkça görülüyor ki, bu formüller şu anda var olan Athara Veda'nın metninde bulunmuyor çünkü bu formüller Swamiji'nin kendisi tarafından oluşturulmuştu. Vedic matematiği yalnızca matematiksel bir mucize değil ayrıca mantıklıdır da. Bu nedenle Vedic matematiği göz ardı edilemeyecek derecede bir saygınlığa sahiptir. Bu olağanüstü özelliklerinden dolayı Vedic matematiği çoktan Hindistan'ın sınırlarını aşarak diğer ülkelerde ilginç bir araştırma konusuna döndü. Vedic matematiği hem basit hem de karışık matematiksel işlemlerle ilgilenir. Özellikle basit aritmetiğin metotları oldukça yalın ve güçlüdür [4, 7]. "Vedic" kelimesi, bütün bilgilerin evi anlamına gelen "Veda" kelimesinden türemiştir. Vedic matematiği temelde aritmetik, cebir, geometri vb. gibi matematiğin dallarıyla ilgilenen 16 Sutra'dan (veya aforizmadan) oluşur. Bu Sutralar, kısa anlamlarıyla birlikte aşağıda alfabetik olarak listelenmiştir.

1) (Anurupye) Shunyamanyat – Biri oranda ise, diğeri sıfırdır.
2) Chalana-Kalanabyham – Farklılıklar ve benzerlikler.
3) Ekadhikina Purvena – Bir öncekinden bir fazla.
4) Ekanyunena Purvena – Bir öncekinden bir eksik.
5) Gunakasamuchyah – Toplamın terimleri, terimlerin toplamına eşittir.
6) Gunitasamuchyah – Toplamın çarpımı, çarpımın toplamına eşittir.
7)Nikhilam Navatashcaramam Dashatah – All from 9 and last from 10.
8) Paraavartya Yojayet – Devirme ve düzeltme.
9) Puranapuranabyham – Tamamlama veya tamamlamama ile.

10) Sankalana- vyavakalanabhyam – Toplama ve çıkarma ile.

11) Shesanyankena Charamena – Son basamağın kalanları.

12) Shunyam Saamyasamuccaye – Toplam aynı olduğu zaman o toplam sıfırdır.

13) Sopaantyadvayamantyam – Sonuncu ile sondan bir öncekinin iki katı.

14) Urdhva-tiryagbhyam – Dik ve çapraz şekilde.

15) Vyashtisamanstih – Parça ve bütün.

16) Yaavadunam – Eksikliğinin kapsamı ne olursa olsun.

Bu metotlar trigonometriye, düzlem veya küresel geometriye, koniklere, hesaplara (hem türevsel hem de integral) ve çeşitli türlerin uygulamalı matematiğine doğrudan uygulanabilir. Önceden de bahsedildiği gibi, bütün bu Sutralar geçen yüzyılın başlarında eski Vedic metinler kullanılarak yeniden oluşturulmuştur. Aynı zamanda, burada söz konusu olmayan birçok Alt-sutra da keşfedilmiştir. Vedic matematiğinin güzelliği, geleneksel matematikteki diğer ağır görünümlü hesapları çok basit bir hâle dönüştürmesinde yatar. Bu, Vedic formüllerin insan aklının doğal çalışma şartlarına dayandıkları iddia edildiği için böyledir. Bu çok ilginç bir alandır ve programlama ile dijital sinyal işleme gibi mühendisliğin çeşitli dallarına uygulanabilecek etkili bazı algoritmalar sunar [1, 3]. Çarpıcıların yapısı genel olarak üç kategoriye ayrılır.

Bunlardan ilki, donanımın üzerinde yoğunlaşan ve olabildiğince az çip alanı kullanan dizi şeklindeki çarpıcılar. İkincisi, yüksek hızda matematiksel işlemler yapan paralel çarpıcılar (sıralı veya ağaç şeklinde). Fakat bu çarpıcıların dezavantajı oldukça fazla çip alanı kullanmasıdır. Üçüncüsü ise çok zaman harcayan dizi şeklindeki çarpıcı ile yer kaplayan paralel çarpıcı arasında iyi bir alternatif olarak duran dizi-paralel çarpıcıdır.

2.3. VEDİC ÇARPMA

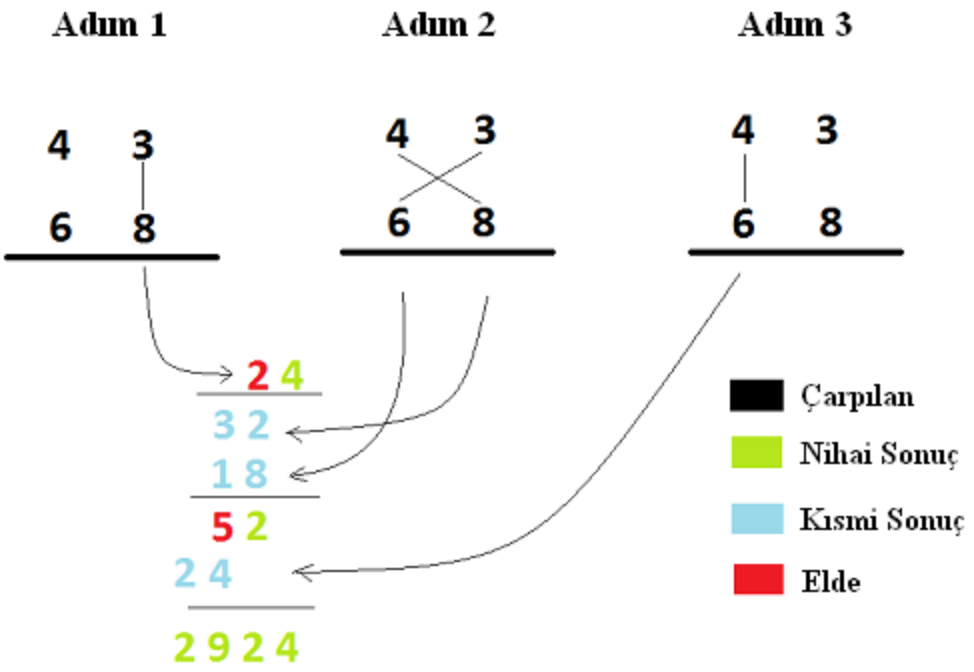
Önerilen Vedic çarpanı, Vedic çarpma formüllerini (Sutralar) temel almıştır. Bu Sutralar geleneksel olarak iki sayının çarpmasını ondalık sayı sisteminde kullanmıştır. Bu çalışmada; önerilen algoritimi dijital donanımla uygun hale getirmek amacıyla, aynı düşünceyi ikili sayı sistemine uygulayacağız. Bazı algoritimlere dayandırılan Vedic çarpma aşağıda tartışılmıştır.

2.3.1. Urdhva Tiryakbhyam Sutra

Çarpan, eski Hint Vedic matematiğinin Urdhva Tiryakbhyam (Dikey ve Çapraz) algoritimine dayanmaktadır. Urdhva Tiryakbhyam Sutra, bütün çarpma durumlarına uygulanabilecek bir genel formüldür. Kelime anlamı "Dikey ve Çapraz" sözcüklerine denk gelmektedir. Bütün kısmi sonuçların üretilmesine ve ardından bu kısmi sonuçların eş zamanlı eklemelerinin yapılmasına yardım eden yeni bir düşünceye dayanır. Böylece, kısmi sonuçların ve bunların toplamalarının üretimindeki paralellik Urdhava Tiryakbhyam kullanılarak sağlanabilir. Algoritim, n x n bit sayısı ile genellenebilir. Kısmi sonuçlar ve bunların toplamaları parallel olarak hesaplandığından, çarpan, işlemcinin saat frekansından bağımsızdır. Bu sayede, çarpan sonucu hesaplamak için aynı süreye gerek duyacak ve böylece saat frekansından da bağımsız olacaktır. Asıl avantajı, giderek yükselen saat fekanslarını yönetmek için mikro-işlemcilere duyulan ihtiyacı azaltmasıdır. Daha yüksek bir saat frekansı genellikle artan bir işletim gücüyle sonuçlanırken, dezavantajı ise aygıt yönetiminin ısısının artmasına neden olan güç dağılımını arttırmasıdır [5,7]. Çarpanın avantajı, bitlerin sayısı arttıkça, kapı gecikmesi ve alanının da diğer çarpanlara oranla daha yavaş artmasıdır. Bu yüzden, zaman, alan ve güç bakımından verimlidir. Bu yapının silicon alan/hız bakımından oldukça randımanlı olduğu gösterilmiştir [4].

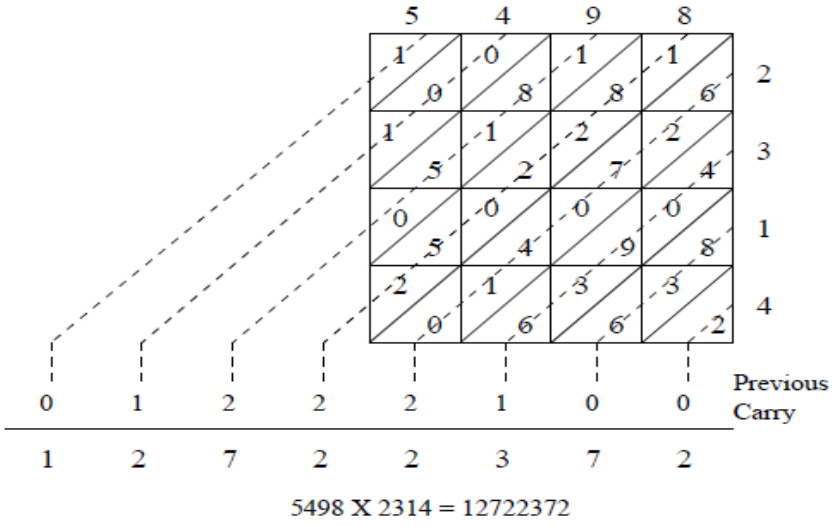
Örnek 2.1: İki ondalık sayının çarpımı :- 43 * 68

Bu çarpma tablosunu örneklemek için, iki ondalık sayının çarpımını (43*68) ele alalım. Çizginin her iki tarafindakı dijitler çarpılıp, bir önceki basamaktan kalan eldeye eklenir. Bu bir sonuç dijit ve bir de elde dijiti meydana getirir. Bu elde bir sonraki basamağa eklenir ve böylece işlem devam eder. Eğer birden fazla çizgi bir basamakta mevcut ise, bütün sonuçlar bir önceki eldeye eklenir. Her basamakta, daha yüksek dijitler bir sonraki basamak için elde olarak hareket ederken, birimin yer dijiti sonuç bit sayısı gibi hareket etmektedir. İlk olarak elde 0 olarak ele alınır. Bu algoritimin çalışma sistemi Şekil 2.2'de gösterilmiştir.



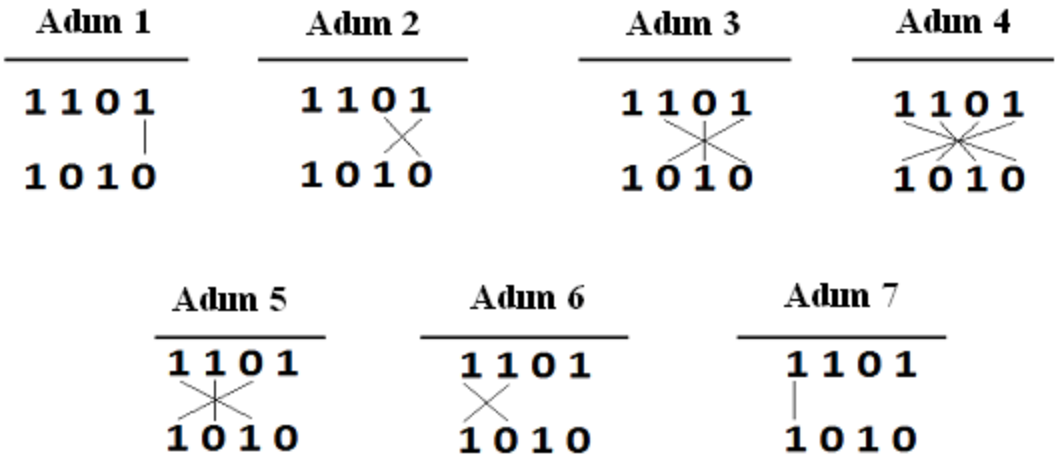
Şekil 2.2 : Urdhva Tiryakbhyam Kullanılarak 2 dijit ondalık sayının çarpımı

Urdhva Tiryakbhyam Sutra kullanılarak uygulanabilen alternatif bir çarpma yöntemi Şekil 2.3'te gösterilmiştir. Çarpılacak sayılar karenin ardışık olan iki tarafına şekilde gösterildiği gibi yazılır. Kare, her satır/sütunun çarpanın ya da çarpılanın bir dijitine denk geldiği sıralara ve sütunlara bölünür. Böylece, çarpanın her dijiti, çarpılanın bir dijitinin ortak olduğu kutucuğa sahip olur. Bu kutucuklar çapraz çizgiler ile iki eşit parçaya bölünür. Çarpanın her dijiti daha sonra bağımsız olarak çarpılanın her dijitiyle teker teker çarpılır ve iki dijitin sonucu bir ortak kutucuk içerisine yazılır. Çapraz kesik çizgilerde bulunan bütün dijitler bir önceki eldeye eklenir. Elde edilen sayının önemi en az olan dijiti, sonuç dijit olarak ve geri kalanı da bir sonraki basamak için elde olarak hareket eder.İlk basamağın eldesi (Yani, en sağ taraftaki kesik çizgi) 0 olarak ele alınır[6,7].



Şekil 2.3 : Urdhva Tiryakbhyam Sutra kullanarak yapılan alternatif bir çarpma

Şimdi bu algoritmanın ikili sayılarla nasıl kullanıldığını göreceğiz. Örnek olarak (1101 * 1010) Şekil 2.4'te verilmiştir.



Şekil 2.4 : Urdhva Tiryakbhyam'ın İkli Sayılar için Kullanılışı

İlk olarak önemi en az olan bitler çarpılır, bu da sonucun (dikey) değeri en az olan bitini verir. Daha sonrasında, Çarpılanın LSB'si çarpanın sıradaki daha büyük bir bitiyle çarpılıp, çarpanın LSB'sinin sonucuna ve çarpılanın (çapraz) sıradaki daha büyük bitine eklenir. Toplam, sonucun ikinci bitini verir ve elde, çapraz ve dikey çarpma ile değeri en az olan konumdaki iki sayının üç bitinin toplamı vasıtasıyla elde edilmiş olan sonraki

aşamanın toplamının sonucuna eklenir. Ardından, bu dört bitin tamamına, toplam ve eldeye ulaşmak amacıyla çapraz çarpma ve toplama işlemi uygulanır. Toplam, sonucun uygun biti olur, elde ise yeniden LSB haricindeki üç bitin sonraki aşama çarpımına ve toplamasına eklenir. Aynı işlem, sonucun MSB'sini vermesi amacıyla iki MSB'nin çarpımına dek devam eder. Örneğin; herhangi bir orta basamakta 110 bulunmaktaysa; 0 sonuç, 11 ise elde görevini görür. Eldenin çoklu-bit sayısı olma ihtimaline açık bir şekilde dikkat çekilmelidir.

Böylece aşağıdaki ifade elde edilir:

$r_0=a_0b_0;$ (1)

$c_1r_1=a_1b_0+a_0b_1;$ (2)

$c_2r_2=c_1+a_2b_0+a_1b_1+a_0b_2;$ (3)

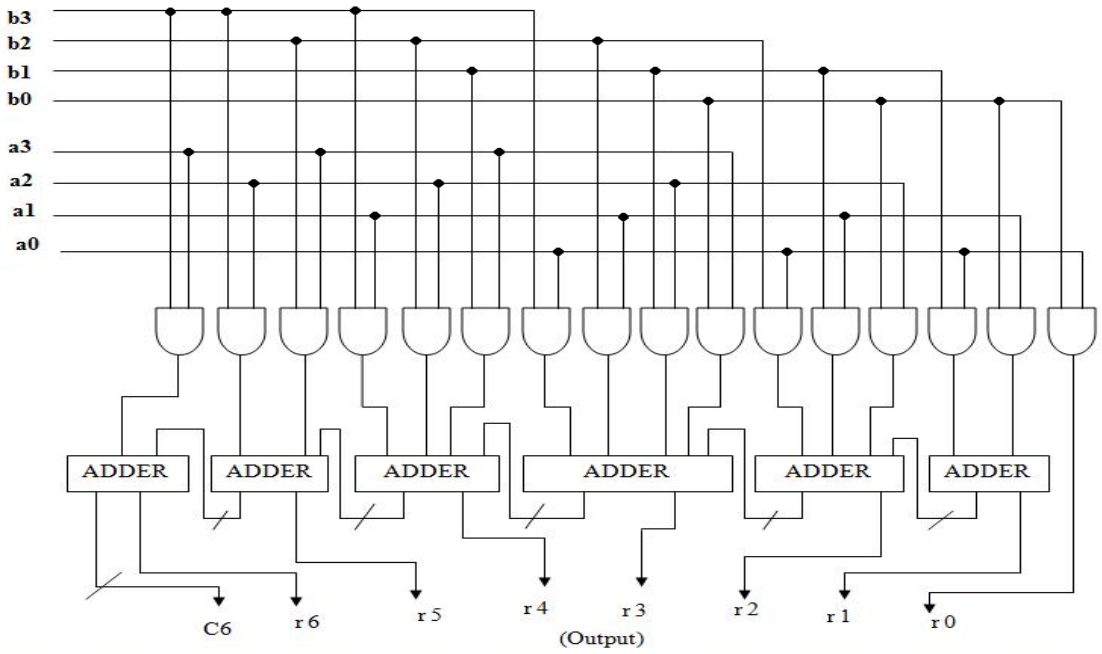
$c_3r_3=c_2+a_3b_0+a_2b_1+a_1b_2+a_0b_3;$ (4)

$c_4r_4=c_3+a_3b_1+a_2b_2+a_1b_3;$ (5)

$c_5r_5=c_4+a_3b_2+a_2b_3;$ (6)

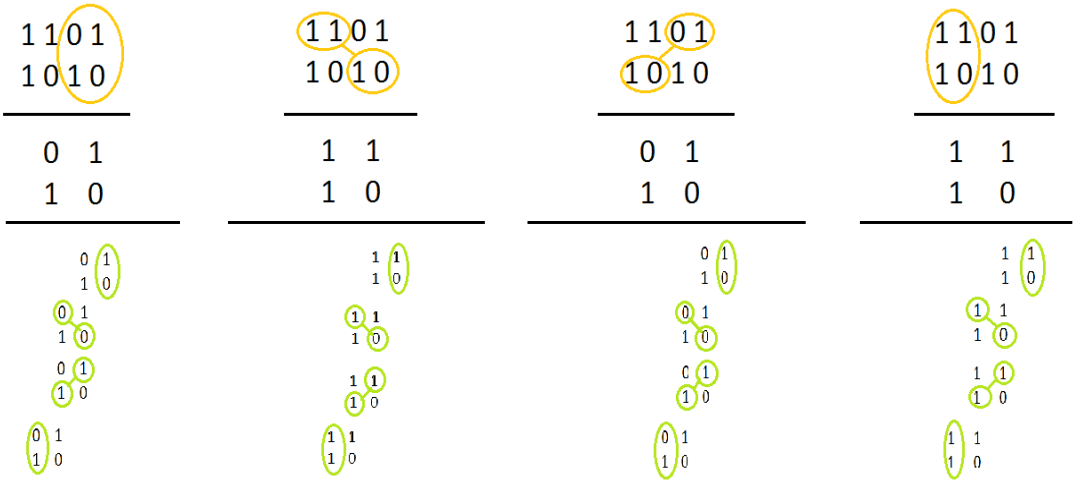
$c_6r_6=c_5+a_3b_3$ (7)

Nihai sonuç olarak $c_6r_6r_5r_4r_3r_2r_1r_0$'a varılır. Bu yüzden, bu bütüm çarpma durumlarına uygulanabilecek genel bir matematik formülüdür [7].



Şekil 2.5 : 4-bitlik Urdhva Tiryakbhyam çarpanın donanımsal yapısı [7]

Buradan bir şeyi gözlemlemektyiz. Bitlerin sayısı arttıkça, elde iletim gecikmesi de artar ve son katta bir elde dalgalı toplama (RCA) oluşturulur. Urdhva Tiryakbhyam'ın daha verimli bir kullanımı Şekil 2.6'da gösterilmiştir.



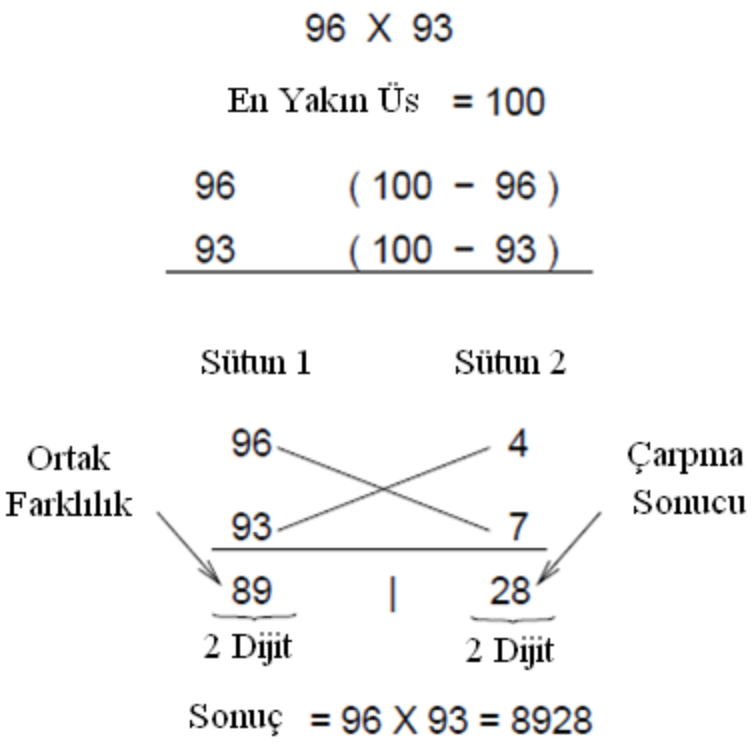
Şekil 2.6 : Urdhva Tiryakbhyam'ın İkili sayılar için daha verimli bir uygulması

Yukarıda, bir 4x4 bit çarpma işlemi, 4 adet 2x2 bit çarpma işleminin paralel olarak hesplanmasına indirgenmiştir. Bu lojik seviye sayısını azaltarır ve böylece çarpanın gecikmesi de azalır. Bu örnek, Urdhva Tiryakbhyam Sutra'nın daha iyi sonuç veren parallel uygulanmasını göstermektedir. Bu algoritma, daha büyük bir sayının (N x N, N bitlerinin her biri) çarpımının onu daha küçük değerlere (N/2 = n) parçalayarak nasıl yapılacağını göstermektedir ve bu küçük sayılar da (2x2)'nin çarpılan değerine ulaşana dek daha küçük sayılara (her biri n/2) tekrar bölünür. Böylece, bütün çarpma işlemi basitleştirilip, işlemin hızı arttırılır [7,8].

Şekil 2.5 'teki devrenin tasarımı, nihai sonuca ulaşmak için bir dizi toplayıcıya ihtiyaç duyulan meşhur dizi çarpıcı (Array multiplier) tasarımına çok benzemektedir. Bütün kısmi sonuçlar paralel olarak hesaplanır ve bunlarla ilişkili olan gecikmeler temel olarak çarpmanın dizilimini oluşturan toplayıcılar vasıtasıyla eldenin yayılımı için geçen zamandır. Açıkça, bu yayılma gecikmesinden dolayı büyük sayıların çarpımı için etkili bir algoritma değildir. Bu problemi çözmek için, iki büyük sayının çarpımında etkili bir yöntem sunan Nikhilam Sutra'yı ele alacağız [7].

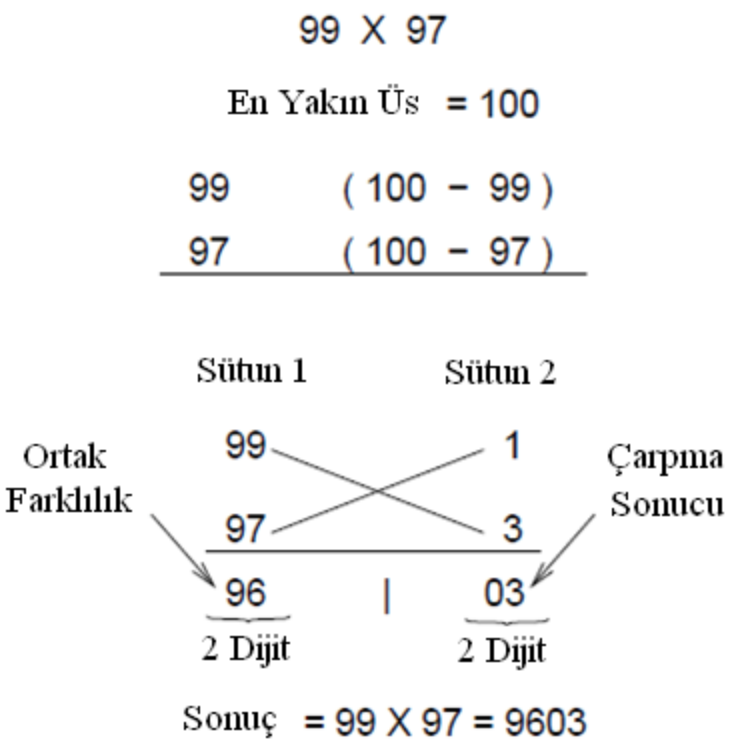
2.3.2. Nikhilam Sutra

Nikhilam Sutra kelimesinin tam manasıyla "all from 9 and last from 10"demektir.Yani temel olarak en sol dijitten başlayarak her dijitten 9 çıkartılıp, son dijitten ise 10 çıkartılması anlamına gelir [7,9]. Her ne kadar Nikhilam Sutra bütün çarpma işlemlerine uygulanabilir olsa da, asıl olarak sayılar büyük olduğunda etkilidir.Sutra algoritması sayıların çarpımında iki farklı yönteme dayanır.Birincisi çarpma işleminde iki sayıya en yakın üssü bulması, ikincisi de çıkarma yöntemidir. Sutrada asıl sayı ne kadar büyükse, çarpma işleminin karmaşıklığı da o kadar az olur. Sutra'yı, verilen sayıların ikisinden de daha büyük en yakın üslünün 100 olduğu iki ondalık sayının (96 * 93) çarpımını ele alarak örneklendireceğiz [7].



Şekil 2.7 : Nikhilam Sutra Kullanılarak Yapılan Çarpma

Çarpmanın sağ yarısı (RHS) sadece 2. Sütundaki sayıların çarpılmasıyla (7*4=28) bulunabilir. Çarpmanın sol yarısı (LHS) ise 2. Sütundaki ikinci sayının, 1. Sütundaki ilk sayıdan ya da tam tersinin çaprazlama bir şekilde çıkarılmasıyla bulunabilir[7].Örneğin: 96 - 7 = 89.



Şekil 2.8 : Nikhilam Sutra Kullanılarak Yapılan Çarpma

Bu nedenle iki n- bit sayının çarpımı onların üslerinin çarpımına indirgenebilir. Bu indirgemenin avantajını kullanmak için, üslerini aldıktan sonra elde edilen sayılar, asıl sayılardan daha küçük olduğundan emin olunmalıdır. Bu durum ancak iki asıl sayı 10^n / 2, örn: x > 10^n / 2'den daha büyük olduğunda doğru sonuçlar verecektir. Bu, Nikhilam Sutra'nın küçük sayılardansa, neden büyük sayıların çarpımında daha etkili olduğunun nedenidir [2,7,10].

Sutranın ikinci yöntemini anlatacak olursak,
7 ve 8 sayılarını çarpmak istersek şu adımları yaparak çarpma işlemi gerçekleştirilebilir:-

1) İki sayıda bir dijit olduğu için son dijit olarak algılanır ve iki sayıdan da 10 sayısı çıkartılır.
2) Çarpmanın sağ yarısı (RHS) sadece 2. Sütundaki sayıların çarpılmasıyla bulunabilir.
3) Çarpmanın sol yarısı (LHS) ise 2. Sütundaki ikinci sayının, 1. Sütundaki ilk sayıdan ya da tam tersinin çaprazlama bir şekilde çıkarılmasıyla bulunabilir [9].

Örnek 2.2: (7) x (8) = 56

(1)

7	-3
8	-2

(2)

7	-3
8	-2
	6

(3)

7	-3
8	-2
5	6

Örnek 2.3: (995) x (988) = 983060

(1)

995	-5
988	-12

(2)

995	-5
988	-12
	060

(3)

995	-5
988	-12
983	060

Örnek 2.4: (89) x (92) = 8188

(1)

89	-11
92	-08

(2)

89	-11
92	-08
	88

(3)

89	-11
92	-08
81	88

3. MALZEME VE YÖNTEM

Bu bölümde, etkin bit indirgeme metodu kullanan aritmetik çarpma işlemlerine ait çeşitli algoritmalar incelenmiştir. İncelenen aritmetik çarpma işlemler olarak, yaygın kullanılan Vedic çarpma algoritması ve klasik Booth çarpma algoritması seçilmiştir. Bununla birlikte algoritmaların çalışma mantığının anlaşılabilmesi için çarpma algoritmaların örnekler üzerinde temel prensipleri, donanımsal gerçekleme devreleri ve performans özellikleri verilmiştir. Fakat Vedic matemetiğine dayanan bizim geliştirdiğimiz Önerilen (Proposed) algoritması ise detaylı olarak bir sonraki bölümde anlatılmaktadır.

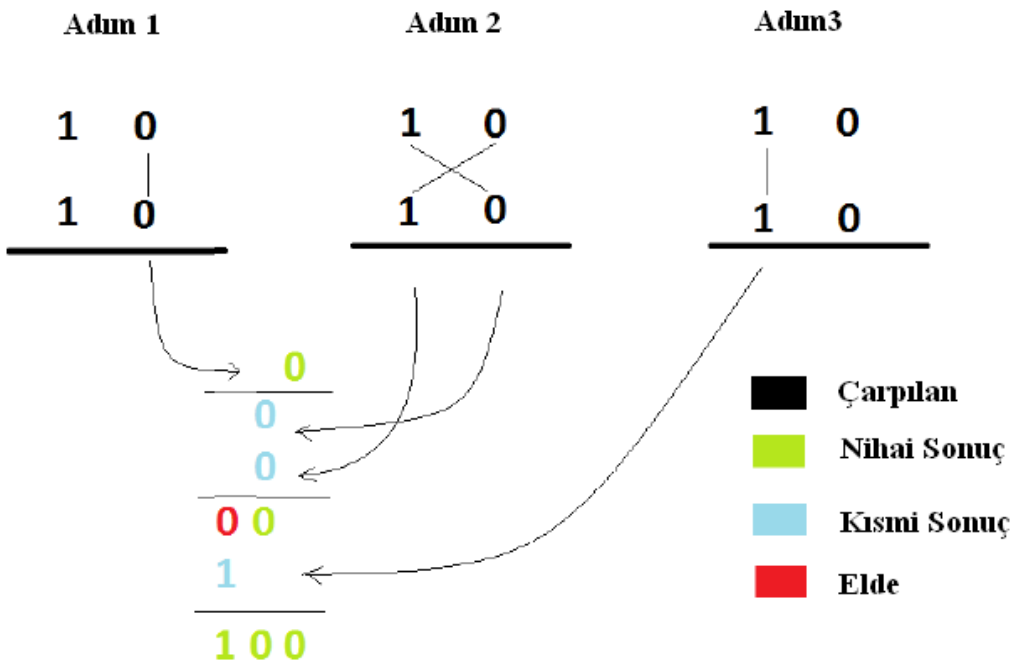
Ayrıca, bu çalışmada çarpma işlemlerinde kullanılan algoritmalar MATLAB ve VHDL (VHSIC (Very High Speed Integrated Circuit) Hardware Description Language) programlama dillerini kullanılarak gerçeklenmiştir.MATLAB (Sürüm 7.6) dilinde grafiksel kullanıcı arayüzü GUI(Graphic User Interfaces) kütüphanesi kullanılarak çarpma algoritmalarının simülasyonları için ara yüzü gerçekleştirip performans sonuçları elde edilmiştir. VHDL ile yazılan programları MAX+plus II(Sürüm 10.2) , QUARTUS II (Sürüm 10.0 sp1) , XILINX ISE (Sürüm 12.4i) kullanılarak derlenmiş ve simülasyon sonuçları elde edilmiştir.Bu uygulamalar XILINX (www.xilinx.com/ise), ALTERA (www.altera.com) firmaları tarafindan geliştirilmiş olup halen tasarımcılar tarafindan yaygın olarak kullanılmaktadır.Çarpma algoritmalarını derlendikten ve simülasyon sonuçlarını elde ettikten sonra FPGA kiti üzerinde sentezleme yapıp ve tekrardan bu kez FPGA kiti üzerinde algoritmaların simülasyon sonuçları elde edilip performansları incelenmiştir.

3.1. VEDİC ÇARPMA ALGORİTMASI

Vedic çarpma algoritması için, 2x2 bit ilk temel çarpım blokları oluşturulur ve sonrasında bu bloklar kullanılarak 4x4 çarpım bloğu oluşturulur, ardından da 4x4 çarpım bloğu kullanılarak 8x8 bit çarpım bloğu oluşturulur ve son olarak 16x16 bit Çarpım sonucu elde edilir [7].

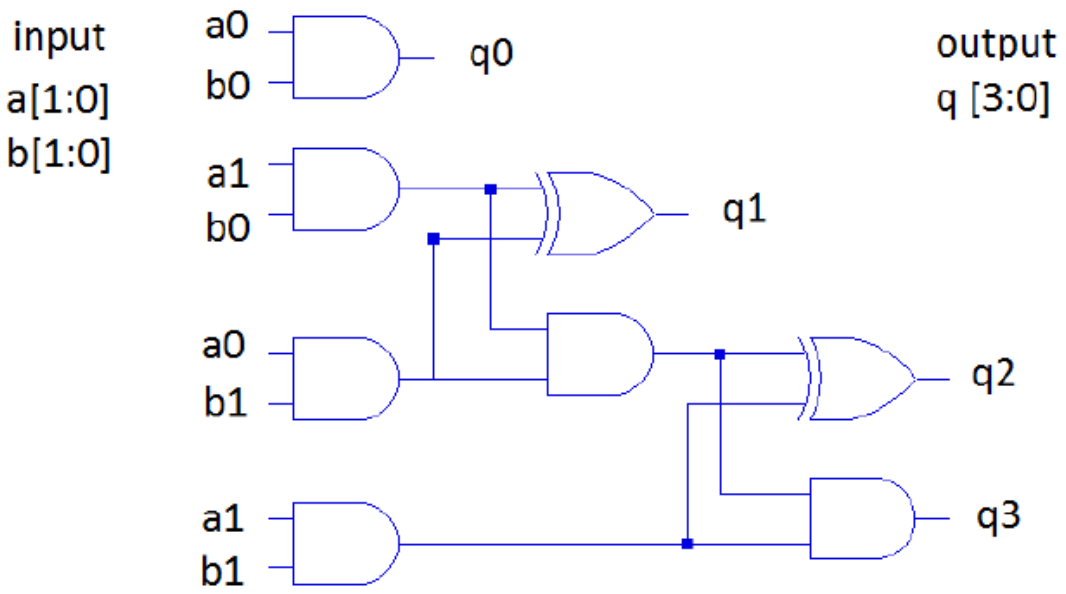
3.1.1. 2x2 bit Urdhva Çarpımı

2x2 bit urdhva çarpımında, çarpılan ve çarpan herbiri 2 bite ve çarpmanın sonucu da 4 bite sahiptir. Bu nedenle giriş kısmında girişler (00)'dan (11)'e ve çıkış ise (0000, 0001, 0010, 0011, 0100, 0110, 1001) serisi şeklinde değerler alır [10]. Buna dayanarak, Şekil 3.2'de basit bir tasarım verilmiştir. Urdhva Tiryakbhyam kullanılarak çarpma Şekil 3.1'da gösterildiği gibi gerçekleşir. Burada çarpılan a ve çarpan b her ikisi de (10) olacak şekilde seçilmiştir.Çarpmadaki ilk basamak çarpan ve çarpılanın LSB'sinin dikey çarpımıdır, 2. basmak ise çapraz çarpım ve kısmi sonuçların toplanmasıdır. 3. basamak, çarpan ve çarpılanın MSB'sinin dikey çarpımını ve 2. basamaktan aktarılan eldenin toplanmasını içermektedir.



Şekil 3.1 : 2x2 bit Urdhva yöntemi kullanılarak ikili sayının çarpımı

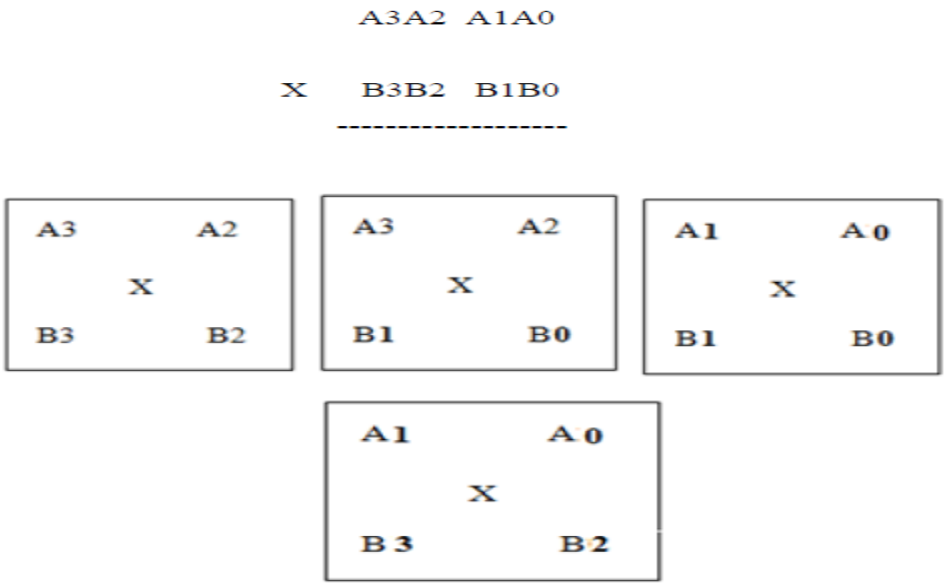
2x2 bit Urdhva çarpım işlemini gerçekleyen devre Şekil 3.2'te gösterilmiştir.Basitlik adına, saat (clock) ve saklayıcıların (registers) kullanımı gösterilmemiştir, bununla beraber algoritmanın anlaşılması üzerinde vurgu yapılmıştır.



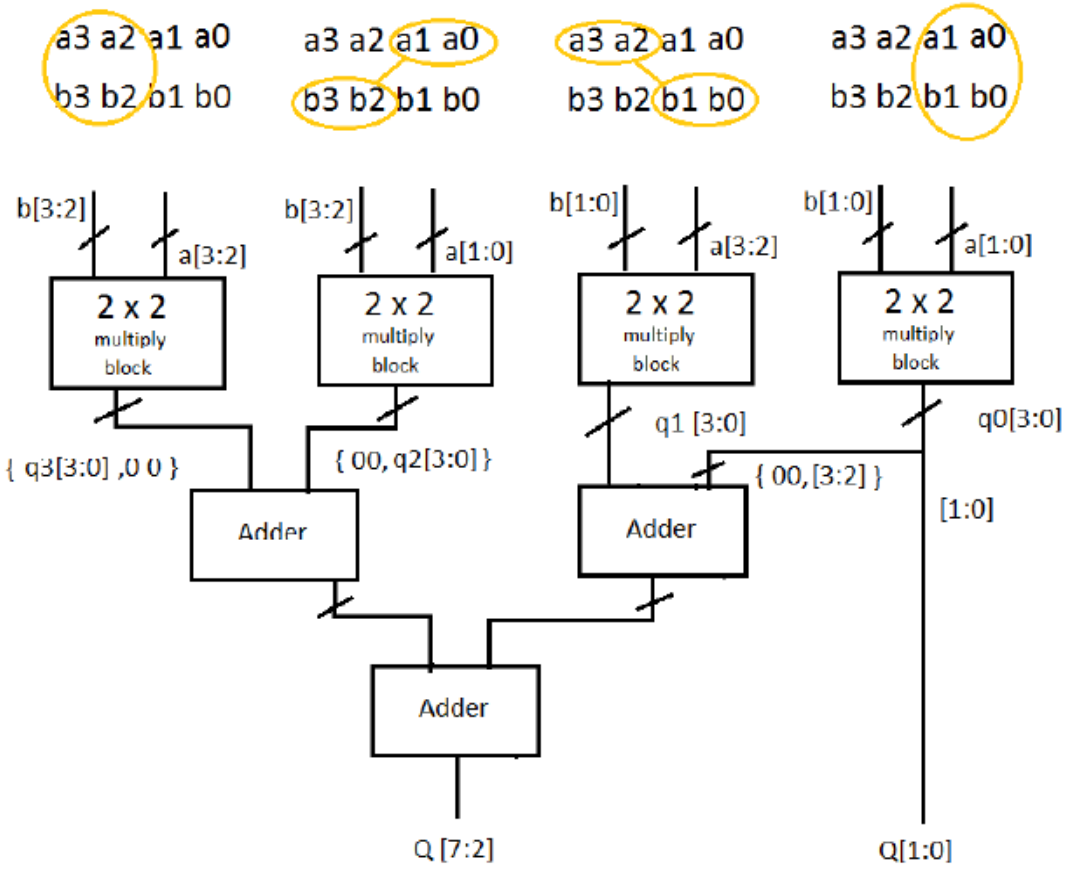
Şekil 3.2 : 2x2 bit Urdhva bloğunun donanımsal yapısı

3.1.2. 4x4 Bit Urdhva Çarpımı

4x4 bit çarpımı, 4 adet 2x2 çarpım bloğunun kullanılmasıyla oluşturulur. Burada, sonuç 8 bit uzunluğunda, çarpan ve çarpılan (n=4) bit uzunluğndadır. a ve b olan her iki sayı n/2 =2 uzunluklu daha küçük parçalara bölünür. Bu yeni oluşan 2 bitin parçaları, 2x2 çarpım bloğuna giriş olarak verilmiştir ve 2x2 çarpım bloğundan elde edilen 4 bitlik bir sonuç Şekil 3.3 ve Şekil 3.4'te görüldüğü üzere toplama için toplama ağacına gönderilir[7,10].



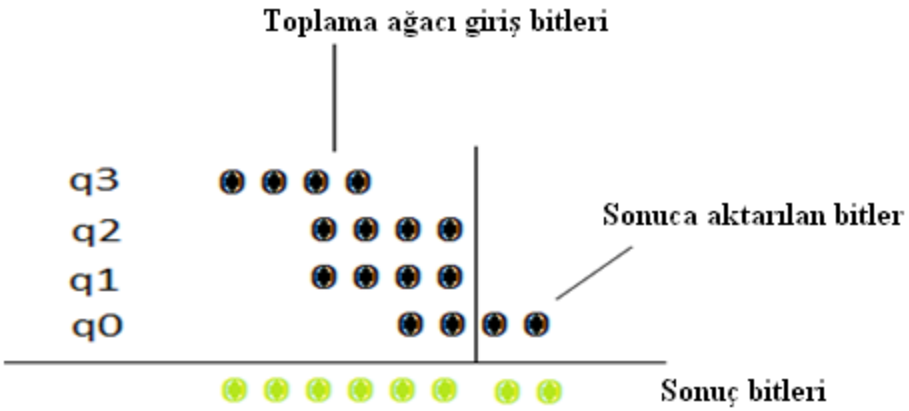
Şekil 3.3 : 4x4 bit Urdhva Çarpım Algoritması



SONUÇ = Q (7:2) & Q (1:0)

Şekil 3.4 : 4x4 bit Urdhva çarpımı blok diyagramı

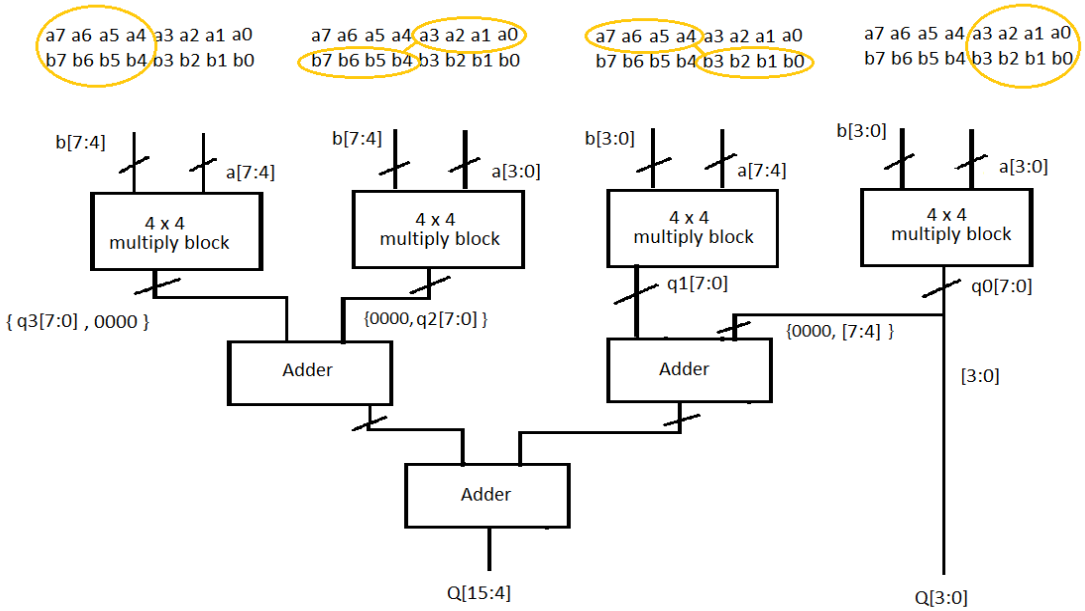
Burada, ardışık toplama yerine toplama ağacı Wallace ağacına benzer şekilde değiştirilmiştir.Böylece toplama seviyesi 3'ten 2'ye düşürülmüştür. Burda, q0'ın daha yüksek bitleri toplama ağacında kalırken, q0'ın daha düşük iki biti doğrudan çıkışa geçer. Toplama ağacına eklenen bitler daha sonara Şekil 3.5'te diyagramla gösterilebilir.



Şekil 3.5 : 4x4 bit Urdhva blokundaki kısmi sonuçların toplamı

3.1.3. 8x8 Bit Urdhva Çarpımı

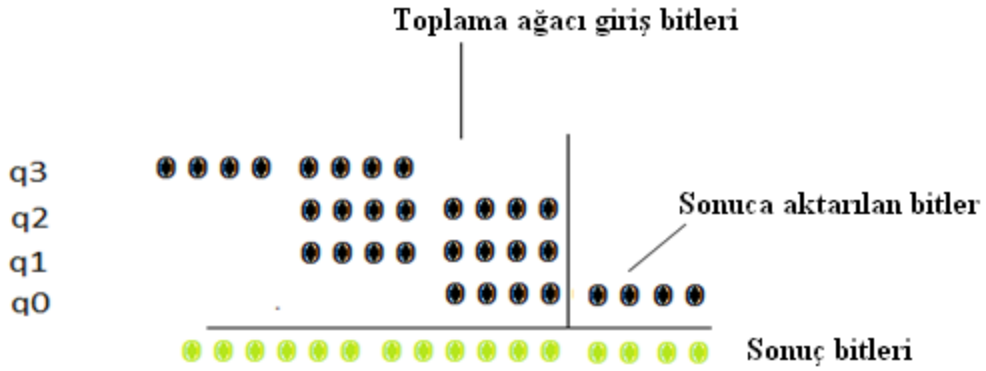
8x8 bit Çarpımı, 4 adet 4x4 çarpım bloğunun kullanılmasıyla oluşturulur. Burada, sonuç 16 bit uzunlukta, çarpan ve çarpılan (n=8) bit uzunluktadır. Tıpkı 4x4 çarpma bloğu durumundaki gibi, a ve b sayıları n/2 = 4 bit uzunluktaki daha küçük parçalara bölünür. Bu yeni oluşan 4 bitlik parçalar,giriş olarak 4x4 çarpan bloğuna eklenir ve yine burada bu yeni parçalar n/4 = 2 bit uzunluklu daha da küçük parçalara bölünüp 2x2 çarpma bloğuna eklenir. Elde edilen sonuç, 8 bit uzunluklu 4x4 çarpma bloğunun çıkışından, Şekil 3.6'da gösterildiği gibi toplama için bir toplama ağacına gönderilir [8,10].



SONUÇ = Q (15:4) & Q (3:0)

Şekil 3.6 : 8x8 bit Urdhva çarpımı blok diyagramı

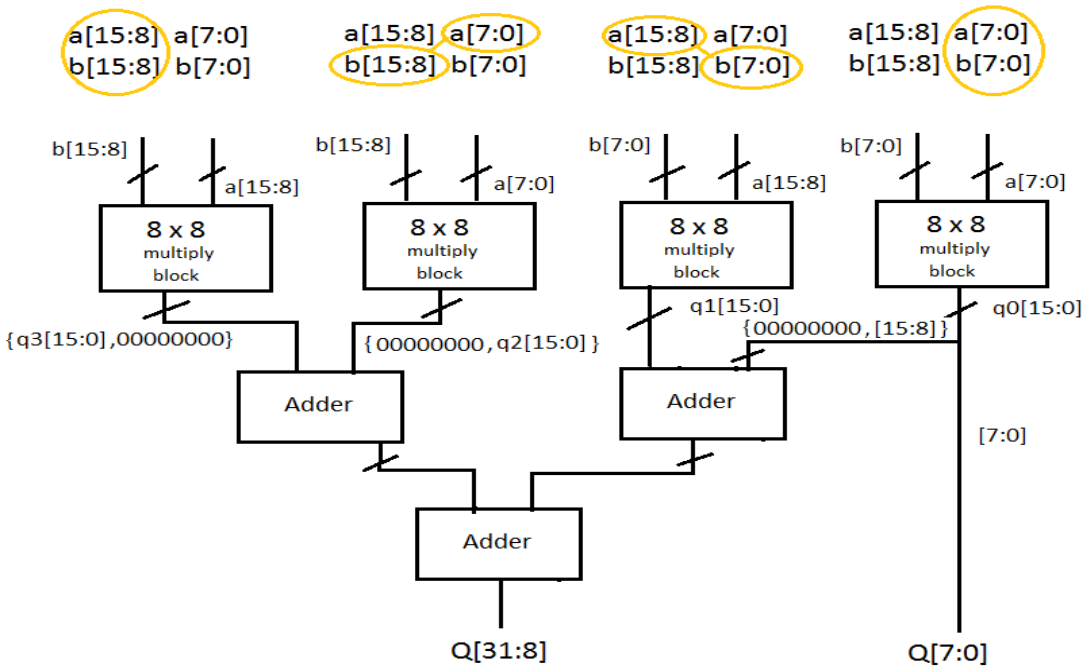
Burada, her 4x4 çarpma bloğunu Şekil 3.4'te gösterildiği gibi oluşturduğu aşikardır. 8x8 çarpma bloğunda, q0'ın daha düşük 4 biti doğrudan çıkışa, geri kalan bitleri de toplama için Şekil 3.6'da gösterildiği gibi toplama ağacına eklenir. Kısmi sonuçların toplamı Şekil 3.7'de gösterilmiştir.



Şekil 3.7 : 8x8 bit Urdhva bloğundaki kısmi sonuçların toplamı

3.1.4. 16x16 Bit Urdhva Çarpımı

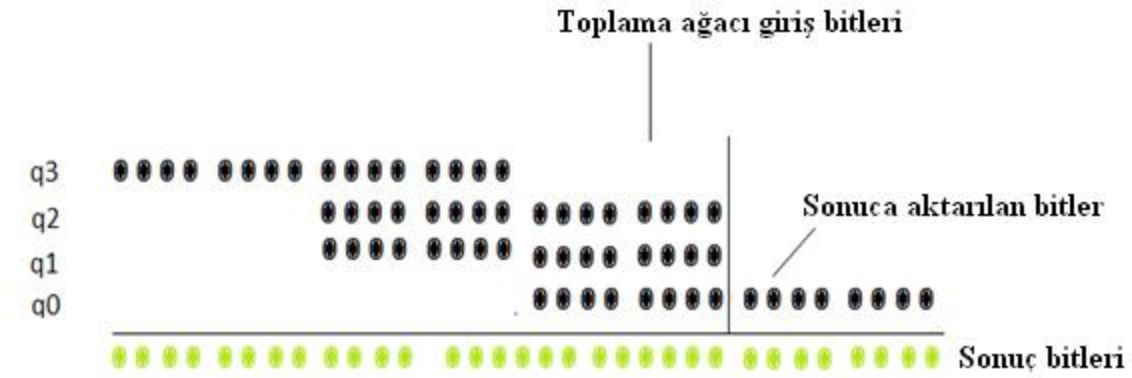
16x16 bit Çarpım işlemi, 4 adet 8x8 çarpım bloğu kullanarak elde edilir. Burada, sonuç 32 bit uzunluklu, çarpan ve çarpılan sayılar (n=16) bit uzunlukludur. a ve b olan her iki giriş n/2 = 8 bit uzunluklu daha küçük parçalara bölünür. Bu yeni oluşturulan 8 bitlik parçalar 8x8 çarpım bloğuna giriş olarak eklenir ve yeniden bu yeni oluşan parçalar, 8x8 çarpma bloğunda olduğu gibi, n/4 = 4 bit uzunluklu daha da küçük parçalara bölünüp 4x4 çarpma bloğuna eklenir. Yine, yeni parçalar 2 bit uzunluklu parçalar elde etmek için 2x2 çarpım bloğuna eklenecek olan iki yarıya bölünür. Üretilen sonuç, 16 bit uzunluğunda 8x8 bit çarpma bloğunun çıktısından, Şekil 3.8'de gösterildiği gibi toplama için toplama ağacına gönderilir [8,10].



SONUÇ = Q (31:8) & Q (7:0)

Şekil 3.8 : 16x16 bit Urdhva çarpımı blok diyagramı

Burada, Şekil 3.8'de gösterildiği gibi, q0'ın daha yüksek bitleri toplama için toplama ağacına eklenirken, daha düşük 8 biti doğrudan sonuç kısmına geçer. Kısmi sonuçların toplamı Şekil 3.9'da gösterilmiştir.



Şekil 3.9 : 16x16 bit Urdhva bloğundaki kısmi sonuçların toplamı

3.2. BOOTH ÇARPMA ALGORİTMASI

Booth çarpma algoritması işaretli sayıların çarpımında çok etkili olan bir algoritmadır. Hem negatif hem de pozitif sayılar üzerinde aynı şekilde işlem yapabilir [11,12].

Standart topla/ötele algoritmaları için, her bir çarpan biti kısmi toplama eklenmek üzere bir çarpılan değeri üretir. Bu tür işlemlerde çarpanın değeri arttıkça daha çok çarpılan sayı değerinin toplanma zorunluluğu ortaya çıkmaktadır. Bu durumda çarpım işlemindeki gecikme değeri gerçekleştirilmiş olan toplama işlemlerinin sayısına bağlıdır. Eğer buradaki toplama işlemlerinin sayısını düşürebilecek bir yol bulunursa, bu performansı çok daha artıracaktır [13,14,15].

Örnek 3.1:

	110011	(Çarpılan) = 51
x	010111	(Çarpan) = 23
	110011	topla (çarpan(0) = 1)
	110011	topla (çarpan(1) = 1)
	110011	topla (çarpan(2) = 1)
	000000	ötele (çarpan(3) = 0)
	110011	topla (çarpan(4) = 1)
+	000000	ötele (çarpan(5) = 0)
	10010010101	= 1173

Booth algoritması çarpılan sayıları azaltan bir metotdur. Belli bir aralıkta sunulan sayıyı daha yüksek tabanlı bir sayı olarak sunarak basamak sayısının azaltılmasını sağlar. Bu durumda k bitlik bir sayı k/2 basamak olarak 4'lük tabanda bir sayı, k/3 basamak olarak 8'lik tabanda bir sayı gibi yorumlanabilmektedir. Böylece yüksek tabanlı çarpma işlemi yaparak herbir çevrimde birden fazla çarpan ile işlem yapmak mümkün olmaktadır. Booth çarpma algoritması çarpma işleminde toplanarak çarpım sonucu bulunacak kısmi çarpımların elde edilmesinde kullanılır [11,13]. Booth algoritması birbirini takip eden sıfir ve birlerden oluşan gruplar için daha az kısmi sonuç oluşturulmasını sağlamaktadır. Çarpandaki birbirini takip eden sıfirlar için yeni bir kısmi toplam oluşturulmasına gerek yoktur. Sadece daha önceden hesaplanmış olan kısmi sonucun bitlerinin her bir sıfir için bir bit sağa ötelenmesi yeterlidir. Çarpandaki birlerden oluşan grup için ,...0{11..11}0... m 'den (m çarpandaki birlerden oluşan grubu gösterir) daha az yeni kısmi sonuç oluşturabilir. Yukarıdaki dizi bir biti sıfir olmayan iki bit dizisinin farkına eşit olan aşağıdaki ifadeye eşittir [16].

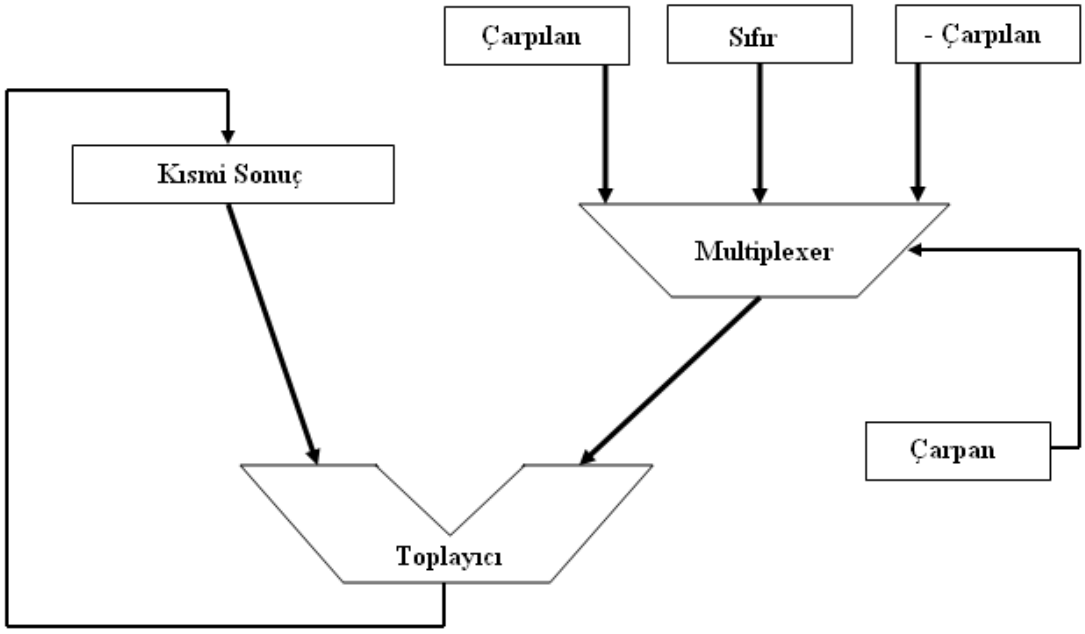
...0{11..11}0... = ...1{00..00}0... - ...0{00..01}0...

yukarıdaki gösterim işaretli rakam (SD – sign digit) kullanılarak dizi ...0{11..01}0...olarak yazılabilir.

(111111) = 1(000000) – (000001) $\longrightarrow$ 1(00000 $\bar{1}$)

$(63)_{10} = (64)_{10} - (1)_{10}$

Yukarıda olduğu gibi m kısmi sonuçları oluşturmak yerine sadece iki kısmi sonuç oluşturulabilir.Bu işleme SD kodda çarpanların yeniden kodlanması denir.



Şekil 3.10 : BOOTH algoritmasının donanımsal yapısı

Tablo 3.1 : BOOTH algoritması için kodlama tablosu

X_i	X_{i-1}	İşlem	Z_i
0	1	Topla ve Ötele	1
1	0	Çıkar ve Ötele	1'
1	1	Sadece Ötele	0
0	0	Sadece Ötele	0

Başlangıçta " 0 " değeri çoğu çarpan sayının sağ tarafına eklenir. Böylece tabloya göre 2'şer bitlik kodlama doğru bir şekilde yapılabilmektedir. Bu eşitlik şu şekilde sunulabilir .

$$Z_i = X_{i-1} - X_i \tag{3.1}$$

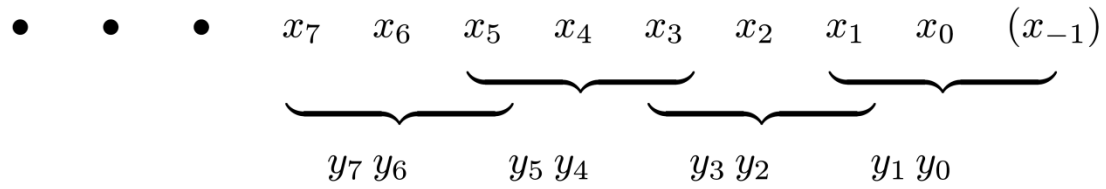
En basit yeniden kodlama tekniği Booth algoritmasıdır. Bu algoritmada çarpan x_{n-1} $x_{n-2}\ldots$ x_1 x_0 'ın x_i biti ve bir önceki biti x_{i-1}, yeniden kodlanmış çarpan z_{n-1} $z_{n-2}\ldots$ z_1 z_0'ın i. bitini oluşturmak için sırayla incelenir. Önceki biti x_{i-1} burada sadece referans biti olarak kullanılır. Buna göre x_{i-1}, x_{i-2} referans biti olarak kullanılarak yeniden kodlanmış

z_{i-1} elde edilir. i = 0 için referans biti x_{i-1} 0 olarak tanımlanır. Yeniden kodlanmış bitin hesaplanmasının kolay yolu $z_i = x_{i-1} - x_i$ 'dir [12].

0011110011 (0) çarpanının 01000 1' 0010 1' olarak yeniden kodlanırsa 6 yerine 4 tane toplama / çıkarma işlemi elde edilir. Parantez içinde gösterilen sıfır, x_0 için x_{i-1} referans bitidir .

Booth algoritmasında iki sorun vardır. Birincisi, toplama / çıkarma işlemlerinin sayısı değişkendir ve iki birbirini takip eden toplama / çıkarma arasındaki öteleme işlemlerinin sayısı da değişkendir. Bunlar, eş zamanlı çarpanlar tasarlanırken bazı zorluklara neden olur. Bir diğer sorun ise ayrık 1 'lerin olması durumunda ortaya çıkar. Mesela " 001010101 (0) " sayısı yeniden kodlanınca ortaya " 0 1 1' 1 1' 1 1' 1 1' " sayısı çıkar ve buda 4 yerine 8 işlem gerektirir [12,14].

Bu durum çarpan X 'in aynı zamanda iki yerine 3 bitinin incelenmesi ile aşılabilir. x_i ve x_{i-1}, z_i ve z_{i-1} 'e yeniden kodlanır. x_{i-2} referans biti olarak kullanılır. ayrı bir aşamada x_{i-2}, x_{i-3} ve x_{i-4} referans biti yardımı ile z_{i-2} ve z_{i-3} bitlerine yeniden kodlanır. böylece üç bitlik grupların her birisi üst üste gelir, en sağdaki x_1 x_0 (x_{i-1}) olur ve bu şekilde devam eder. Değiştirilmiş Booth algoritmasında çarpan bitleri aşağıdaki gibi incelenerek bu sorun aşılabilir .



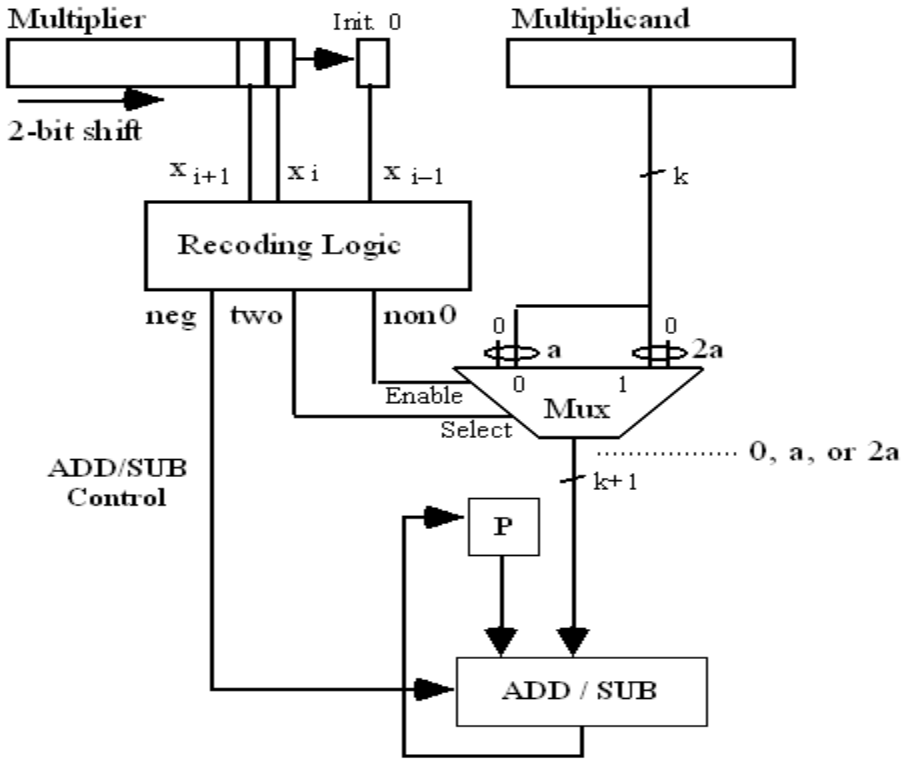
2'li tabanda yapılan Booth Recoding işlemi modern aritmetik devrelerde direk olarak uygulanmamaktadır. Bu tabanda yapılan Booth Recoding işlemi daha yüksek tabanlı sayılarda yapılan recoding işleminin daha kolay anlaşılabilmesini sağlar. Bunun için recoding işlemi için 4 tabanı tercih edilir. Aşağıda 2'li tabanı 4'lü tabana dönüştürümü verilmiştir [11].

Tablo 3.2 : BOOTH algoritması 4 tabanında yeniden kodlama tablosu

X_{i+1}	X_i	X_{i-1}	Yorumlar	Y_i	Y_{i-1}	Z_i
0	0	0	0 Dizisi	0	0	+0A
0	0	1	1 Dizisinin Sonu	0	1	+A
0	1	0	Tek 1	0	1	+A
0	1	1	1 Dizisinin Sonu	1	0	+2A
1	0	0	1 Dizisinin Başlangıcı	1'	0	-2A
1	0	1	Tek 0	0	1'	-A
1	1	0	1 Dizisinin Başlangıcı	0	1'	-A
1	1	1	1 Dizisi	0	0	+0A

Başlangıçta " 0 " değeri çoğu çarpan sayının sağ tarafına eklenir. Böylece tabloya göre 3'er bitlik kodlama doğru bir şekilde yapılabilmektedir. Bu eşitlik şu şekilde verilebilir.

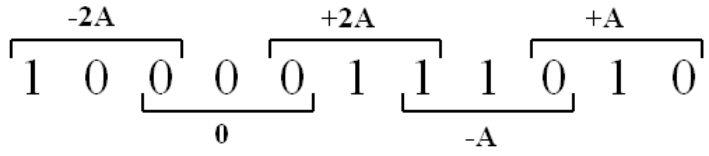
$$Z_i = (Y_i \quad Y_{i-1}) = -2X_{i+1} + X_i + X_{i-1} \tag{3.2}$$



Şekil 3.11 : 4 Tabanında değiştirilmiş Booth algoritmasının donanımsal yapısı

Şekilde gördüğümüz gibi 4 tabanında değiştirilmiş Booth algoritmasının donanımsal yapısı görülmektedir.Burada "neg" topla/çıkar kontrol bitini gösterir, eğer "neg"=1 çıkarma işlemi yapılır(-A) ,"neg" = 0 toplama işlemi yapılır (+A). "two" tek/çift çarpan kontrolünü gösterir, eğer 1'e eşitse Mux devresinden (2a) ,0 'e eşitse (a) çıkış yapmaktadır. "non0" ise Mux devrsinin "Enable" kısmını kontrol etmektedir.Eğer "non0" = 1se (a) veya (2a) , 0 ise (0) değerileri giriş yapmaktadır .

İnceleyeceğimiz örnekte çarpan 1000011101 olarak alınmıştır. Sağ tarafa " 0 " eklenmiş gösterim 10000111010 (eklenen " 0 " değeri) 3'er basamak seçilerek elde edilen gösterim aşağıdaki gibidir .



Şekil 3.12 : Değiştirilmiş BOOTH algoritması için 3 'er bitlik kodlama

Orijinal Booth algoritması ile bu algoritmayı karşılaştırırsak ayrık 1 ve 0 'ların burada daha etkin kullanıldığını görebiliriz. Eğer x_{i-1} ayrık 1 ise y_{i-1} = 1 ve sadece tek bir işlem gerekir. Benzer sadeleştirme x_{i-1}, 1 dizisi içinde ayrık 0 ise meydana gelir. Bu durumda ...10 (1)..., ...1' 1...'e yeniden kodlanır ve bu da kolaylıkla ...0 1'...'e yeniden kodlanır ve tekrar sadece bir işlem gerçekleştirilir. Gereken işlemleri bulmak için basit bir yol x_{i-1} + x_{i-2} – $2x_i$'nin tek değerleri için hesaplamak ve sonucu 2-bit ikili sayı olarak Y_i Y_{i-1} SD gösterimi ile temsil etmekdir. 01 | 01 | 01 | 01 | (0) için 01 | 01 | 01 | 01 | buluruz. Burada işlem sayısı 4 olarak kalır. Bu da minimum değerdir. Diğer yandan 01 | 01 | 01 | 01 |(0) için 01 | 0 1' | 0 1' |0 1'| elde ederiz ki bu da üç yerine dört işlem gerektirir .

Örnek 3.2: A = 0000 0101 (+5) X = 1001 1111 (-97)

Çarpılan(A)		00 00 01 01	
Çarpan (X)	x	10 01 11 11	
Y		1'0 10 00 01'	(yeniden kodlanmış çarpan)
		-2A 2A 0A -A	
Sonuç		00 00 00 00	
Topla -A	+	11 11 10 11	
2 bit Ötele		11 11 11 10 11	(yeni sonuç)
+0A		00 00 00 00	
Sadece 2 bit Ötele		11 11 11 11 10 11	(yeni sonuç)
+2A		00 00 10 10	
Topla	+	00 00 10 01 10 11	
2 bit Ötele		00 00 00 10 01 10 11	(yeni sonuç)
-2A		11 11 01 10	
Topla	+	11 11 10 00 01 10 11	
2 bit Ötele		11 11 11 10 00 01 10 11	(yeni sonuç)
Nihai Sonuç		11 11 11 10 00 01 10 11 = (- 485)	

Örnek 3.3: A = 0110 (+6) X = 1010 (- 6)

Çarpılan(A)		10 10	
Çarpan (X)	x	10 10	
Y		01' 1'0	(yeniden kodlanmış çarpan)
		-A -2A	
Sonuç		00 00 00	
Topla -2A	+	11 01 00	
2 bit Ötele		11 11 01 00	(yeni sonuç)
- A		11 10 10	
Topla	+	11 01 11 00	
2 bit Ötele		11 11 01 11 00	(yeni sonuç)
Nihai Sonuç		11 11 01 11 00 = (- 36)	

Yukarıdaki Örnek (3.2)'de çarpma işleminde n / 2 = 4 tane Örnek (3.3)'de de n / 2=2 tane adım vardır ve her adımda iki çarpan biti işlem görür.Sonuç olarak bütün öteleme işlemleri, iki bit uzunlukludur. Bu nedenle 2A toplamasını kontrol edebilmek için doğru işareti tutmak amacı ile ek bir bit gereklidir.

Aynı anda üç biti yeniden kodlayacak şekilde uzatmak mümkündür. Bunun sonucunda her biri dört bitlik gruplar elde edilebilir. Böyle bir algoritmaya da 8 tabanına göre Değiştirilmiş Booth algoritması adı verilir.Aşağıdaki tablo Booth algoritmasının 8 tabanında yeniden kodlama şeklini sunmaktadır [11,15].

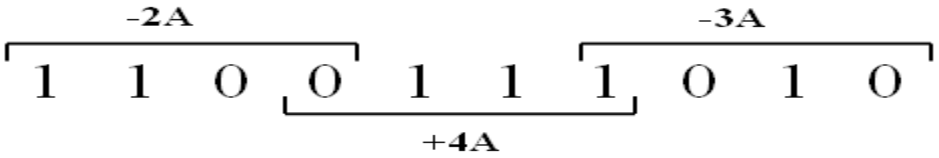
Tablo 3.3 : BOOTH algoritması 8 tabanında yeniden kodlama tablosu

X_i	X_{i-1}	X_{i-2}	X_{i-3}	Zi	Y_i	Y_{i-1}	Y_{i-2}
0	0	0	0	0	0	0	0
0	0	0	1	+1	0	0	1
0	0	1	0	+1	0	0	1
0	0	1	1	+2	0	1	0
0	1	0	0	+2	0	1	0
0	1	0	1	+3	0	1	1
0	1	1	0	+3	0	1	1
0	1	1	1	+4	1	0	0
1	0	0	0	-4	1'	0	0
1	0	0	1	-3	1'	0	1
1	0	1	0	-3	1'	0	1
1	0	1	1	-2	0	1'	0
1	1	0	0	-2	0	1'	0
1	1	0	1	-1	0	0	1'
1	1	1	0	-1	0	0	1'
1	1	1	1	0	0	0	0

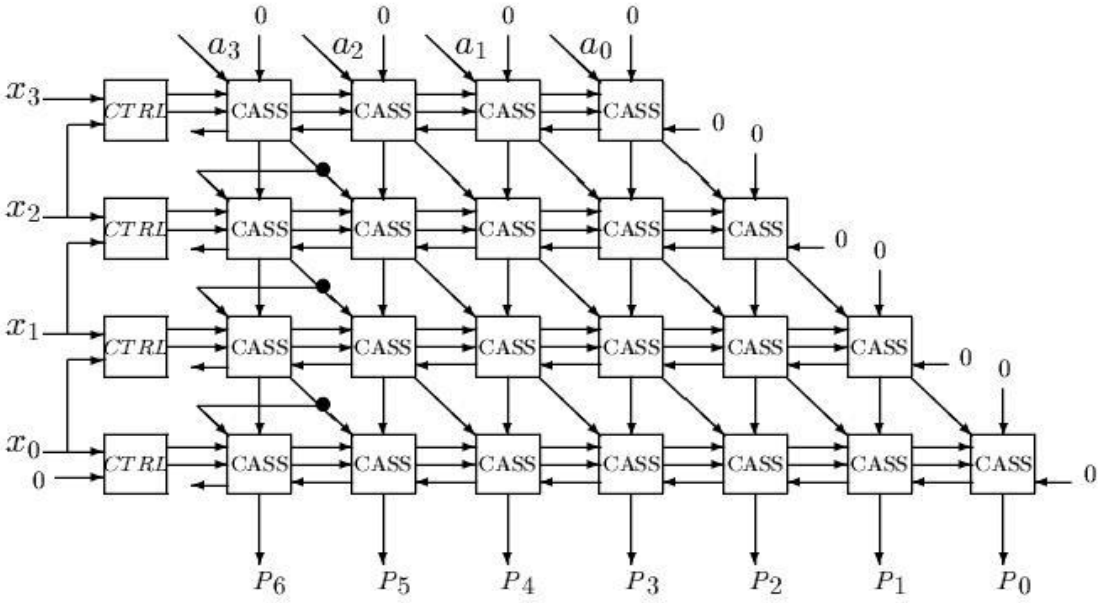
Başlangıçta " 0 " değeri çoğu çarpan sayının sağ tarafina eklenir. Böylece tabloya göre 4'er bitlik kodlama doğru bir şekilde yapılabilmektedir. Bu eşitlik şu şekilde sunulabilir.

$$Z_i = (Y_i \quad Y_{i-1} \quad Y_{i-2}) = X_{i-3} + X_{i-2} + 2X_{i-1} - 4X_i \qquad (3.3)$$

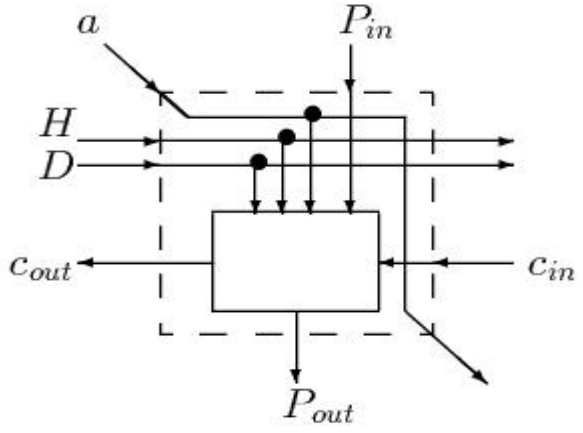
Burada çarpan 110011101 ‘dir. Sağ tarafa “ 0 ” eklenmiş gösterim 1100111010 (eklenen “ 0 ” değeri) 4’er basamak seçilerek elde edilen gösterim aşağıdaki gibidir.



Şekil 3.13 : Değiştirilmiş BOOTH algoritması için 4 ’er bitlik kodlama



Şekil 3.14 : BOOTH algoritmasının mantıksal yapısı



Şekil 3.15 : BOOTH algoritması için CASS hücresinin mantıksal gösterimi

3.3. MATLAB GRAFİKSEL KULLANICI ARAYÜZÜ (GUI)

Matlab programcısı tarafından hazırlanan grafik tabanlı uygulamaların, son kullanıcıya fare ve klavye arabirimi ile enteraktif olarak hitap etmesini sağlayan bir platformdur.Diğer bir anlamıyla içeriğinde yer alan nesnelerin kullanılması ile kullanıcıya etkileşim sağlayan ve bir işin veya bir programın koşturulmasını sağlayan grafiksel bir program arayüzüdür [17]. Açılımı Graphical User Interface (GUI) dir. Matlab GUI uygulamalarının gerekliliğinin temel sebeplerinin başında günümüzde hazırlanan uygulamaların grafik tabanlı oluşu ve bu uygulamaların son kullanıcı tarafından kullanım kolaylığına sahip olması gelmektedir . GUI nesneleri menüler, araç çubukları, radio butonlar, liste kutuları veya kaydırıcılar olabilir. Bunların yanında MATLAB GUI ile MATLAB'in sunduğu hesaplama imkânları kullanılarak da data alımı ve grafik çizimi gibi pek çok işlem gerçekleştirilebilir [18].

3.3.1. MATLAB Grafiksel Kullanıcı Arayüzü (GUI) Oluşturma Yöntemleri ve Çalıştırılması

MATLAB GUI tasarımları iki ayrı yöntem kullanılarak yapılabilir. Bunlar,

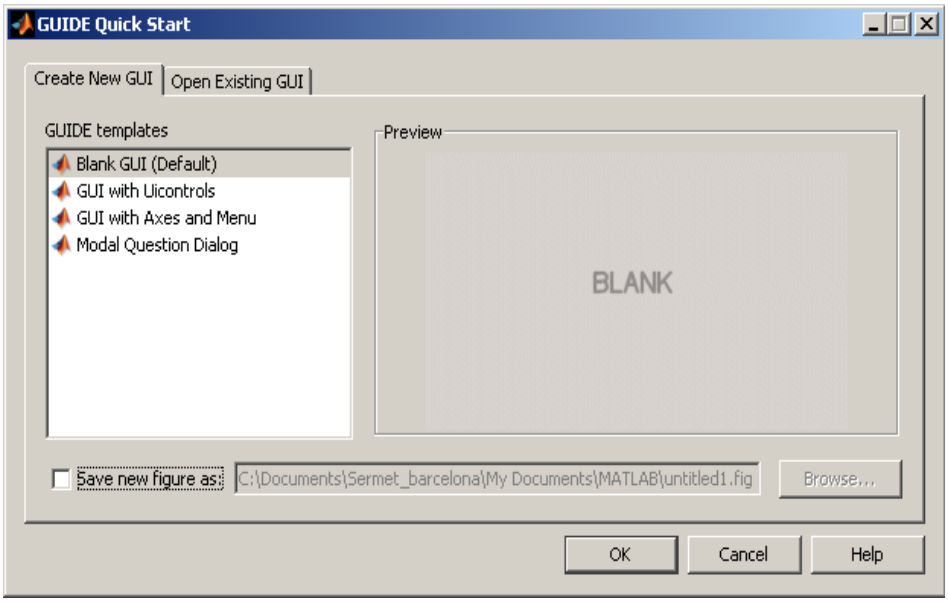
- MATLAB GUIDE aracı kullanılarak.
- M-File programlama yöntemi kullanılarak.

Özellikle GUI tasarımında hızlı arayüzler dizayn etmek ve bu işe ilk başlayan programcılar için MATLAB GUIDE aracının kullanılması büyük bir kolaylık sağlar. Bu aracın kullanılması ile GUI arabirimi kolaylıkla ve yorulmadan sürükle bırak ve açılan pencerelerde özelliklerin değiştirilmesine dayanan bir yöntem kullanılır. Ayrıca, bu yöntemi kullanmanın ileride var olan bir GUI nin düzenlenmesi ve değişiklik yapılması bakımından da çok yararlıdır.

M-File programlama yönteminde tüm GUI tasarımları ve callback program parçalarının yazılması tamami ile programlama kodları kullanılarak yapılır. Burada tasarımcı her şeye hakimdir ve bu teknik uzman bir programlama bilgisi gerektirir. Bu yöntem ile tasarım zamanı uzamasına rağmen programcı her türlü manipülasyonu yapabildiği için programcı açısından çok yararlıdır.

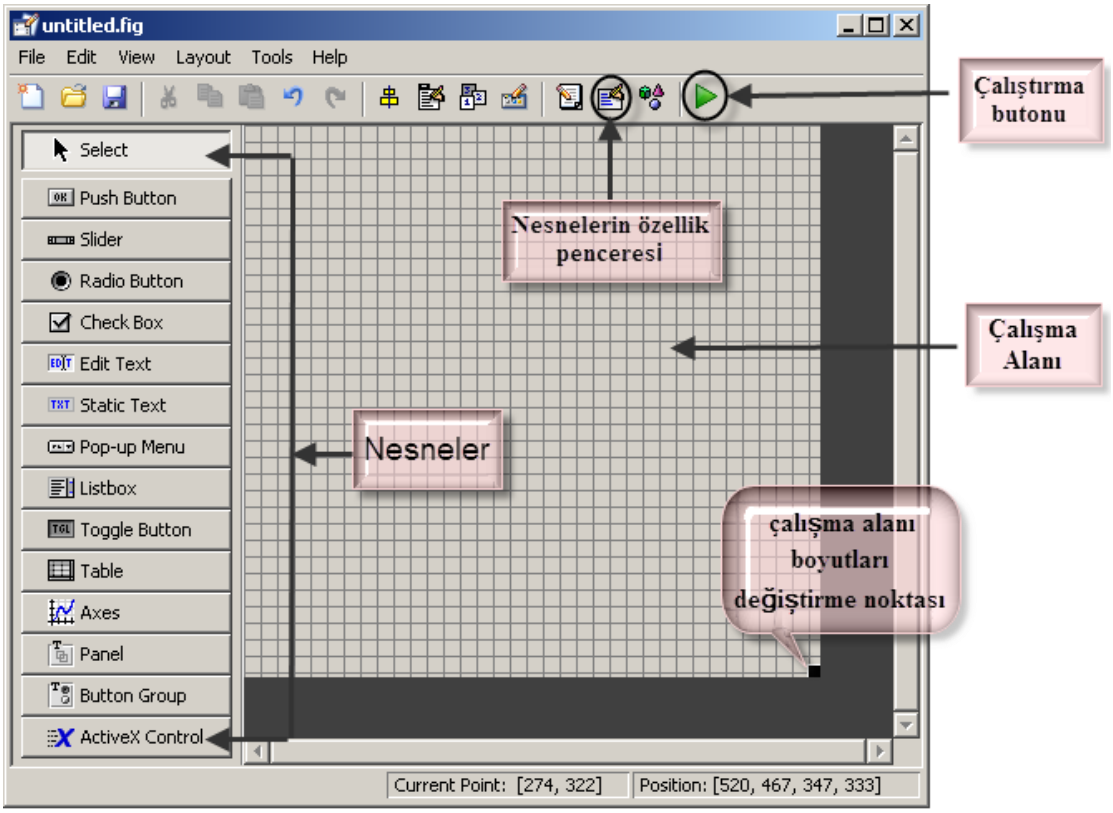
Her bir nesne (veya komponent) GUI için tanımlanan programlama dosyasında callback diye adlandırılan ayrı alt rutin programlama parçalarına sahiptir. Bu şekilde her bir nesnede oluşan olaylara (örnek olarak bir buton nesnesinin tıklanması ile click event oluşması gibi) GUI o olaya ait callback rutinlerini icra ettirir. Yani, GUI hem bir arayüz hem de bir program çağrılarını icra ettirme mekanizması olarak çalışır.

GUI çalıştırmak için ya MATLAB komut satırından GUIDE komutu verilir ya da Start düğmesi tıklanarak MATLAB/GUIDE komutu verilir. Bu adımdan sonra karşımıza Şekil 3.16'daki gibi bir pencere gelir.



Şekil 3.16 : MATLAB GUİDE başlangıç penceresi

Bu pencereden eğer yeni bir GUI tasarımı yapacak isek Blank GUI seçeneğini seçeriz. Şayet önceden yapılmış bir tasarımı açmak istiyor isek Open Existing GUI sekmesinden sonra istenilen dosyayı seçeriz. Burada yeni bir tasrarım oluşturulacağını kabul edelim. Bundan sonra OK düğmesi tıklanılarak Şekil 3.17'deki GUIDE LAYOUT Editor (GUIDE Çalışma Alanı) penceresine ulaşırız.



Şekil 3.17 : MATLAB GUIDE layout editor (GUIDE Çalışma Alanı) penceresi

3.4. VHDL DONANIM TANIMLAMA DİLİ

VHDL, IEEE (IEEE standard 1976) tarafından standart bir tanımlama dili olarak kabul edilmiştir. Dil pek çok yenilemeden geçmiştir. Şu anda 1993 yılı üretimi en çok kullanılan halidir. Elektronik sistemlerin karmaşıklığının artması tasarım yöntemlerinin de gelişmesini gerektirmiştir. Bu sebeple, geleneksel "kağıt ve kalem kullanarak tasarımı yap" ve "devreyi kurarak denemeleri yap" yöntemlerinin yerini "tanımla ve sentezle" yöntemleri almıştır. Donanım Tanımlama Dilleri'nin (Hardware Description Languages : HDLs) "tanımla ve sentezle" yönteminde önemli bir rolü vardır [19]. HDLs bir elektronik sistemin tanımlanmasında, test edilmesinde ve sentezlenmesinde kullanılırlar. Pek çok donanım tanımlama dillerinin arasında VHDL (**V**HSIC (**V**ery **H**igh **S**peed **I**ntegrated **C**ircuit) **H**ardware **D**escription **L**anguage) (yüksek hızlı tümleşik devreler için donanım tanımlama dili) en yaygın kullanılanıdır.

VHDL, önce sadece dökümantasyon ve modelleme dili olarak ele alındı. Sayısal bir sistem davranışını tanımlamak ve simülasyonunu yapmak için geliştirildi. Zamanla

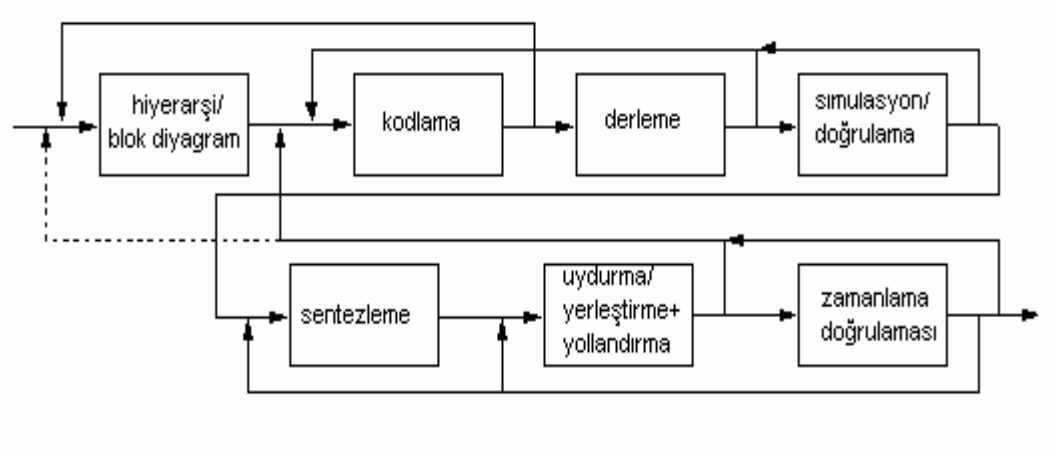
yaygınlaşmaya başlayınca VHDL sentezleme araçları geliştirildi. Bu programlar VHDL davranışsal tanımından yola çıkarak lojik devre yapısını oluşturmaktadır. Bir yonga üzerinde, basit bir kombinezonsal devreden karmaşık bir mikroişlemciye kadar herhangi bir sayısal tasarım VHDL kullanılarak tasarımı, simülasyonu ve sentezlemesi yapılabilir.

VHDL in özellikleri aşağıdaki gibidir:

- Tasarımlar hiyerarşili şekilde bileşenlerine ayrılabilir.
- Her bir tasarım elemanı iyi tanımlı bir arayüze (diğer elemanlarla bağlantı için) ve hatasız davranışsal tanımlamaya sahip olmalıdır.
- Uyumluluk, zamanlama ve saatle denetim modellenebilir. VHDL senkron ve asenkron ardışıl devre yapılarını gerçekleyebilir. İşlemlerin ve zaman davranışının simulasyonu yapılabilir.
- Davranışsal tanımlama yapılırken algoritma veya gerçek donanım yapıları kullanılarak elemanların işlemi belirtilir.

3.4.1. TASARIM AKIŞI

VHDL tabanlı tasarım işlemi birkaç adımda yapılabilir. Bu adımlar HDL tabanlı bütün tasarımlara uygulanabilir. Şekil 3.18'de tasarım adımlarının taslağı görülmektedir.



Şekil 3.18 : VHDL tasarım adımları blok şeması

Bir sayısal sistemin tasarımı, genel olarak aşağıda gösterildiği gibi yedi aşamada yapılır.

- Tasarım aşaması
- Kodlama
- Derleme
- Simülasyon / Doğrulama
- Sentezleme
- Yerleştirme / Yollandırma
- Zamanlama doğrulaması

3.4.1.1. Tasarım aşaması

Büyük sayısal tasarımlar genel olarak hiyerarşik bir yapıya sahiptir. Tasarıma yukarıdan aşağıya yaklaşımda, sayısal sistem daha az karmaşık alt sistemlere ayrılır ve bu alt sistemler gerçekleştirilmeye çalışılır. Yukarıdan aşağıya doğru, tasarm şu adımları gerektirir: Sistem tasarım, yapısal tasarım, saklayıcı iletişim seviyesinde (Register Transfer Level:RTL) tasarım ve lojik seviyede tasarım. Sistem tasarımı seviyesinde, tasarımcı sistemi alt sistemlere ayırır ve alt sistemler arasındaki haberleşmeyi tanımlar. Yapısal tasarımda, yapı özellikleri ve her alt sistemin performansı tanımlanır. Saklayıcı iletişim seviyesindeki tasarımda, yapı birbirine bağlı modüllere ayrılır. Son olarak da, modüller, lojik kapılar kullanılarak tanımlanır.

3.4.1.2. Kodlama

Modüllerin VHDL kodlarının yazılmasıdır. Bu kodlama işleminde modüllerin ara yüzleri ve yapıları detaylı bir şekilde yazılır. VHDL, metin temelli dil olduğundan dolayı, bu işlemi yapmak için metin editörü kullanılabilir.

3.4.1.3. Derleme

Kod yazıldıktan hemen sonra yapılması gereken, kodun doğruluğunu kontrol etmek için derleme işleminin yapılmasıdır. VHDL derleyicisi , yazılan kodların söz dizim kurallarına uygunluğunu ve diğer modüllerle uyumluluğunu analiz eder. Derleyici aynı zamanda simülasyon işleminde kullanılacak dosyaları oluşturur.

3.4.1.4. Simülasyon / Doğrulama

VHDL simülatörü, tasarımdaki girişleri uygulamayı ve çıkışları gözlemlemeyi herhangi bir fiziksel devreye sahip olmadan gerçekleştirir. Küçük projelerde girişleri üretmek ve çıkışları gözlemlemek elle yapılabilir. Fakat, büyük projeler için, kolaylık açısından "test bench" oluşturulup, otomatik olarak girişlerin uygulanması ve çıkışların gözlemlenmesi yapılabilir. Simülasyon adımına aynı zamanda doğrulama da diyebiliriz. Simüle edilen devrenin istenen çıkışı verdiğini görmemiz gerekir. Burada fonksiyonel ve zamanlama olma üzere iki farklı doğrulama var. Fonksiyonel simülasyonda, zamanlamadan bağımsız olarak devrenin sayısal işlemi üzerinde durulur. Kapı gecikmeleri ve diğer zamanlama parametreleri sıfır olarak düşünülür. Zamanlama doğrulamasında , devrenin tahmini yaklaşık gecikmesini ve diğer zamanlama gereksinimleri incelenir. Burada elde edilen zamanlama sonuçlar tahmini değerlerdir. Gerçek sonuçlar sentezleme ve yerleştirme sonuçlarına bağlıdır.

3.4.1.5. Sentezleme

Burada VHDL tanımları, donanımsal olarak hedef teknolojiye uygulanır. Sentezleme işlemi kullanılan araçlara ve hedef teknolojiye göre farklı şekillerde yapılır. Devre gerçekleştirilirken çalışılacak üretici (Xilinx, Altera Atel, vs) seçilir. Lojik sentezlemenin gerçekleştirilmesi için kullanılan teknolojiyi destekleyen bir sentezleyiciye ihtiyaç vardır. Sentezleme araçları, hedef teknolojiye göre, kullanılacak kapıların listesini ve bunlar arasındaki ara bağlantıları belirten bir bağlantı listesi (netlist) üretir. Sentezleme sırasında yapılacak optimizasyonun alan veya hıza göre yapılabilmesi kullanıcı tarafından sağlanabilmektedir. Ayrıca, devrenin çalışma frekansı, bazı bağlantıların gecikme bilgileri, saat ofseti gibi kısıtlamalar sentezleme sırasında verilebilir ve bu kısıtlamalara göre en uygun devrenin sentezlenmesi sağlanabilir. Aşağıda davranışsal sentezleme, RTL sentezleme ve lojik sentezleme anlatılmıştır :-

- Davranışsal sentezleme, C programına benzer, algoritmik olarak yazılmış tanımlamanın RTL tanımlamaya çevrilmesidir. RTL'deki tanımlama veri yolları,bellek elemanları, kontrol birimleri içerir.

- RTL sentezleme, saklayıcı iletişim fonksiyonlarından ardışıl bir devre elde etmek üzere devre şemasının oluşturulmasıdır.
- Lojik sentezleme, Boole fonksiyonlarının kombinezonsal devre elemanlarıyla gerçeklenmesidir.

3.4.1.6. Yerleştirme / Yollandırma

Sentezleme sırasında elde edilen bağlantı listesi yerleştirme adımında yerleştirme araçları tarafından kullanılır. Bu araçlar sentezlenmiş bileşenleri hedef devreye yerleştirir. Bu adımda tasarımcı, Modüllerin yerleşimi, dış giriş ve çıkış pinlerinin atanması gibi ek kısıtlamalar tanımlayabilir.

3.4.1.7. Zamanlama doğrulaması

En son aşama da, yerleştirilen devrenin zamanlama doğrulamasının yapılmasıdır. Sadece bu adımda, kablo uzunluğu, elektriksel yükleme ve diğer faktörlerden kaynaklanan gerçek devre gecikmesi hesaplanabilir. Eğer hesaplanan değerler isteği karşılıyorsa tasarım tamamlanır.Tasarım yapılırken, şekil 3.18'de görüldüğü gibi, adımlar arasında ileri ve geri gitme olabilir.Örneğin, kodlama sırasında herhangi bir problemle karşılaşılırsa, geriye dönülüp hiyerarşi tekrar incelenir. Simülasyon sonucu istenildiği gibi değilse kodlama işlemi gözden geçirilir.

VHDL kullanımını en büyük avantajları şu şekilde sıralanabilir :-

- Tasarım tanımlamaların bağımsız olması.
- Bir çok üreticinin kullanılabilmesi.
- Teknolojinin gerisinde kalan elemanların kullanımının engellenmesi.
- Tasarımın güncellenmesi.
- Belgelemenin standart bir şekilde hazırlanması.
- Tasarım süresini kısaltır.
- Ürünün pazara hızlı çıkmasını sağlar.
- Tasarım maliyetini azaltır.
- Tasarım kalitesi artar.
- Fonksiyonların yüksek seviyede kontrolü yapılabilir.

3.4.2. Yüksek Hızlı Tümleşik Devre Donanım Tanımlama Dili (VHSIC)

VHDL en çok kullanılan yapısal tanımlama dilidir. HDL'ler sayısal sistemin yapısını veya davranışını tanımlamada kullanılır. HDL'de yapılan devre tanımı kullanılarak, bir benzetim programı ile yazılan tanımlamanın istenilen fonksiyonu sağlayıp sağlamadığı kontrol edilir. Tanımlamanın doğruluğu görüldükten sonra, bir sentezleme programı ile devrenin fiziksel yapısı elde edilir.HDL tabanlı tasarım yöntemlerinin geleneksel kapı seviyesinde tasarım yöntemlerine göre bir takım üstünlükleri vardır. İlk olarak, tasarımcının daha az zamanda ve devre tasarımı hakkında çok fazla bilgi sahibi olmadan tasarım yapmasını sağlamaktadırlar. İkinci olarak, aynı tanımlama ile devrenin teknolojilerdeki yapısı elde edilebilmektedir.Devrenin hangi teknoloji kullanılarak gerçekleneceği göz önünde bulundurulmaksızın HDL tanımlaması yapılır. Bir sentezleme programından yararlanarak, HDL ile yapılan tanımlama her defasında farklı teknolojilerdeki elemanlardan oluşan devre şemasına çevrilebilir [16].

VHDL kullanılarak, sayısal tasarımın algoritmik, RTL ve lojik kapı seviyeleri gibi farklı seviyelerde tanımlanması mümkündür. Yukarıdan aşağıya tasarım yönteminde, system yüksek seviyede tanımlanır ve sonra sistem daha alt seviyedeki bloklara ayrılır.

Şekil 3.19'da VHDL ile N bitlik tam toplayıcının farklı tanımlama şekillerine göre yazılmış kodları görülmektedir.Devrede "entity" bölümünde tanımlanan A , B ve EG girişleri ve EÇ ,Toplam çıkışları vardır. Entity'de yeni bir modül ismi ve modülün giriş/çıkışları tanımlanır. Mimari bölümünde, girişler ve çıkışlar arasındaki ilişkler veri akışı (data flow) , davranışsal (behaviour) veya yapısal (structure) olarak tanımlanır.

- Veri akışı şeklindeki mimari ile kontrol ve veri işaretlerinin devre içindeki akışı tanımlanır. Bu işlem yapılırken, toplayıcılar, karşılaştırıcılar, kod çözücüler gibi daha önceden tanımlanmış kombinezonsal devre fonksiyonları ve basit lojik kapılar kullanılır.
- Davranışsal mimari, ile bir sistemden beklenen fonksiyon programa benzer şekilde, tanımlanır. Fakat sistemin nasıl gerçekleneceği konusunda ayrıntılı bilgi verilmez.
- Yapısal mimari ile tasarımın alt sistemlerini oluşturan daha önce tanımlanmış modüllerin birbirleriyle olan bağlantıları tanımlanır.

```
Entity  TOPLAYICI  is

      port (A, B : in STD_LOGIC_VECTOR(N-1  downto 0);
      Toplam  : out STD_LOGIC_VECTOR(N-1  downto 0);
      EG : in STD_LOGIC;
      EÇ : out STD_LOGIC);

End  TOPLAYICI;
```

(a)

```
 Architecture  DATAFLOW  of  TAM_TOPLAYICI  is

      Signal  S : STD_LOGIC;
 Begin
      S <= A xor B;
      TOPLAM <= S xor EG after  10 ns;
      EÇ <= (A and B) or (S and EG) after 5 ns;

 End  DATAFLOW;
```

(b)

```
Architecture  BEHAVIOUR  of  TAM_TOPLAYICI  is
Begin
    process (A, B, EG)
    begin
        if (A='0' and B='0' and EG='0') then
            TOPLAM <= '0';
            EÇ <= '0';
        elsif  (A='0' and B='0' and EG='1') or (A='0' and B='1' and EG='0') or
        (A='1' and B='0' and EG='0') then
            TOPLAM <= '1';
            EÇ <= '0';
        elsif  (A='0' and B='1' and EG='1') or (A='1' and B='0' and EG='1') or
        (A='1' and B='1' and EG='0') then
            TOPLAM <= '0';
            EÇ <= '1';
        else
           TOPLAM <= '1';
           EÇ <= '1';
        endif;
   end process;

End  BEHAVIOUR;
```

(c)

```
Architecture STRUCTURE of TAM_TOPLAYICI is

  Component YARI_TOPLAYICI
      port (G1, G2 : in STD_LOGIC;
      ELDE, TOPLAM : out STD_LOGIC);
  End component;

  Component VEYA_KAPISI
      port (G1, G2 : in STD_LOGIC;
     ÇIKIS : out STD_LOGIC);
  End component;

signal N1, N2, N3 : STD_LOGIC;

Begin
       YT1 : YARI_TOPLAYICI port map (A, B, N1, N2);
       YT2 : YARI_TOPLAYICI port map (N2, EG, N3, TOPLAM);
       VEYA1 : VEYA_KAPISI port map (N1, N3, EÇ);

End STRUCTURE;
```

(d)

Şekil 3.19 : Tam Toplayıcı için (a) Entity (b) Veri Akışı Mimarisi (c) Davranışsal Mimari (d) Yapısal Mimari

3.4.3. VHDL'in Yapısal Elemanları

Sayısal bir sistemin VHDL tanımlaması yapılırken birkaç yapı kullanılır.

Bu yapılar :-

- Entity (Varlık).
- Architecture (Mimari).
- Configuration (Konfigürasyon).
- Package (Paket).
- Process (İşlem).
- Library (Kütüphane).

VHDL deki temel birimler varlıklar, mimariler, konfigürasyon ve paketlerdir. Sayısal bir sistem, modüllerin bir hiyerarşi ile birleştirilmesiyle oluşturulur. Her modül VHDL de bir entity ile ifade edilir. Her entity ile giriş çıkışları ve fonksiyonu tamamen belirlenmiş olan bir donanım yapısı gösterilir. Her tanımlamanın entity bildirimi ve mimari bölümleri olmak üzere iki bölümü vardır. Entity bildirimi tasarımın dış

bağlantılarının, mimari bölümü ise içeride yapılacak işlemlerin gösterilmesinde kullanılır. Pakette pek çok tanımlamada ortak olarak kullanılacak genel bilgiler verilir. Konfigürasyon ile yapısal tanımlamada kullanılacak alt sistemlerin giriş çıkış bilgileri tanımlanır.

3.4.3.1 ENTITY (Varlık)

VHDL'de her blok kendi başına bir devre olarak düşünülür ve entity olarak adlandırılır. **Entity:** Verilen bir lojik fonksiyon için bütün giriş ve çıkışları tanımlar. Yani lojik fonksiyonun dış dünyayla bağlantısını tanımlar. Her VHDL tasarım mutlaka en az bir entity içerir. Yazılış kuralı şu şekildedir:

Entity tanımlayıcı is
(jenerik bildirimi);
Port (işaret tanımlama);
End entity tanımlayıcı;

Port: Port giriş ve çıkış işaretidir. Port ifadesi, her işaret için, port tanımlayıcı, port yönü ve port veri tipini belirlemelidir. Port da 3 yön kullanılır. Giriş için " in ", çıkış için " out ", çift yönlü portlar için " inout " kullanılır. Pek çok değişik veri tipi kullanılabilir. Port() yazılış şekli aşağıda verilmiştir.

Port (
port_ismi : [mod] tipi [:= ilk degeri]
{;port_ismi : tipi [:= ilk degeri]}
);

Veri tipleri:

Bit: 0 ve 1 değerini alabilir.
Bit-vector: aynı isim altında bir dizi 0 ve 1'i göstermek için kullanılır.
İnteger: pozitif ya da negatif tamsayılara karşı düşer
Natural: 0'dan başlayıp istenen bir limit değere kadar tamsayılar için kullanılır.
Positive: 1'den başlayıp istenen bir limit değere kadar tamsayılar için kullanılır.
Boolean: Doğru ve yanlış olmak üzere 2 değerli bir veri tipidir.

Generic bildirimi modülün yapısını veya davranışını kontrol etmek üzere kullanılan sabitlerin tanımlandığı alandır. Yazılış şekli aşağıda verilmiştir.

```
Generic (
        sabit_ismi  : tipi [:= ilk degeri]
    );
```

Giriş/çıkışlar modülün çevre modüllerle haberleşmesini sağlayan yollardır. Her giriş/çıkış bir mod (in, out, inout, buffer) ve bir veri tipi ile gösterilir. Giriş/çıkışların dört modu şu şekildedir:

in: Sadece okunabilir. Sadece giriş için kullanılır.
out: Sadece bir deşer verilebilir, okunamaz. Sadece çıkış olarak kullanılır.
buffer: Okunabilir ve değer yazılabilir. Devrenin içinden sadece bir süreni olabilir.
inout: Okunabilir ve değer yazılabilir. Devrenin içinden birden fazla süreni olabilir.

Aşağıda bir bitlik bir tam toplayıcının bağlantıları görülmektedir. Modülün ismi TAM_TOPLAYICI'dır. Modülün, isimleri A, B ve EG olan üç adet girişi ve isimleri TOPLAM ve EÇ olan iki adet çıkışı vardır. Bütün giriş/çıkışlar STD_LOGIC tipindedirler. Bu özelliklere sahip birimin VHDL tanımlaması şekil 3.20'de gösterilmiştir. Modülün yapısı ve zamanlaması jenerik sabitler kullanılarak kontrol edilebilir. Sentezleme ve simülasyon sırasında jenerik sabitlerin değeri ihtiyaca göre değiştirilebilir.

```
Entity  TAM_TOPLAYICI  is

        port (A, B, EG : in std_logic;
              TOPLAM, EC : out );

End  TAM_TOPLAYICI;
```

Şekil 3.20 : Tam Toplayıcının entity bildirimi

3.4.3.2. MİMARİ BİLDİRİMLERİ (ARCHITECTURE)

Entity ‘de belirtilen kısımların davranışlarının belirlendiği yerdir. BEGIN – END komutları arası paralel (concurent) olarak işlenir. Bir mimari kesinlikle belirli bir varlığa(entity) bağlanmalıdır. Bir varlık, içerisinde birden fazla mimari barındırabilir, örneğin aynı algoritmanın farklı gerçekleştirmeleri veya farklı soyutlama seviyesi. Aynı varlığın farklı mimarileri farklı şekilde isimlendirilmelidir ki ayırt edilebilir olsun. İsim " architecture " anahtar sözcüğünden sonra gelmelidir. Bunu takiben " of " anahtar sözcüğü kullanılır ve varlığın ismi arayüz ('Modül_ismi') olarak kullanılır. Mimarinin başlığı " is " ile sonlandırılır, varlık ifadelerindeki gibi. Bu açıdan " begin " anahtar sözcüğü ifade sonlandırılmadan önce biryerlere koyulmalıdır. Bu varlık ifadelerindeki gibi yapılır: " end " anahtar sözcüğü mimari isminden sonra kullanılır. Mimaride modülün girişleri ile çıkışları arasındaki ilişkiler tanımlanır. Tanımlama davranışsal, veri akışı veya yapısal olmak üzere üç farklı şekilde yapılabilir.Şekil 3.21'de mimari bildirimin yazılış şekli gösterilmektedir.

```
Architecture mimari_ismi of modülün_ismi is
    {mimari_bildirim_bölümü}
Begin
    {eszamanlı_satırlar}
End [mimari_ismi];
```

Sekil 3.21 : Mimari bildiriminin genel yapısı

Mimari bildirim bölümü ile mimari tanımlaması sırasında gereken ön tanımlar, eşzamanlı satırlar ile modülün çalışma şekli verilir. Mimari bildirim bölümünde mimarinin alt parçalarını birbirine bağlanmasında kullanılan işaretlerin tanımlaması yapılır. Her işaret taşıdığı verinin tipini belirten bir veri tipi ile tanımlanır. Her modül farklı mimarilerle tanımlanabilir.

Mimari bölümünde, girişler ve çıkışlar arasındaki ilişkler 3 şekilde tanımlanabilir:-

- Veri akışı mimarisi (data flow).
- Davranışsal mimari (behaviour).
- Yapısal mimari (structure) .

3.4.3.3. Veri Akışı Mimari

Veri akışı mimaride, kontrol işaretleri ve verilerin, toplayıcı, karşılastırıcı, kod çözücü ve basit lojik kapılar gibi kombinezonsal devre elemanları üzerindeki eş zamanlı hareketleri tanımlanır [16].

Aşağıda tam_toplayici için yazılmış veri akışı şeklindeki mimari gösterilmiştir. Bu mimaride üç adet eş zamanlı işlenen işaret ataması vardır. Her satır, duyarlılık listesi sağ taraftaki işaretler olan bir process olarak düşünülebilir. İlk satırda herhangi bir gecikme süresi verilmediğinden, işlem simülatörde bir delta süresi kadar gecikmeli olarak yapılacaktır. İkinci satırdaki tanımlaya göre, toplam işareti, S ve EG'nin değerlerinde değiğiklik varsa, bu değişiklikten 10 ns sonra S xor EG değerini alacaktır. Aynı şekilde, A, B, S veya EG'nin değerinde bir değişiklik varsa bu değişiklikten 5 ns sonra EÇ işareti, (A and B) or (S and EG) değerini alacaktır.

Gecikme parametreleri olarak jenerik sabitler kullanılabilir. Aşağıdaki şekil ile bir gecikme parametresinin tanımı ve mimari de kullanımını gösterilmiştir.

```
Entity  TAM_TOPLAYICI  is

        Generic  (N : TIME := 5 ns);

         Port (A, B, EG : in STD_LOGIC;
               TOPLAM, EÇ : out STD_LOGIC);
         End TAM_TOPLAYICI;

Architecture   DATAFLOW   of TAM_TOPLAYICI   is

   signal  S : STD_LOGIC;

   Begin
          S <= A xor B;
          TOPLAM <= S xor EG after 2*N;
          EÇ <= (A and B) or (S and EG) after N;

End DATAFLOW;
```

Şekil 3.22 : Tam toplayıcının sabit kullanılan veri akışı mimari yapısı

3.4.3.4. Davranışsal Mimari

Davranışsal mimaride sistemin yaptığı işlem process'ler kullanılarak program benzeri bir şekilde yazılır, fakat tasarımın nasıl gerçeklenecegi detaylandırılmaz. Aşağıdaki Şekil 3.23'te tam_toplayici'ya ait davranışsal mimari görülmektedir.

Mimaride, duyarlılık listesinde A, B ve EG olan bir process vardır. Process'in duyarlılık listesindeki işaretlerde herhangi bir değisiklik olmadıysa, process çalıştırılmaz. Ne zaman duyarlılık listesindeki işaretlerden biri değer değiştirirse, o zaman process aktif hale gelir ve içindeki satırlar ardışıl olarak işletilir [16].

```
Architecture BEHAVIOUR of TAM_TOPLAYICI is
Begin
    process (A, B, EG)
    begin
        if (A='0' and B='0' and EG='0') then
            TOPLAM <= '0';
            EÇ <= '0';
        elsif (A='0' and B='0' and EG='1') or (A='0' and B='1' and EG='0') or
        (A='1' and B='0' and EG='0') then
            TOPLAM <= '1';
            EÇ <= '0';
        elsif (A='0' and B='1' and EG='1') or (A='1' and B='0' and EG='1') or
        (A='1' and B='1' and EG='0') then
            TOPLAM <= '0';
            EÇ <= '1';
        else
           TOPLAM <= '1';
           EÇ <= '1';
        endif;
   end process;

End BEHAVIOUR;
```

Şekil 3.23 : Tam Toplayıcının davranışsal mimari yapısı

3.4.3.5. Yapısal Mimari

Yapısal mimaride eş zamanlı çalışan alt modüllerin listesi ve birbirlerine bağlantıları tanımlanır. Aşağıda Şekil 3.24'te bir bitlik bir tam toplayıcının yapısal mimarisi görülmektedir. Yapı isimleri yari_toplayici ve veya_kapisi olan iki alt modüle ayrılmıştır. Tam toplayıcının tasarımında iki yarı toplayıcı ve bir VEYA kapısı

kullanılmıştır. Tasarımın karmaşıklığı arttıkça bütün tasarımı bir defada tanımlamak oldukça zorlaşır. Bu sebeple, sistem daha az karmaşık alt sistemlere ayrılır. Her alt sistem tek başına gerçekleştirilebilen veya alt sistemlere ayrılabilen birimlerdir. VHDL'de yapısal mimari kullanılan yüksek seviyedeki tasarım, sadece daha once tanımlanmış alt sistemlerin listesi ve onların birbiriyle olan bağlantılarından oluşur [16].

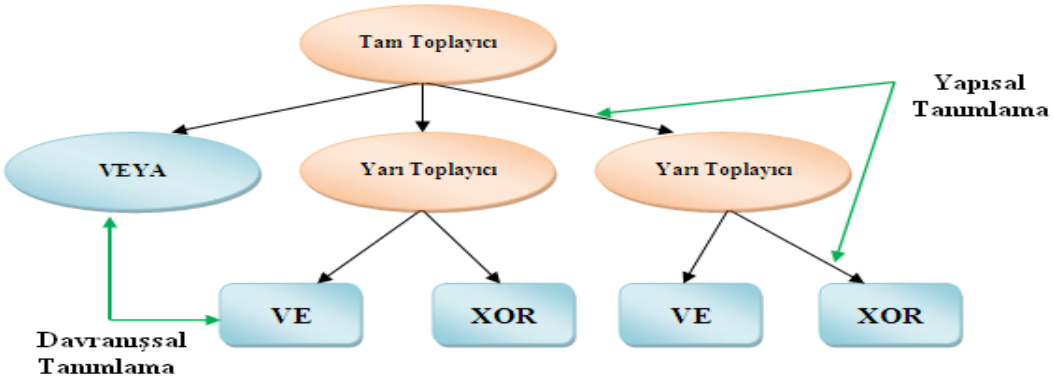
```
 Architecture  STRUCTURE  of  TAM_TOPLAYICI  is

Component  YARI_TOPLAYICI
        port (G1, G2 : in STD_LOGIC;
        ELDE,  TOPLAM : out STD_LOGIC);
   End component;
      Component  VEYA_KAPISI
        port (G1, G2 : in STD_LOGIC;
       ÇIKIS : out STD_LOGIC);
      End component;
 signal  N1, N2, N3 : STD_LOGIC;
 Begin
         YT1 : YARI_TOPLAYICI  port map (A, B, N1, N2);
         YT2 : YARI_TOPLAYICI  port map (N2, EG, N3, TOPLAM);
         VEYA1 : VEYA_KAPISI  port map (N1, N3, EÇ);

End  STRUCTURE;
```

Şekil 3.24 : Tam Toplayıcının yapısal mimarisi

Şekil 3.25'te tam toplayıcının alt sistemlere ayrılması gösterilmiştir. Yarı toplayıcılar da alt sistemlere ayrılarak VE ve EXOR kapılarının uygun şeklide bağlanmasıyla elde edilmiştir.



Şekil 3.25 : Tam Toplayıcının alt sistemlere ayrılışı

Her birimin daha önce tanımlanması ve çalışılan kütüphanenin içinde saklanması gerekir. Örneğin, YARI TOPLAYICI ve EXOR kapısının aşağıdaki şekilde görülmektedir.

```
Entity YARI_TOPLAYICI is

Port (G1, G2 : in STD_LOGIC;
      ELDE, TOPLAM : out STD_LOGIC);
End YARI_TOPLAYICI;

Architecture STRUCTURE of YARI_TOPLAYICI is

Component EXOR_KAPISI
      Port (G1, G2 : in STD_LOGIC;
            ÇIKIS : out STD_LOGIC);
End component;
Component VE_KAPISI
      Port (G1, G2 : in STD_LOGIC;
            ÇIKIS : out STD_LOGIC);
End component;

Begin
B1 : EXOR_KAPISI port map (A, B, TOPLAM);
B2 : VE_KAPISI port map (A, B, ELDE);

End STRUCTURE;
```

(a)

```
Entity EXOR_KAPISI is

Port (G1, G2 : in STD_LOGIC;
      ÇIKIS : out STD_LOGIC);
End EXOR_KAPISI;

Architecture BEHAVIOUR of EXOR_KAPISI is
Begin
    process(G1, G2)
    Begin
        if (G1='0' and G2='0') or(G1='1' and G2='1') then
            ÇIKIS <= '0';
        else
            ÇIKIS <= '1';
        endif;

     End process;

End BEHAVIOUR;
```

(b)

Şekil 3.26 : (a) Yarı Toplayıcı (b) Exor kapısı birimlerinin VHDL tasarımalrı

3.4.3.6. Konfigürasyon (Configuration)

Varlıkların tanımlanması ve örneklenmesi gerçekte kullanılan VHDL modellerinden bağımsızdır. VHDL konfigürasyon içinde bileşenlerin ,tam bir tasarım yapmak için, varlık/mimari çiftine bağlantı oluşturulur. Özet olarak,bir bileşen tanımlanması kesin bir soket tipinin bileşen örneklenmesi ile belirlenmesi demektir. Bir aygıtın örneklenmiş sokete gerçekten eklenmesi konfigürasyon ile yapılmaktadır. Varlıkla mimari arasındaki ilişki konfigürasyonda kullanılan simülasyonla desteklenir, son tasarım hiyerarşisini oluşturur. Bu en üst seviye varlık mimarisinin seçimini içerir. Konfigürasyon VHDL 'deki sentezlenebilen ve simüle edilebilen tek objedir. Konfigürasyon sürecinin simülasyon amaçlı elle kontrolü mümkün olduğunda, sentez araçları herzaman varsayılan kural kümesini uygular. Bunu başarılı bir şekilde gerçekleştirmek için bileşen isimleri ile varolan varlık isimleri birbiriyle uyuşmalıdır. Buna ek olarak, port isimleri, modlar ve veri tipleri birbiriyle uyumlu olmalıdır.Bileşen tanımlamadaki portların sırası önemli değildir.

```
entity A is
  port(A, B:          in  bit;
       SUM, CARRY: out bit);
end A;

architecture RTL of A is
...
```

```
entity B is
  port(U,V: in  bit;
       X,Y: out bit);
end B;

architecture GATE of B is
...
```

```
entity FULLADDER is
  port(A, B, CARRY_IN: in  bit;
       SUM, CARRY:     out bit);
end FULLADDER;
architecture STRUCT of FULLADDER is

  component HALFADDER
    port(A, B:          in  bit;
         SUM, CARRY: out bit);
  ...

  signal W_SUM, W_CARRY1, W_CARRY2: bit;

begin
  MODULE1: HALFADDER
    port map (A, B, W_SUM, W_CARRY1);

  MODULE2: HALFADDER
    port map(W_SUM, CARRY_IN, SUM, W_CARRY2);
  ...

end STRUCT;
```

Şekil 3.27 : Tam Toplayıcının VHDL tasarımı

Şekilden ve koddan anlaşıldığı üzere bir Tam Toplayıcının kodu görülmektedir. Elemanların portları mimarinin sinyalleriyle, pozisyonları yardımıyla bağlantı kurulmuştur, Örneğin ilk sinyal ilk portla bağlantılıdır. HAL FADDER(yarı toplayıcı) varlığı kullanılamaz ancak bunun iki varlığı A ve B 'nin RTL ve GATE diye farklı mimarileri vardır ve bunlar kullanılabilinir.Her iki varlık da eleman tanımlamasıyla uyuşmaktadır.

3.4.3.7. Paket (Package)

Bir paket, veri tiplerinin, altprogramların, sabitlerin, vb tanımlamalardan oluşur. Bu takım çalışması için kullanışlıdır böylece herkes aynı veri tipleri,vb çalışacaktır. Bu farklı tasarımcıların modüllerinin bir araya getirilip VHDL modelin oluşturulmasını kolaylaştırmaktadır. Bir paketi başlık (header) ve gövde (body) olarak bölümlere ayırmak mümkündür. Paketin başlığı, fonksiyon veya prosedürün prototip belirtimi, gerekli tüm veri tiplerinin,vb tanımlanamsını içerir. Altprogramın gerçek implementasyonu (gerçekleştirimi) gövdede yer almaktadır. Bu derleme sürecini kolaylaştırmaktadır, çünkü kısa paket başlıkları kullanılan VHDL kodunun belirtim / tanımlamalarına (declarations/definitions) uyumlu olmalıdır . Bir pakete bir kullanım ifadesiyle referans verilir. Anahtar sözcük " use " 'u seçilen isim "selected name" takip eder. Bu isim paketin yer aldığı kütüphanenin ismini içerir, paket isminin kendisi ve obje ismi referans gösterilmiş olunur. Genellikle " all " anahtar sözcüğü pakette görünen tüm objelere referans göstermek için kullanılır.Paket tanımlama yazılış şekli aşağıda gösterilmektedir.

Package paket_ismi is

Declarations (Types, Constants, Components, Attributes, Functions and Procedures)

End package paket_ismi ;

3.4.3.8. PROCESS (İşlem)

Bir işlem, mimarideki diğer ifadeler gibi davranmaktadır ve geleneksel programlama dillerindeki gibi biri diğerinden sonra gelen ifadeler içerirler. Aslında işlem ifadelerini VHDL ifadelerindeki koşutzamanlı tek ifade olarak kullanmak mümkündür. Bir işlemin gerçekleştirilmesi olayların tetiklenmesiyle ilgilidir. Olası olay kaynakları hassasiyet listesinde veya kesin bekleme ifadelerinde listelenmiştir ve gerçekleştirmenin akışını

kontrol etmek için kullanılır. Bu iki seçenek birbiriyle karşılıklı ilişkilidir birinden biri olabilir yalnızca,hassaslık listesinde bekleme ifadesi olamaz. Hassasiyet listesi sentez araçları tarafinda yok sayılır, bir VHDL simülatörü listelenmiş sinyallerden biri değiştiğinde işlem kodunu çağırır. Tüm sinyaller kombinasyonel süreç içerisinde okunmuştur, bu davranışı etkiler,hassasiyet listesinde bahsedilmiş olmalıdır ki sentezlenmiş donanım simülasyonla aynı sonucu üretsin. Elbette aynısı zamanlı işlemler içinde geçerlidir, yeni girilmiş değerler herbir aktif saat darbesinde hesaplanırlar. Bu nedenle hassasiyet listesi saat sinyallerini ve asenkron kontrol sinyallerini de içerir(reset-sıfırlama). Bir işlem ifadesi isteğe bağlı bir etiketle başlar ve ':' sembolünden sonra " process " anahtar sözcüğü takip eder. Hassasiyet listesi isteğe bağlıdır ve " (' ') " çifti arasında tanımlanır. Mimari ifadesine benzer olarak , tanıtılan bölüm başlık kodu ile " begin " anahtar sözcüğü arasında bulunmaktadır. Sıralı ifadeler " begin " ve " end process " arasında bulunmaktadır. Anahtar sözcük " process " tekrarlanmalıdır. Eğer bir etiket bir işlem için seçildiyse son ifadede tekrarlanmalıdır.

Process'in özellikleri aşağıdaki gibidir:

- Sıralı olarak yürütülen ifadeler içerir.
- Sadece bir mimarinin içinde bulunur.
- Birçok iş aynı zamanda yürütülebilir.
- Yürütüm (Execution) şunlarla kontrol edilir:
    - Hassaslık listesi(sensitivity list) (tetikleyici sinyallerini içerir).
    - Bekleme ifadeleri.
- İşlem etiketi isteğe bağlıdır.

Process'in yazılış şekli aşağıda gösterilmektedir :-

Optional_label (isteğe göre etiket) : **Process** (sensitivity list [Duyarlılık Listesi]) is

Process Declarations (İfadeler);

Begin

Sequential Statements (İşlemler);

End process optional_label;

3.4.3.9. KÜTÜPHANE (Library)

Bir kütüphanede tüm incelenmiş objeler yani paketler, paket gövdeleri, varlıklar, mimariler ve konfigürasyonları bulabilirsiniz. VHDL 'de kütüphane, derlenmiş objelerin grup halinde bulunduğu ve referans verilebildiği mantıksal isimdir. Varsayılan kütüphaneye "work"("çalışma") denir. Bu mantıksal isim diğer kullanılmakta mantıksal kütüphane isimlerine eşlenmiş olabilir, fakat mutlaka bir depolama aygıtına fiziksel yolla eşlenmiş olmalıdır.

Bir kütüphane tek başına analiz edilebilir. Ancak belirli bir sırayla analiz edilmelidir. Örneğin, bir "entity", gerçeklenmesinin tanımlandığı mimariden önce analiz edilmelidir. Bir paket de onu kullanan tasarımdan daha önce analiz edilmelidir. Kütüphane'nin yazılışı aşağıda gösterilmektedir.Aşağıdaki ifadeler kodların en üstüne yazılmalıdır.

Library kütüphane adı ;
Use kütüphane adı.package adı.**all**;

IEEE KÜTÜPHANESİ

```
Library IEEE;
```

IEEE PAKETLERİ

```
use IEEE.math_complex.all;
use IEEE.math_real.all;
use IEEE.numeric_bit.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_misc.all;
use IEEE.std_logic_signed.all;
use IEEE.std_logic_textio.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;
```

3.5. FPGA DONANIMINA GENEL BAKIŞ

Günümüzde devre tasarımında boyutların küçülmesi yanında hız da büyük önem kazanmıştır. Uygulama alanlarının çeşitliliği artması nedeniyle artık uygulamaya özgü devreler ASIC' ler (application specific integrated circuits) geliştirilmiştir. Fakat uygulamanın mimarisinde en ufak değişmelerde bile entegreyi yeniden tasarlayıp üretmek gerekli olmuştur. Devre tasarımı kadar layout (serim düzeni) tasarımı da artık insan emeğini aşmış durumdadır. Aynı zamanda birkaç bin adet üretim/satış bile yonga(chip) tasarımı ve üretim hazırlığı maliyetlerini karşılamayabilir. Çoğu durumda daha önceden seri olarak üretilmiş programlanabilir yongalar kullanmak ASIC üretmekten ve ürettirmekten çok daha uygun olacaktır.

Bağlantı hatları (interconnect) vasıtasıyla, kapıları(AND-OR) ve flip-flop'ları tekrar konfigüre edilebilen devrelere programlanabilir yonga (chip) denir. Başlıca mimarileri aşağıda bulunmaktadır [20,21]:-

- Simple Programmable Logic Devices (SPLD)
- Complex Programmable Logic Devices (CPLD)
- Application-Specific Integrated Circuit (ASIC)
- Field Programmable Gate Arrays (FPGA)

3.5.1. Basit Programlanabilir Mantık Dizisi (SPLD)

Genelde PAL(Programmable Array Logic, Vantis), GAL(Generic Array Logic, Lattice), PLA(Programmable Logic Array), PLD(Programmable Logic Device) olarak da bilinir. Programlanabilir yongaların en ucuz ve en basit olanıdır. Örnek olarak PLA ve PAL aşağıda verilmiştir [21].

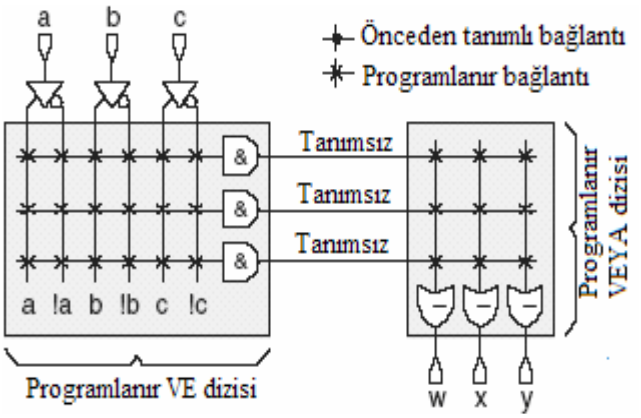
3.5.1.1. Programlanabilir Mantık Dizisi (PLA)

Programlanabilir mantık dizisi teknolojisi kullanıcıya en fazla yapılandırma imkanı verendir.Çünkü VE ve VEYA dizilerinin her ikisi de programlanabilir.

VE dizisindeki VE fonksiyon sayısı, giriş sayısından bağımsızdır. Ek VE kapıları dizideki satırlara kolayca dahil edilebilir [21].

VEYA dizisindeki VEYA fonksiyon sayısı da giriş ve VE fonksiyon sayılarından bağımsızdır. VEYA kapıları dizideki sütunlara kolayca dahil edilebilir.

PLA' lar özellikle mantık eşitlikleri çok sayıda ortak terim içeren çıkışlara sahip büyük tasarımlarda tercih edilebiliyor. Önceden tanımlı karşıtlarına nazaran programlanır bağlantılardan geçişler daha uzun sürer .



Şekil 3.28 : 3-giriş, 3-çıkışlı programlanmamış PLA

PLA 'ların bazı özellikleri:-

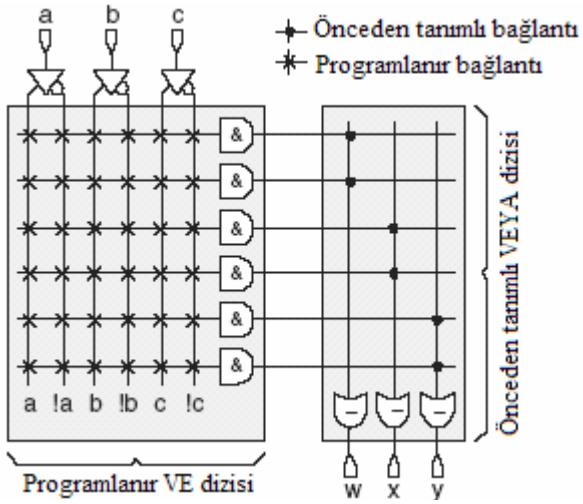
- İki adet programlanabilme düzlemi mevcuttur.
- Herhangi bir AND/OR kombinasyonu gerçekleştirilebilir.
- PAL'dan daha hızlıdır.

3.5.1.2. Programlanabilir Dizi Mantıkları (PAL)

Programlanabilir dizi mantıkları PLA' ların tam tersi bir yapıya sahiptir. VE dizileri programlanır ve VEYA dizileri önceden tanımlıdır. Aşağıdaki şekilde 3 giriş 3 çıkışlı programlanmamış durumda basit PAL verilmiştir [21].

PAL 'ların bazı özellikleri:-

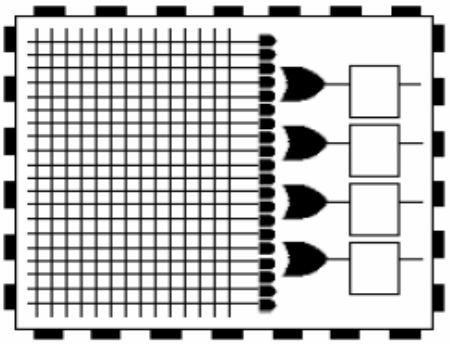
- Bir adet programlama düzlemi mevcuttur.
- Sınırlı sayıda AND/OR kombinasyonu gerçekleştirilebilir.
- PLA'dan yavaştır.



Şekil 3.29 : 3-giriş, 3-çıkışlı programlanmamış PAL

3.5.2. Karmaşık Programlanabilir Mantık Dizisi (CPLD)

Genelde SPLD ile aynı yapıda olup, daha yüksek kapasiteye sahiptirler. Bünyesinde 2 ile 64 adet SPLD barındırmaktadırlar. Diğer bir deyişle PLD'lerden oluşan bir programlanabilir yongadır diyebiliriz.Genelde EPLD(Erasable Programmable Logic Devices), PEEL, EEPLD(Electrically Erasable Programmable Logic Devices), MAX(Multiple Array matriX, Altera) olarak da bilinirler [20].



Şekil 3.30 : Genel CPLD yapısı

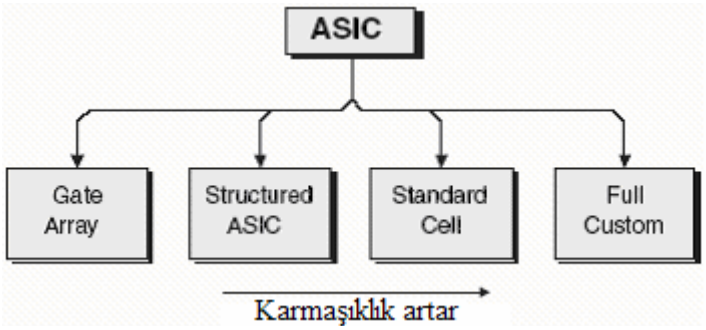
Yukarıda blok yapısı gösterilen çeşidin genel özellikleri şöyle sayılabilir:

- Basit tasarım yapılabilir
- Kolay yönlendirilebilir.
- Hızlı, karmaşık kapılar kullanılabilir.
- 50-200 arasında kapı kullanır.
- Üretim ve geliştirme maliyeti düşüktür.
- Küçük bir alana sığabilir.

3.5.3. Uygulamaya Özel Tümleşik Devre (ASIC)

ASIC devreler yüzlerce milyon kapı içerebilen ve oldukça karmaşık işlevleri yerine getirebilen tümdevrelerdir. Büyük hacimli yongalar gerçeklemeye imkan veren fakat uzun tasarım süreci gerektiren yöntemlerdir [20].

Uygulamaya özgü tümdevre kendi içerisinde dört alt başlıkta incelenir.



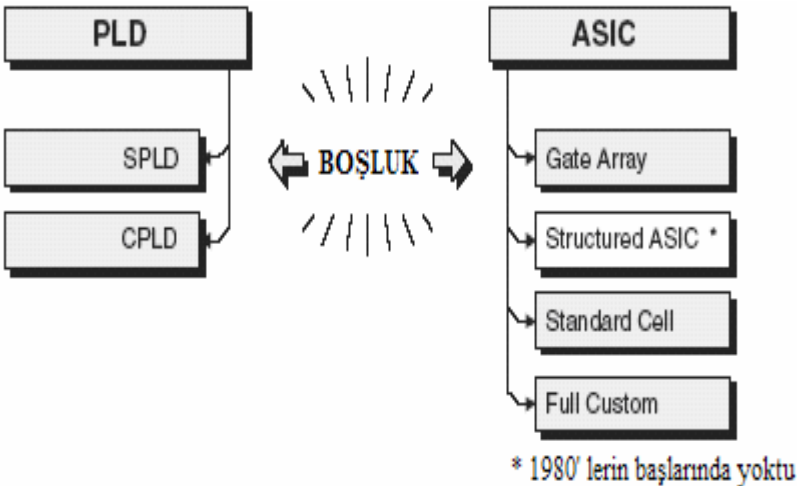
Şekil 3.31 : ASIC tasarım yöntemleri

3.5.4. Field Programmable Gate Arrays (FPGA)

"Sahada Programlanabilir Kapı Dizileri", 1984 yılında Xilinx şirketi kurucularından Ross Freeman tarafından icat edilmiştir.Çok genel bir tanımlamayla FPGA, dijital tasarımlarda kullandığımız farklı lojik kapıların milyonlarcasının tek bir entegre üzerinde toplanmasıdır. Bu lojik kapılar istenildiği gibi programlanabilir ve istenilen her türlü dijital tasarım bu elemanlar üzerinde gerçeklenebilir [20].

Sayısal tümdevre sürecinde 80' li yıllarda belli boşluklar görülmeye başlandı. Bir tarafta SPLD ve CPLD gibi programlanabilir yongalar vardı. Bunlar yüksek yapılandırılabilme, hızlı tasarım ve değişiklik sürelerine sahiptiler ama geniş ve karmaşık tasarımları destekleyemiyorlardı.

Diğer tarafta ASIC tasarım bulunmaktaydı. Çok geniş ve karmaşık işlevleri desteklemelerine rağmen, oldukça pahalı ve zaman harcayan sürece sahiptirler. Üstelik tasarımın geri dönüşü yoktur.



Şekil 3.32 : PLD ve ASIC yaklaşımları arasındaki boşluk

Bu aralığı doldurmak amacıyla Xilinx firması FPGA adını verdiği yeni bir IC sınıfı geliştirdi, FPGA 'lar bu anlamda PLD ler ve ASIC ler arsında bir yerde düşünülebilir. İşlevleri PLD ler gibi özelleştirilebilirken, içerebildikleri milyonlarca kapı kapasitesi ile daha önceleri sadece ASIC tasarımlarla gerçeklenebilen karmaşık yapıları mümkün kılarlar. Öte yandan ASIC ile karşılaştırıldığında tasarım süreci çok daha "kolay, hızlı ve ucuz" dur. ASIC tasarımlar sadece büyük şirketlerin göze alabilecekleri derece pahalıdır. FPGA'lar sayesinde birçok kücük girişimci şirket ayakta kalabilmiştir.

FPGA'lar, SPLD ve CPLD'lerden farklıdırlar ve daha yüksek lojik kapasite sunarlar. FPGA programlanabilir I/O blokları tarafından kuşatılan ve programlanabilir alt bölümler ile bağlantılı lojik blok dizlerinden oluşurlar.

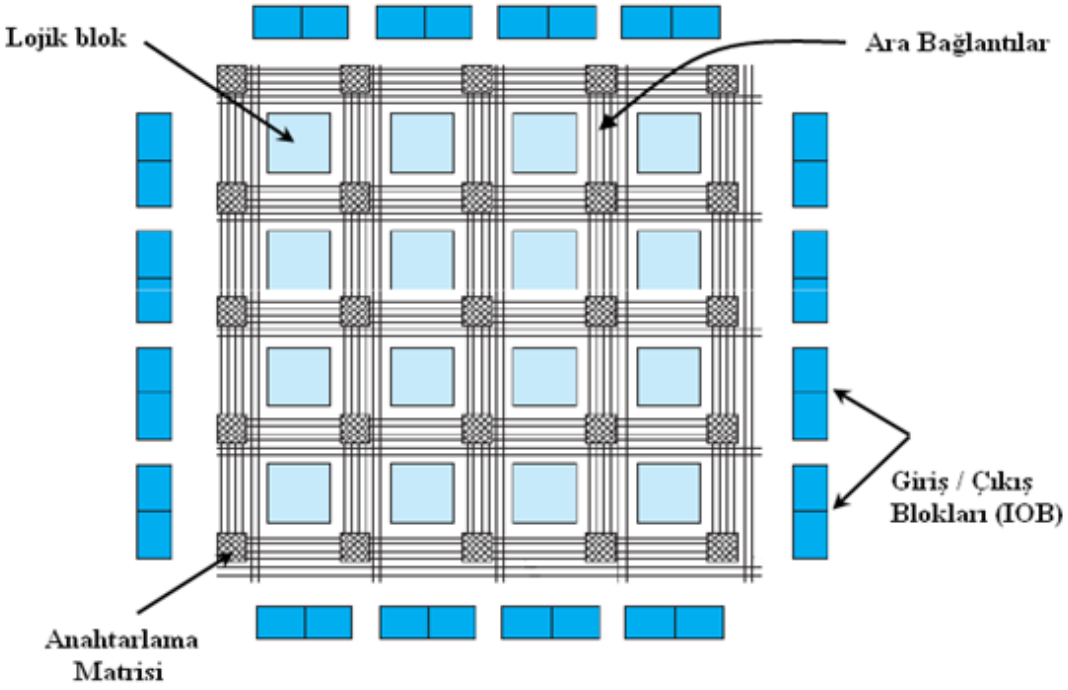
ASIC'lerin yüksek maliyeti, tasarım ve test aşamalarındaki gecikmeleri FPGA'ları çekici kılmaktadır. FPGA'ların diğer avantajları şu şekilde sıralanabilir.

- Mikroişlemciler gibi adım adım işlemler yerine paralel (aynı zamanlı) işlemler gerçekleştirebilirler.
- Bu yüzden klasik mikroişlemcilerden kat kat daha hızlıdırlar.
- Daha esnek yapıya sahiptirler.
- Tekrar programlanabilirler, ASIC'lerden daha az riskli ve daha az işlem gerektirmektedirler.

3.5.4.1. FPGA MİMARİSİ

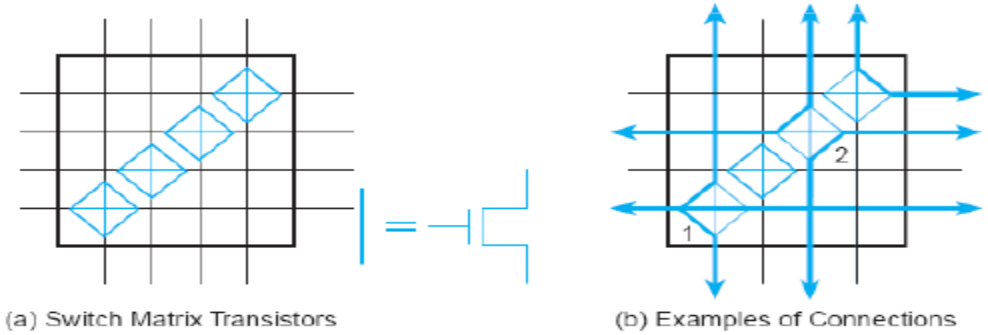
Genel anlamıyla FPGA' lar dört ana birime gereksinim duyarlar:

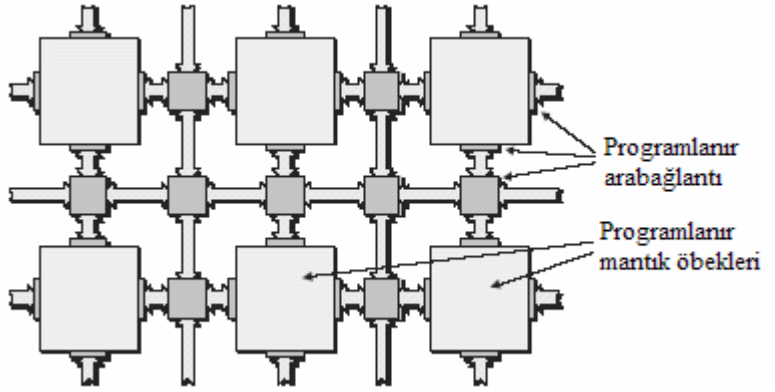
- Lojik bloklar
- Ara bağlantılar
- Giriş/Çıkış blokları
- Anahtarlama matrisi



Şekil 3.33 : FPGA yongasının genel görünümü

(Switch Matrix),Anahtarlama matrisinin 4 kesişim noktasının her birinde 6'şar transistör (pass transistor) bulunmaktadır. Bu yapıyla istenilen bir hat, diğer bir hatta bağlanabilir.





Şekil 3.34 : Genel FPGA mimarisi üstten görünümü

Bir FPGA, birkaç bin ile birkaç milyon arasında kapı içerebilir. (Xilinx Virtex-II modeliyle 10 milyon kapı sayısını aşmış durumdadır). Bu kapılar logic cell'ler, flip-flop'lar ve multiplexer'lar olarak gruplandırılmışlardır ve bloklar arasında bir bağlantı yoktur. Ancak, yonga çalışmaya başladığı anda konfigürasyon amaçlı kullanılan birkaç pini ile dışarıdan tasarım bilgisini alır.

Tasarım yazılımı ile üretilen konfigürasyon dosyası, bir ROM'a yüklenir. FPGA bu tasarım dosyasındaki tasarımı yükler ve kendisine verilen işlevi gerçekleştirmeye başlar. Besleme kesildiğinde FPGA tekrar fabrika çıkışı haline döner. Konfigürasyon bilgisi doğrudan FPGA'e bağlı bir ROM'dan alınabileceği bazı uygulamalarda, aynı devre üzerindeki ROM kullanan başka işlemciden(DSP, mikro denetleyici...) alabilir. Hatta bazı uygulamalarda , FPGA bağlı bulunduğu PC'den her açılışta konfigürasyonu alır .

3.5.4.2. FPGA Kullanım Alanları

Kullanıma ilk başlandığı 1980' lerin ortalarında FPGA yongalardan çoğunlukla ara yapıştırıcı mantık ve kısıtlı veri işleme görevlerinde faydalanıldı. Ara yapıştırıcı mantıklar daha geniş mantık blokları, fonksiyonlar veya cihazlar arasında ara bağlantı olarak kullanıldı.

1990' ların ilk yıllarında artan kapasiteleri sayesinde geniş veri işlemleri gerektiren haberleşme ve ağ ortamlarında kullanımı arttı. 90' ların sonlarına doğru tüketiciye yönelik, otomotiv ve endüstriyel uygulamalardaki kullanımları devasa büyüme sergiledi.

FPGA' lar genellikle ASIC tasarımların ilk örnekleri veya yeni bir algoritmanın fiziksel gerçeklenebilirliğini doğrulamak adına donanım ortamı sağlamak için kullanılırlar.Bununla birlikte, düşük geliştirme maliyetleri ve kısa sürede pazara sunulabilme özellikleriyle gittikçe son ürün yelpazesindeki yerlerini almaktadırlar.

2000' lerin ilk yıllarında milyonlarca kapı içeren yüksek performanslı FPGA' lar piyasada yerini aldı. Bazıları gömülü mikroişlemci çekirdekleri, yüksek hızlı Giriş/Çıkış (I/O) arayüzleri, gömülü RAM ve DSP öbekleri sunmaktadır. Sonuç olarak günümüz FPGA' ları dört ana pazar kolunda yer bulmaktadırlar.

- ASIC ve özel yapım tümdevre
- DSP(Digital Signal Processing)
- Gömulü Mikrodenetleyici uygulamaları (Embedded Microcontroller)
- Fiziksel katmanda yer alan haberleşme tümdevreleri

3.5.4.3. FPGA Üretici Firmalar

Actel	;	www.actel.com
Altera	;	www.altera.com
Anadigm	;	www.anadigm.com
Atmel	;	www.atmel.com
Lattice Semiconductor	;	www.latticesemi.com
Leopard Logic	;	www.leopardlogic.com
QuickLogic	;	www.quicklogic.com
Xilinx	;	www.xilinx.com

Bu firmalar arasında Xilinx ve Altera en büyük hacimli FPGA sağlayıcılarıdır. Her firmanın kendilerine has teknik üstünlükleri vardır.

Bizim geliştireceğimiz projede ise Altera ve Xilinx firmasının Cyclone , Spartan serilerinin Cyclone II , Spartan 3E modelileri üzerinden çalışmalarımızı sürdüreceğiz.

Tablo 3.4 Kullanılan FPGA kitlerinin bazı özellikleri

Device Family	Cyclone II	Spartan 3E
Target Device	EP2C70	XC3S100E
Package	F896	VQ100
Speed Grade	6	-5
I/O Standards	622	540
No. of GCLK's	16	24
No. of Slices	810	960
No. of flip flops	1580	1920

4. BULGULAR

Bu tezde, geliştireceğimiz Vedic matematiğine dayanan 4 bitlik Önerilen (Proposed) çarpma devresi üzerinden çalışmalarımızı sürdüreceğiz. 4 bitllik Önerilen çarpma algoritmasında 8 ve 16 bit tanımlanan koşullarda yapılan bazı değişikliklerle daha büyük bit uzunluklu devreler geliştirilmiştir.Geliştirilen çarpma algoritmalrının örnekler üzerinde temel prensipleri, donanımsal gerçekleme devreleri ve performans özellikleri verilmiştir.

4.1. ÖNERİLEN (PROPOSED) ALGORİTMA

Yeni bir etkin bit indirgen çarpma algoritması önermek için, ikili aritmetikte Nikhilam Sutra algoritması üzerinde değişiklik yapıp, ayrıştırma ve bit öteleme gibi bazı temel özellikleri daha derinlemesine ele alacağız.

4.1.1. 4x4 bit Önerilen çarpma yöntemi

Önerilen algoritmaya dayanarak 4x4-bitlik çarpma işlemi, tek bir 2x2-bitlik çarpma işlemine indirgenebildiği gösterilmiştir.Sonuç olarak, bu algoritma eldenin yayılımında gerçekleşen gecikme süresini herhangi bir 4x4 çarpmadan daha çok azaltmaktadır.4 bitlik çarpma işlemi için önerilen algoritmanın genel mimari yapısı Şekil 4.1'de gösterilmektedir.

4.1.1.1. 4x4 bitlik Önerilen çarpma yönteminin algoritması

Ön işleme aşamasında, çarpan ve çarpılan ikili sayılar en az önemli ardışık sıfir bitlerini kaldırmak için sağa doğrudan kaydırılır. Bu da, çarpan ve çarpılan bitlerin sayısını düşürerek hesaplama süresini azaltır. Kaldırılan sıfir bitlerinin etkisi, son çıktının eşit sayılardaki bitlerle sola kaydırılmasıyla daha verimli bir şekilde birleştirilir. Aşağıda algoritmada açıklandığı üzere, 4 bit çarpan ve çarpılanın ikisi de ön işlem aşamasından sonra, sayıları en az 3-bitliğe azaltmak için Nikhilam Sutra uygulanır. Eğer elde edilen sayılar tam tamına 3-bitlik sayılarsa, çarpmayı herhangi bir standart 2x2 bit çarpma

işlemine indirgeyen sutra yöntemi uygulanır. Başka bir durum da, ön işlem aşamasından sonra elde edilen sayılar 4-bitlik ve 3-bitlik sayılarsa, büyük sayıyı ikili değer 1000'den çıkartarak 3-bitlik bir sayıya azaltılır. Bu, tüm çarpmayı bir kaydırma ve toplama işlemini izleyen 3-bitlik bir çarpma işlemine düşürür. 3-bitlik bir çarpma işlemi, önceki durumda açıklandığı üzere 2-bitlik daha düşük çarpma işlemine azaltılır. Öte yandan, sayılardan biri 3-bitlik bir uzunlukta ve diğeri 2-bitlik bir uzunluktaysa, büyük sayıyı ikili değer 100'den çıkartarak 2-bitlik bir sayıya azaltılır. Bu, bir kaydırma ve toplama işlemiyle tüm çarpmayı 2-bitlik bir çarpma işlemine düşürür. Ön işlem aşaması 3-bitlik ya da 2-bitlik sayılar veriyorsa, sayılar daha önce açıklandığı gibi işlenir. Son olarak başka bir durum da sayılardan birinin 1 olduğu işlem sırasında oluşabilir. Bu durumda sonuç hep işlem aşaması sonrasında elde edilen sayının değerine bakılmaksızın diğer sayıya eşit olur. 4x4 çarpım işleminde izlenen tüm algoritma aşağıda gösterilmiştir. Burada Nikhilam Sutra'nın temel işlem ilkelerini, başka bazı temel aritmetik ayrıştırma ve bit öteleme gibi işlemlerle olan ilişkisiyle ele aldık. Önerilen çarpma algoritması, algoritma koşullarındaki adımlara bağlı olarak bazı değişikliklerle daha büyük sayılar için de genişletilebilir.

İki 4-bitlik sayının çarpılması işlemi için önerilen algoritma aşağıda yer almaktadır.

(a) İlk koşullar

Tanımlama: flag1 = flag2 = flag3 = flag4 = flag5= 0;

(b) Ön işleme

Input 4-bit binary numbers a and b

n1 = Number of least significant consecutive zeros in a

n2 = Number of least significant consecutive zeros in b

n = n1 + n2

a' = Right shift a by n1

b'= Right shift b by n2

(c) Ana işleme

1. IF (a' > 1000 & b' > 1000) THEN

    a' = 10000 – a'; b' = 10000 – b';

    flag1 = 1;

2. IF (a' > 100 & b' > 1000) THEN

b' = b' - 1000;

flag2 = 1;

[If b' > 100 & a' > 1000, THEN a' = a' – 1000;]

3. IF (a' > 100 & b' > 100) THEN

a' = 1000 – a'; b' = 1000 – b';

flag3 = 1;

4. IF (a' > 10 & b' > 100) THEN

b' = b' - 100;

flag4 = 1;

[If b' > 10 & a' > 100, THEN a' = a' – 100;]

5.IF (a' > 10 & b' > 10) THEN

a'=100 – a'; b'=100 – b';

flag5= 1;

6. IF (a' = 1) THEN p' = b' | IF (b' = 1) THEN p' = a'

GOTO Step 8

7. 2-bit çarpma işlemi: p' = a' * b';

8. IF(flag5 = 1) THEN

p'= [LHS=100-(a'+b')+ Co of RHS] | [RHS=(2 bit)p'];

9. IF (flag4 = 1) THEN

p'= a' * 100 + p';

10. IF (flag3 = 1) THEN

p'= [LHS=1000-(a'+b')+ Co of RHS] | [RHS=(3 bit)p'];

11. IF (flag2 = 1) THEN

p' = a' * 1000 + p' ;

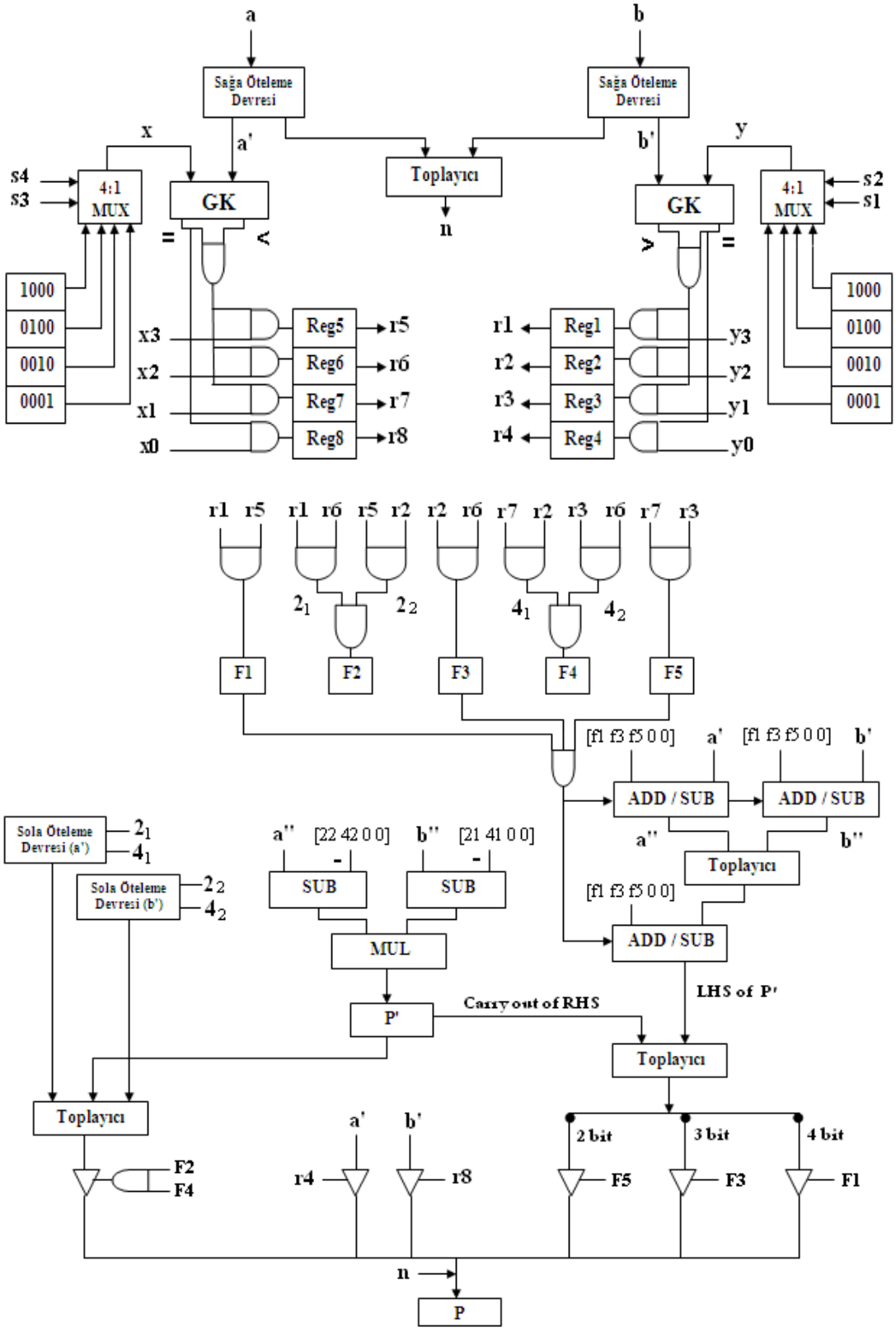
12. IF (flag1 = 1) THEN

p'= [LHS=10000-(a'+b')+ Co of RHS] | [RHS=(4 bit)p'];

13. p = p' değerini n bit sola ötele

14. p çarpımına dön

15. **Sonuç.**



Şekil 4.1 : 4x4 bitlik Önerilen çarpma algoritmasının mimari yapısı

Örnek 4.1:

a= $(1110)_2$ =$(14)_{10}$

b= $(1110)_2$ =$(14)_{10}$

Ön işleme:-

a' = 111 n1 = 1; a'yı 1bit sağa ötele çünkü sıfır sayısı birdir;

b' = 111 n2 = 1; b'yı 1bit sağa ötele çünkü sıfır sayısı birdir;

N= n1 + n2

= 1 + 1 = 2

Ana işleme:-

a' ve b' sayısı $(100)_2$ 'den büyük olduğu için 3.adım gerçekleşiyor.

3.adım

IF (a' > 100 & b' > 100) THEN

a' = 1000 – a'; b' = 1000 – b';

a' = 1000 - 0111; a'=0001;

b' =1000 – 0111; b'=0001;

p'=a' * b';

p'= 0001 * 0001;

p'(RHS)= 0001;

IF (flag3 = 1) THEN

p'= [LHS=1000-(a'+b')+ Co of RHS] | [RHS=(3 bit)p'];

(a'+b') = 0001+0001 = 0010

Co of RHS= 0

P'= [1000 – 0010 + 0] | [001]

P'= [110] | [001]

P'= 110001 Şimdi nihai sonucu N değerine göre sola öteleyeceğiz.

N= 2

P = 11000100 $(196)_{10}$

Örnek 4.2:

a= $(1100)_2$ =$(12)_{10}$

b= $(1010)_2$ =$(10)_{10}$

Ön işleme:-

a' = 11 n1 = 2; a'yı 2bit sağa ötele çünkü sıfir sayısı ikidir;

b' = 101 n2 = 1; b'yı 1bit sağa ötele çünkü sıfir sayısı birdir;

N = n1 + n2

= 2 + 1 = 3

Ana işleme:-

a' sayısı $(10)_2$ 'den ve b' sayısı $(100)_2$ 'den büyük olduğu için 4.adım gerçekleşiyor.

4.adım

IF (a' > 10 & b' > 100) THEN

b' = b' - 100;

b' = 101 – 100

b'= 01

p'= a' * b'

p'= 11 * 01

p'(RHS)= 0011

IF (flag4 = 1) THEN

p'= a' * 100 + p';

p'= (11 * 100) +0011

p'= 1100 + 0011

p'=1111 Şimdi nihai sonucu N değerine göre sola öteleyeceğiz.

N= 3

P = 1111000 $(120)_{10}$

4.1.2. 8x8 bit Önerilen çarpma algoritması

4 bitlik Önerilen çarpma algoritmasında, algoritma koşullarındaki adımlara bağlı olarak bazı değişikliklerle daha büyük sayılar için de uzatılabilir. İki 8-bitlik sayının çarpılması işlemi için önerilen algoritma aşağıda yer almaktadır.

(a) İlk koşullar

Tanımlama: flag1 = flag2 = flag3 = flag4 = flag5 = flag6 = flag7 = 0

flag8 = flag9 = flag10 = flag11 = flag12 = flag13 = 0

(b) Ön işleme

Input 8-bit binary numbers a and b

n1 = Number of least significant consecutive zeros in a

n2 = Number of least significant consecutive zeros in b

n = n1 + n2

a'= Right shift a by n1

b'= Right shift b by n2

(c) Ana işleme

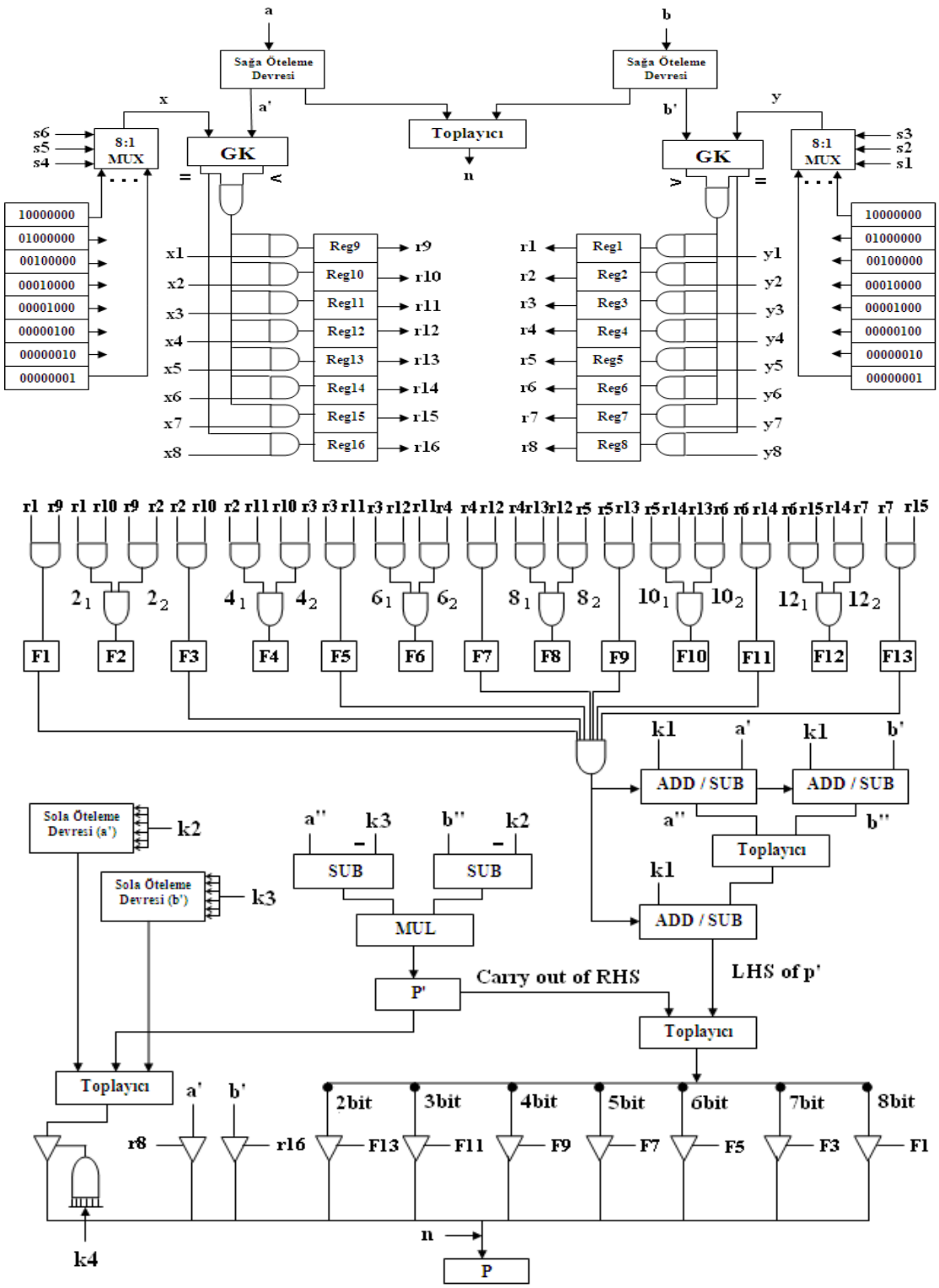
1. IF (a' > 80 & b' > 80) THEN
   a'=100-a'; b=100-b';
   flag1=1;
2. IF (a' > 40 & b' > 80) THEN
   b'=b'-80;
   flag2=1;
   [IF (b' > 40 & a' > 80) THEN a'=a'-80;]
3. IF (a' > 40 & b' > 40) THEN
   a'=80-a'; b=80-b';
   flag3=1;
4. IF (a' > 20 & b' > 40) THEN
   b'=b'-40;
   flag4=1;
   [IF (b' > 20 & a' > 40) THEN a'=a'-40;]
5. IF (a' > 20 & b' > 20) THEN
   a'=40-a'; b'=40-b';
   flag5=1;

6. IF (a' > 10 & b' > 20) THEN
   b'=b'-20;
   flag6=1;
   [IF (b' > 10 & a' > 20) THEN a'=a'-20;]
7. IF (a' > 10 & b' > 10) THEN
   a'=20-a'; b=20-b';
   flag7=1;
8. IF (a' > 08 & b' > 10) THEN
   b'=b'-10;
   flag8=1;
   [IF (b' > 08 & a' > 10) THEN a'=a'-10;]
9. IF (a' > 08 & b' > 08) THEN
   a'=10-a'; b=10-b';
   flag9=1;
10. IF (a' > 04 & b' > 08) THEN
   b'=b'-08;
   flag10=1;
   [IF (b' > 04 & a' > 08) THEN a'=a'-08;]
11. IF (a' > 04 & b' > 04) THEN
   a'=08-a'; b=08-b';
   flag11=1;
12. IF (a' > 02 & b' > 04) THEN
   b'=b'-04;
   flag12=1;
   [IF (b' > 02 & a' > 04) THEN a'=a'-04;]
13. IF (a' > 02 & b' > 02) THEN
   a'=04-a'; b=04-b';
   flag13=1;
14. IF (a'=01) THEN p'=b' | IF (b'=01) THEN p'=a'
   GOTO Step 16
15. 4-bit çarpma işlemi: p'=a'*b';

16. IF (flag13= 1) THEN
    p'= [LHS=04-(a'+b')+ Co of RHS] | [RHS=(2 bit)p'];
17. IF (flag12= 1) THEN
    p'=a'*04+(a'*b');
18. IF (flag11= 1) THEN
    p'= [LHS=08-(a'+b')+ Co of RHS] | [RHS=(3 bit)p'];
19. IF (flag10= 1) THEN
    p'=a'*08+(a'*b');
20. IF (flag9= 1) THEN
    p'= [LHS=10-(a'+b')+ Co of RHS] | [RHS=(4 bit)p'];
21. IF (flag8= 1) THEN
    p'=a'*10+(a'*b');
22. IF (flag7= 1) THEN
    p'= [LHS=20-(a'+b')+ Co of RHS] | [RHS=(5 bit)p'];
23. IF (flag6= 1) THEN
    p'=a'*20+(a'*b');
24. IF (flag5= 1) THEN
    p'= [LHS=40-(a'+b')+ Co of RHS] | [RHS=(6 bit)p'];
25. IF (flag4= 1) THEN
    p'=a'*40+(a'*b');
26. IF (flag3= 1) THEN
    p'= [LHS=80-(a'+b')+ Co of RHS] | [RHS=(7 bit)p'];
27. IF (flag2= 1) THEN
    p'=a'*80+(a'*b');
28. IF (flag1= 1) THEN
    p'= [LHS=100-(a'+b')+ Co of RHS] | [RHS=(8 bit)p'];

29. p= p' değerini n bit sola ötele
30. p çarpımına dön
31. **Sonuç.**

8 bitlik çarpma işlemi için önerilen çarpma devresinin genel mimari yapısı Şekil 4.2'de gösterilmektedir.



Şekil 4.2 : 8x8 bitlik Önerilen çarpma algoritmasının mimari yapısı

Yukarıda görüldüğü gibi şeklin karmışklığından dolayı bazı (k1,k2,k3,k4) gibi semboller kullanılmaktadır.Bu sembolleri açıklarsak k1[f1, f3, f5, f7, f9, f11, f13, 0, 0], k2 [2_2, 4_2, 6_2, 8_2, 10_2, 12_2, 0, 0], k3 [2_1, 4_1, 6_1, 8_1, 10_1, 12_1, 0, 0] ve k4 [f2, f4, f6, f8, f10, f12] değerlerini ifade eder.

Örnek 4.3:

a= $(1100\ 0000)_2$ =$(C0)_{16}$ =$(192)_{10}$ b= $(1100\ 0000)_2$ =$(C0)_{16}$ =$(192)_{10}$

Ön işleme:-

a' = 11 n1 = 6; a'yı 1bit sağa ötele çünkü sıfir sayısı birdir;

b' = 11 n2 = 6; b'yı 1bit sağa ötele çünkü sıfir sayısı birdir;

n = n1 + n2

= 12

Ana işleme:-

a' ve b' sayısı $(10)_2$ 'den büyük olduğu için 13.adım gerçekleşiyor.

13.adım

IF (a' > (02)16 & b' > (02)16) THEN

a' = 100 – a'; b' = 100 – b';

a'= 0100 - 0011; a' = 0001;

b'= 0100 - 0011; b' = 0001;

p' = a' * b';

p' = 0001 * 0001;

p'(RHS) = 0001;

IF (flag13 = 1) THEN

p'= [LHS=100-(a'+b')+ Co of RHS] | [RHS=(2 bit)p'];

(a'+b') = 0001+0001 = 0010

Co of RHS= 0

P'= [0100 – 0010 + 0] | [01]

P'= [10] | [01]

P'= 1001 Şimdi nihai sonucu n değerine göre sola öteleyeceğiz.

N= 12

P = 1001 0000 0000 0000 $(9000)_{16}$ $(36864)_{10}$

4.1.3. 16x16 bit Önerilen çarpma algoritması

İki 16-bitlik sayının çarpılması işlemi için önerilen algoritma aşağıda yer almaktadır.

(a) İlk koşullar

Tanımlama: flag (29 downto 1) = 0;

(b) Ön işleme

Input 16-bit binary numbers a and b

n1 = Number of least significant consecutive zeros in a

n2 = Number of least significant consecutive zeros in b

n = n1 + n2

a'= Right shift a by n1

b'= Right shift b by n2

(c) Ana işleme

1. IF (a' > 8000 & b' > 8000) THEN
   a'=10000-a'; b=10000-b';
   flag1=1;
2. IF (a' > 4000 & b' > 8000) THEN
   b'=b'-8000; flag2=1;

[IF (b' > 4000 & a' > 8000) THEN a'=a'-8000;]

3. IF (a' > 4000 & b' > 4000) THEN
   a'=8000-a'; b=8000-b';
   flag3=1;
4. IF (a' > 2000 & b' > 4000) THEN
   b'=b'- 4000; flag4=1;

[IF (b' > 2000 & a' > 4000) THEN a'=a'-4000;]

5. IF (a' > 2000 & b' > 2000) THEN
   a'=4000-a'; b'=4000-b';
   flag5=1;
6. IF (a' > 1000& b' > 2000) THEN
   b'=b' -2000; flag6=1;

[IF (b' > 1000 & a' > 2000) THEN a'=a'-2000;]

7. IF (a' > 1000 & b' > 1000) THEN
   a'=2000-a'; b=2000-b';

flag7=1;

8. IF (a' > 800& b' > 1000) THEN

b'=b'-1000; flag8=1;

[IF (b' > 800 & a' > 1000) THEN a'=a'-1000;]

9. IF (a' > 800 & b' > 800) THEN

a'=1000-a'; b=1000-b';

flag9=1;

10. IF (a' > 400 & b' > 800) THEN

b'=b'-800; flag10=1;

[IF (b' > 400 & a' > 800) THEN a'=a'-800;]

11. IF (a' > 400 & b' > 400) THEN

a'=800-a'; b=800-b';

flag11=1;

12. IF (a' > 200 & b' > 400) THEN

b'=b'- 400; flag12=1;

[IF (b' > 200 & a' > 400) THEN a'=a'-400;]

13. IF (a' > 200 & b' > 200) THEN

a'=400-a'; b=400-b';

flag13=1;

14. IF (a' > 100 & b' > 200) THEN

b'=b'-200; flag14=1;

[IF (b' > 100 & a' > 200) THEN a'=a'-200;]

15. IF (a' > 100 & b' > 100) THEN

a'=200-a'; b=200-b';

flag15=1;

16. IF (a' > 80& b' > 100) THEN

b'=b' -100; flag16=1;

[IF (b' > 80 & a' > 100) THEN a'=a'-100;]

17. IF (a' > 80 & b' > 80) THEN

a'=100-a'; b=100-b';

flag17=1;

18. IF (a' > 40 & b' > 80) THEN

b'=b'-80; flag18=1;

[IF (b' > 40 & a' > 80) THEN a'=a'-80;]

19. IF (a' > 40 & b' > 40) THEN
    a'=80-a'; b'=80-b';
    flag19=1;

20. IF (a' > 20 & b' > 40) THEN
    b'=b'- 40; flag20=1;

[IF (b' > 20 & a' > 40) THEN a'=a'-40;]

21. IF (a' > 20 & b' > 20) THEN
    a'=40-a'; b=40-b';
    flag21=1;

22. IF (a' > 10 & b' > 20) THEN
    b'=b'-20; flag22=1;

[IF (b' > 10 & a' > 20) THEN a'=a'-20;]

23. IF (a' > 10 & b' > 10) THEN
    a'=20-a'; b=20-b';
    flag23=1;

24. IF (a' > 08 & b' > 10) THEN
    b'=b'-10;
 flag24=1;

[IF (b' > 08 & a' > 10) THEN a'=a'-10;]

25. IF (a' > 08 & b' > 08) THEN
    a'=10-a'; b=10-b';
    flag25=1;

26. IF (a' > 04 & b' > 08) THEN
    b'=b'-08;
   flag26=1;

[IF (b' > 04 & a' > 08) THEN a'=a'-08;]

27. IF (a' > 04 & b' > 04) THEN
    a'=08-a'; b=08-b';
    flag27=1;

28. IF (a' > 02 & b' > 04) THEN
    b'=b'-04;
  flag28=1;

[IF (b' > 02 & a' > 04) THEN a'=a'-04;]

29. IF (a' > 02 & b' > 02) THEN
    a'=04-a'; b=04-b';
    flag29=1;
30. IF (a'=0001) THEN p'=b' | IF (b'=0001) THEN p'=a'
    GOTO Step 32
31. 8-bit çarpma işlemi: p'=a'*b';
32. IF (flag29= 1) THEN
    p'= [LHS=04-(a'+b')+ Co of RHS] | [RHS=(2 bit)p'];
33. IF (flag28= 1) THEN
    p'=a'*04+(a'*b');
34. IF (flag27= 1) THEN
    p'= [LHS=08-(a'+b')+ Co of RHS] | [RHS=(3 bit)p'];
35. IF (flag26= 1) THEN
    p'=a'*08+(a'*b');
36. IF (flag25= 1) THEN
    p'= [LHS=10-(a'+b')+ Co of RHS] | [RHS=(4 bit)p'];
37. IF (flag24= 1) THEN
    p'=a'*10+(a'*b');
38. IF (flag23= 1) THEN
    p'= [LHS=20-(a'+b')+ Co of RHS] | [RHS=(5 bit)p'];
39. IF (flag22= 1) THEN
    p'=a'*20+(a'*b');
40. IF (flag21= 1) THEN
    p'= [LHS=40-(a'+b')+ Co of RHS] | [RHS=(6 bit)p'];
41. IF (flag20= 1) THEN
    p'=a'*40+(a'*b');
42. IF (flag19= 1) THEN
    p'= [LHS=80-(a'+b')+ Co of RHS] | [RHS=(7 bit)p'];
43. IF (flag18= 1) THEN
    p'=a'*80+(a'*b');
44. IF (flag17= 1) THEN
    p'= [LHS=100-(a'+b')+ Co of RHS] | [RHS=(8 bit)p'];

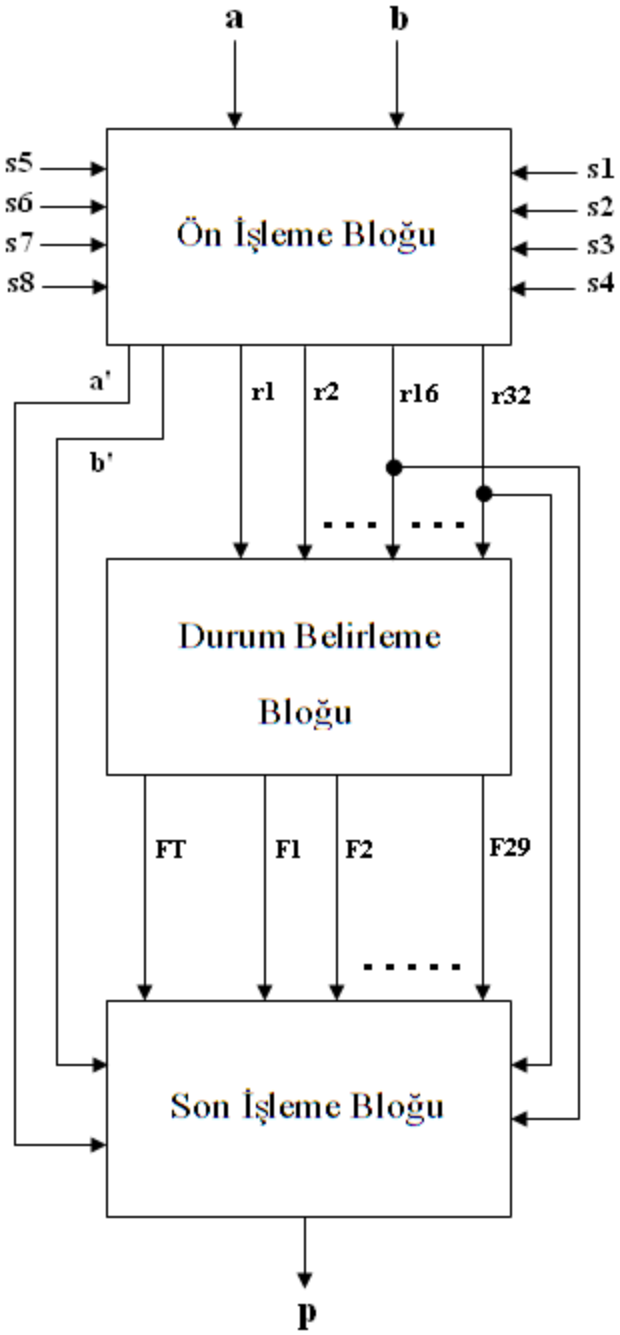
45. IF (flag16= 1) THEN
    p'=a'*100+(a'*b');
46. IF (flag15= 1) THEN
    p'= [LHS=200-(a'+b')+ Co of RHS] | [RHS=(9 bit)p'];
47. IF (flag14= 1) THEN
    p'=a'*200+(a'*b');
48. IF (flag13= 1) THEN
    p'= [LHS=400-(a'+b')+ Co of RHS] | [RHS=(10 bit)p'];
49. IF (flag12= 1) THEN
    p'=a'*400+(a'*b');
50. IF (flag11= 1) THEN
    p'= [LHS=800-(a'+b')+ Co of RHS] | [RHS=(11bit)p'];
51. IF (flag10= 1) THEN
    p'=a'*800+(a'*b');
52. IF (flag9= 1) THEN
    p'= [LHS=1000-(a'+b')+ Co of RHS] | [RHS=(12 bit)p'];
53. IF (flag8= 1) THEN
    p'=a'*1000+(a'*b');
54. IF (flag7= 1) THEN
    p'= [LHS=2000-(a'+b')+ Co of RHS] | [RHS=(13 bit)p'];
55. IF (flag6= 1) THEN
    p'=a'*2000+(a'*b');
56. IF (flag5= 1) THEN
    p'= [LHS=4000-(a'+b')+ Co of RHS] | [RHS=(14 bit)p'];
57. IF (flag4= 1) THEN
    p'=a'*4000+(a'*b');
58. IF (flag3= 1) THEN
    p'= [LHS=8000-(a'+b')+ Co of RHS] | [RHS=(15 bit)p'];
59. IF (flag2= 1) THEN
    p'=a'*8000+(a'*b');
60. IF (flag1= 1) THEN
    p'= [LHS=10000-(a'+b')+ Co of RHS] | [RHS=(16 bit)p'];

61. p= p' değerini n bit sola ötele

62. p çarpımına dön

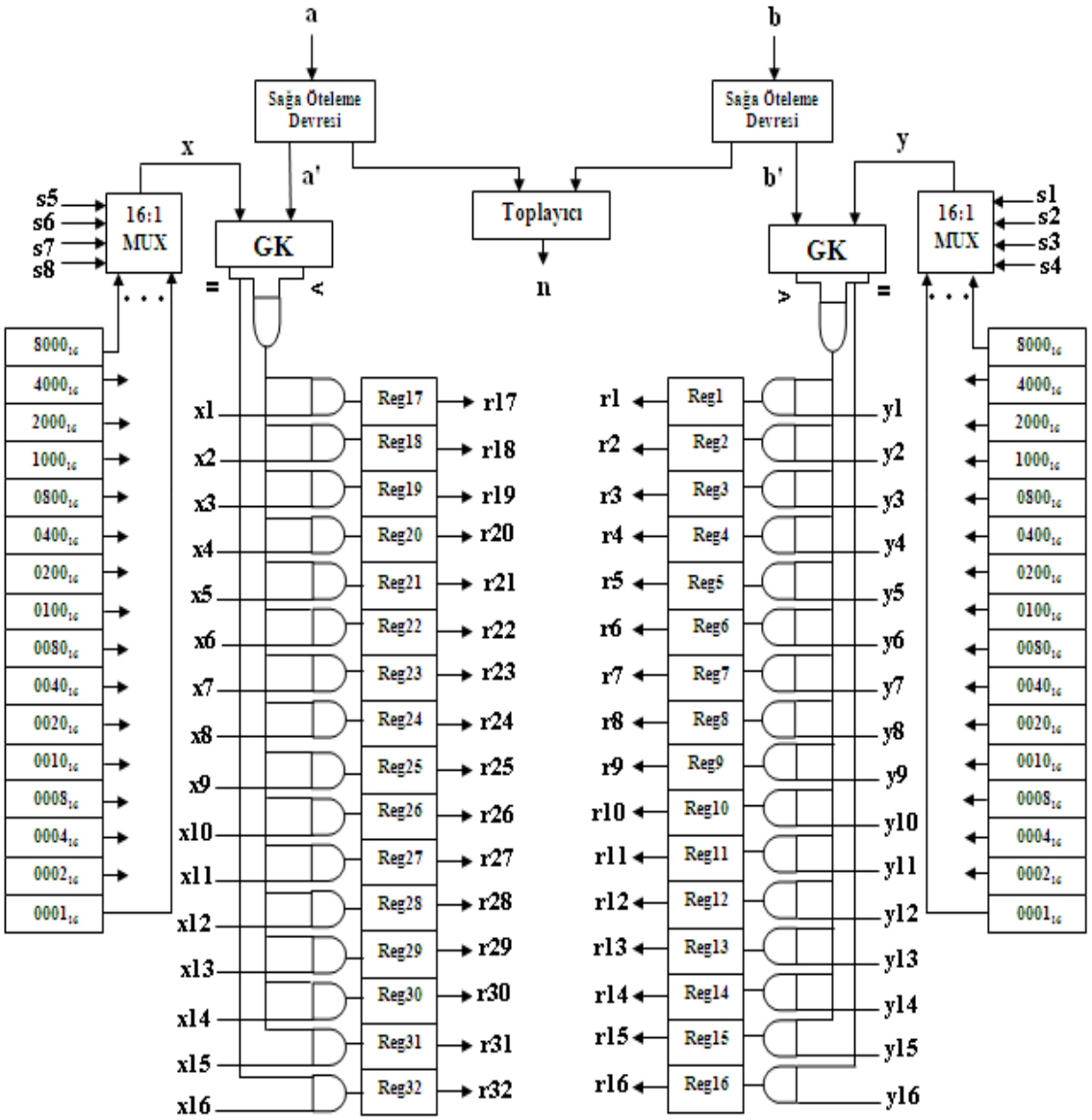
63. **Sonuç.**

16 bitlik çarpma işlemi için önerilen çarpma devresinin genel tasarımı Şekil 4.3'te gösterilmektedir.

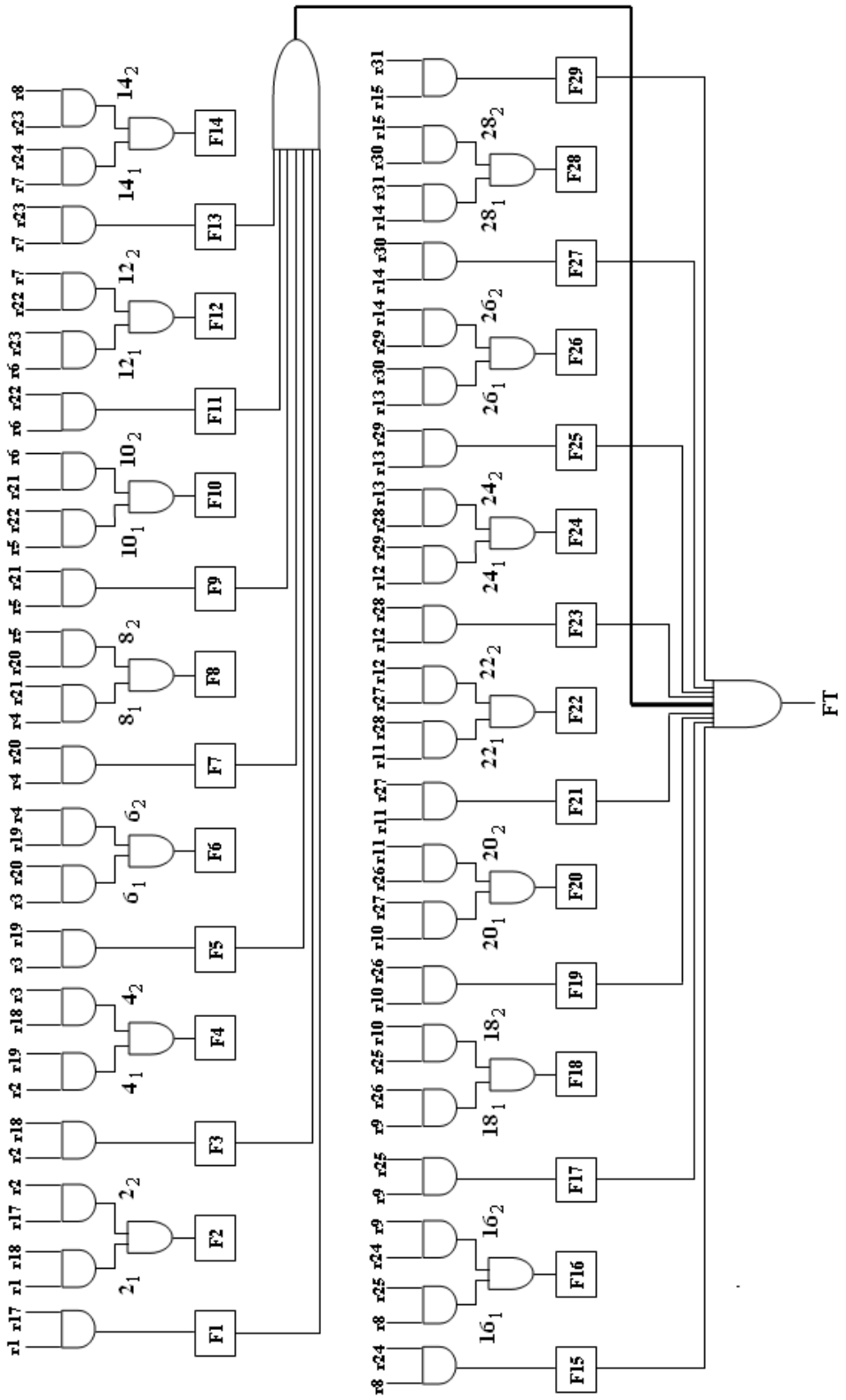


Şekil 4.3 : 16x16 bitlik Önerilen çarpma devresinin genel tasarımı

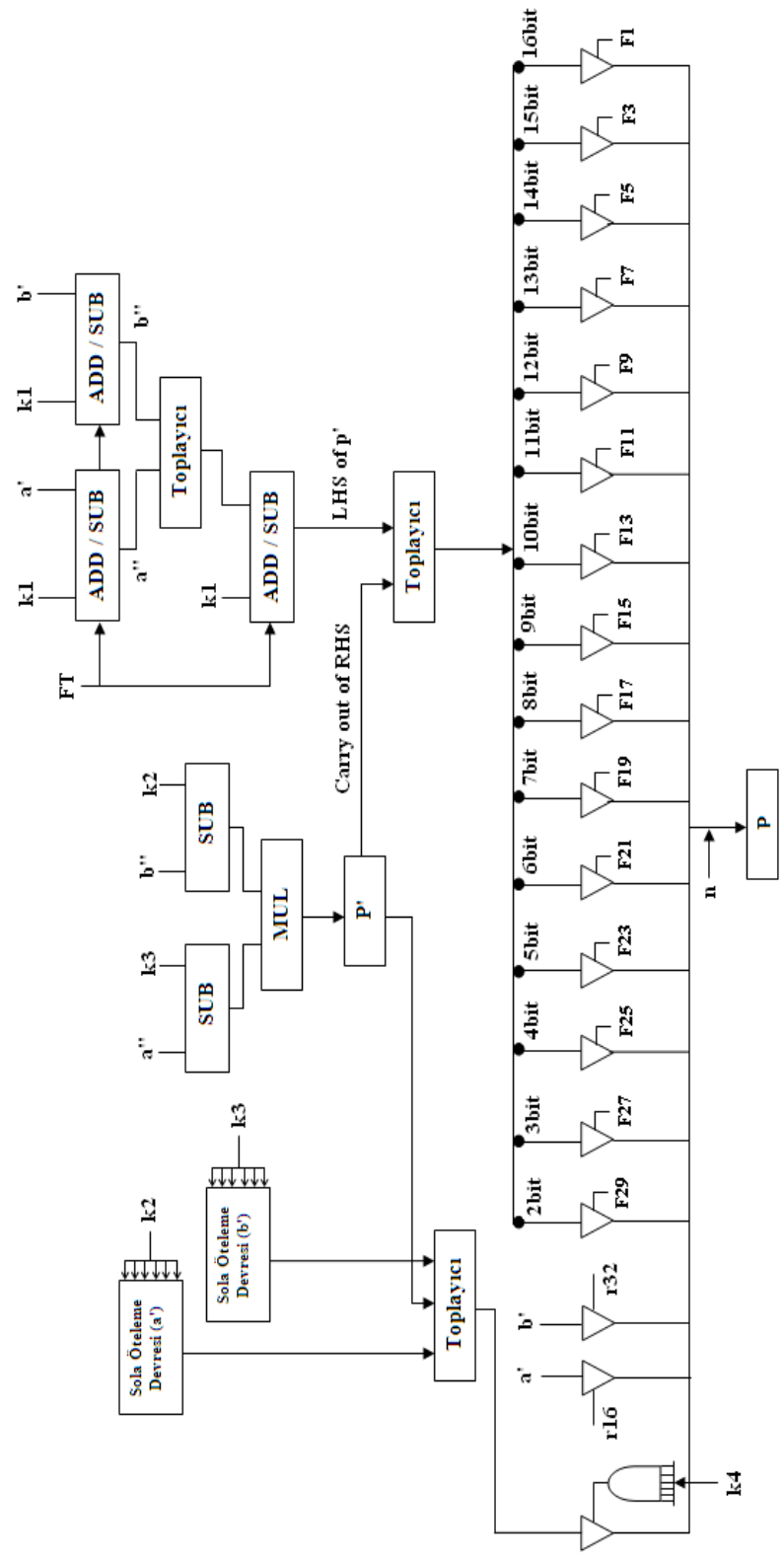
Şekil 4.3'ü daha detaylı göstermeye kalkarsak ancak şekli 3 parçaya bölmemiz lazım, aşağıda görüldüğü gibi 16x16 bitlik Önerilen çarpma devresinin genel tasarımını 3 ayrı garfik halinde gösterilmektedir.Şeklin karmışklığından dolayı bazı (k1, k2, k3, k4) gibi semboller kullanılmaktadır.Bu sembolleri açıklarsak k1[f1, f3, f5, f7, f9, f11, f13, f15, f17, f19, f21, f23, f25, f27, f29, 0, 0], k2[2_2, 4_2, 6_2, 8_2, 10_2, 12_2, 14, 16_2, 18_2, 20_2, 22_2, 24_2, 26_2, 28_2, 0, 0], k3 [2_1, 4_1, 6_1, 8_1, 10_1, 12_1, 14_1, 16_1, 18_1, 20_1, 22_1, 24_1, 26_1, 28_1, 0, 0] ve k4[f2, f4, f6, f8, f10, f12, f14, f16, f18, f20, f22, f24, f26, f28] değerlerini ifade eder.



Şekil 4.4 : Ön işleme bloğunun detaylı gösterimi



Şekil 4.5 : Durum belirleme bloğunun detaylı tasarımı



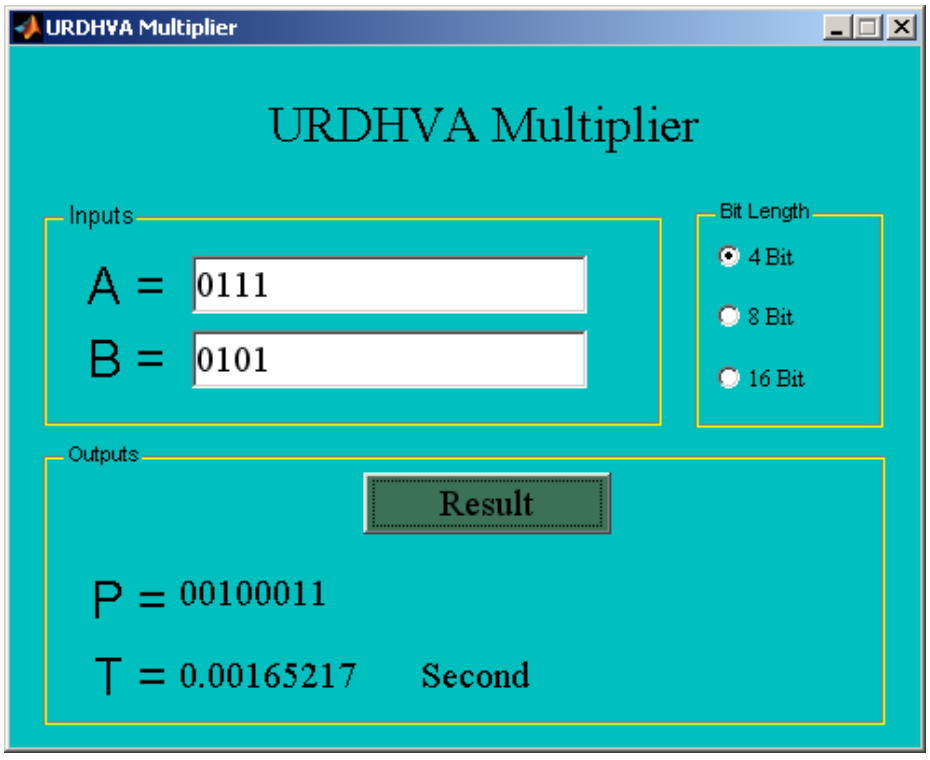
Şekil 4.6 : Son işleme bloğunun detaylı gösterimi

4.2. ÇARPMA ALGORİTMALARININ MATLAB SİMÜLASYONLARI VE PERFORMANS ANALİZLERİ

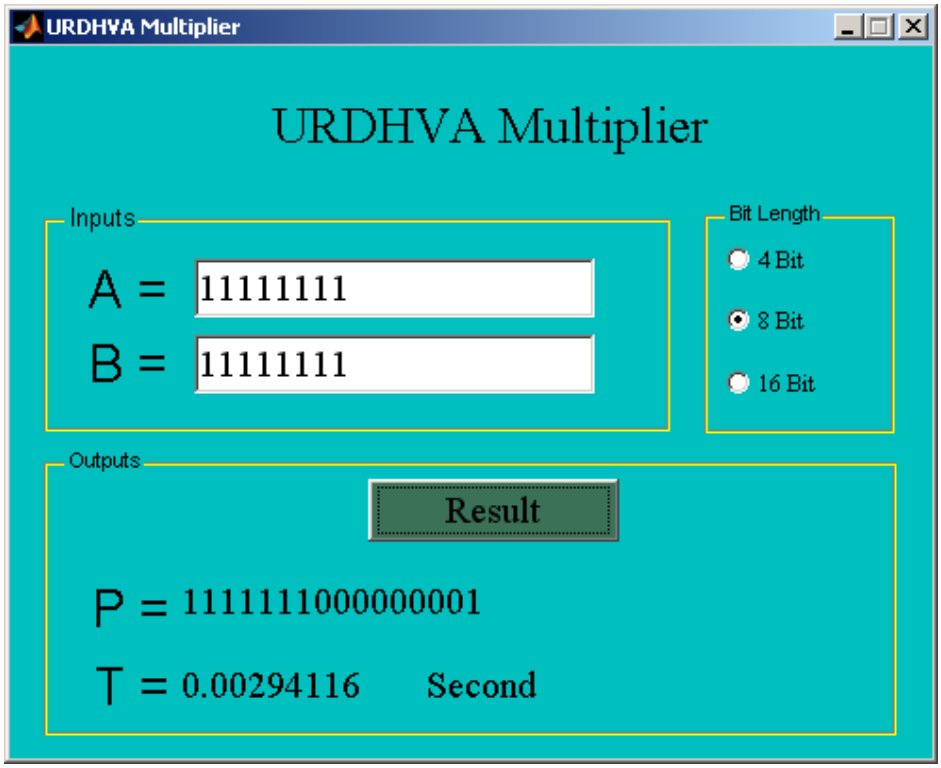
Bu bölümde, Vedic matematiğine dayanan Urdhva çarpma algoritması, Önerilen algoritması ve klasik ama çok kullanılan Booth algoritmasının, MATLAB simülasyonları yapılarak karşılaştırmalı performans analizleri sunulmuştur. Performans ölçütü olarak algoritmaların yazılımsal gecikme süreleri esas alınmıştır.

4.2.1. Urdhva Çarpma Algoritmasının MATLAB Simülasyonları ve Performansları

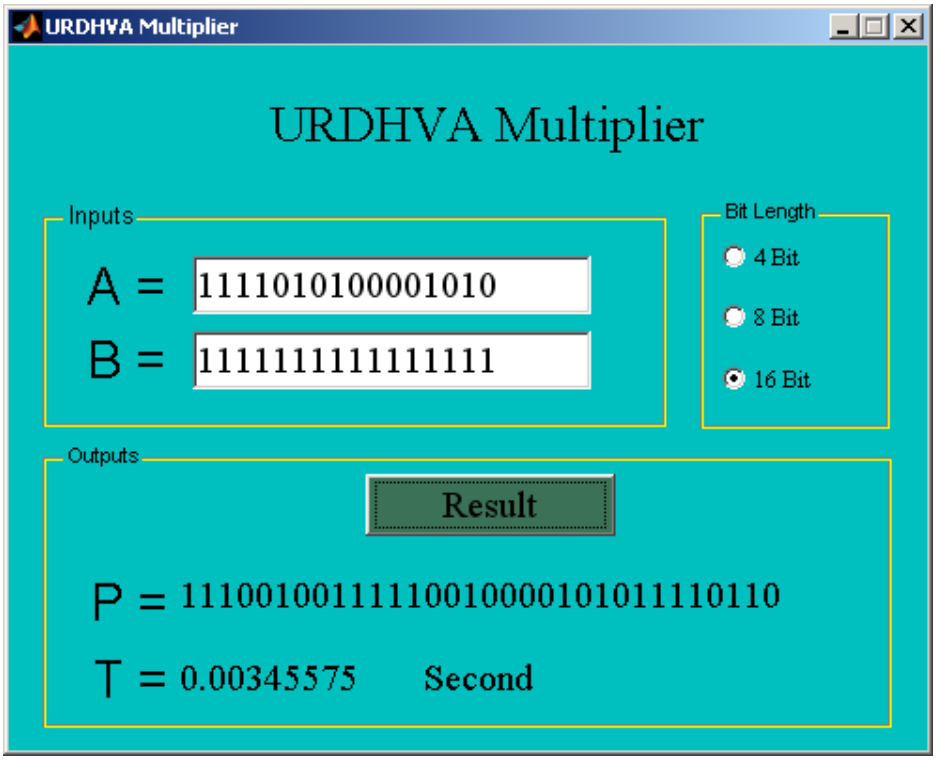
Urdhva çarpma algoritmasının MATLAB simülasyonları 4, 8 ve 16 bitlik operandlar için sırasıyla Şekil 4.7, Şekil 4.8 ve Şekil 4.9'da gösterilmiştir.MATLAB grafiksel kullanıcı arayüzü GUI(Graphic User Interfaces) kütüphanesi kullanılarak çarpma algoritmaların bit uzunluğuna göre arayüzü geliştirilmiştir.Şekillerde görüldüğü gibi (A , B) girişleri, (P) çıkışı ve en son olarak (T) dediğimiz girişten çıkışa yazılımsal gecikme süresini göstermektedir.



Şekil 4.7 : 4 bitlik Urdhva algoritmasının MATLAB simülasyon sonucu



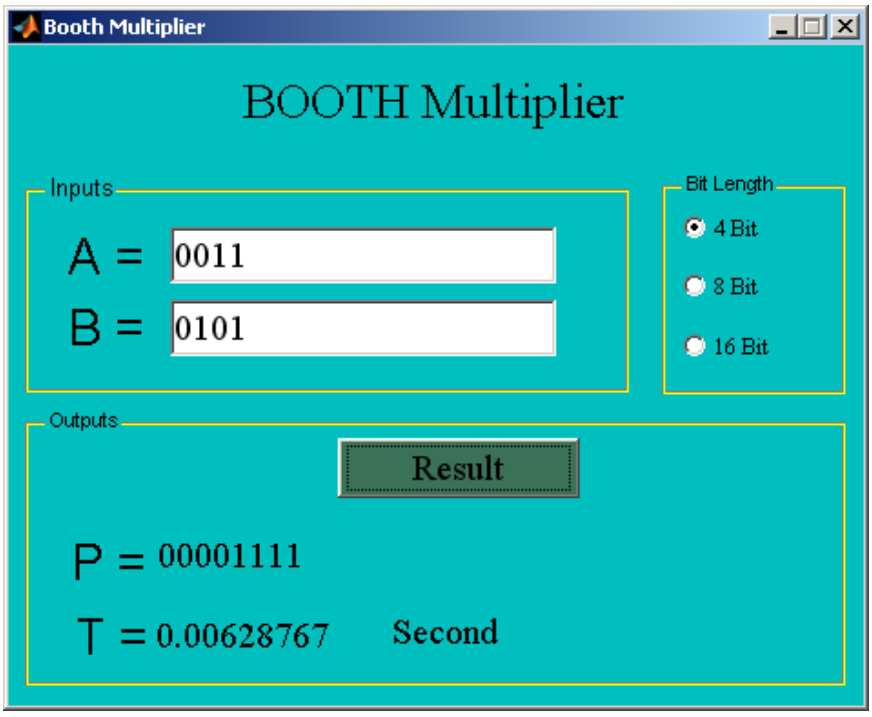
Şekil 4.8 : 8 bitlik Urdhva algoritmasının MATLAB simülasyon sonucu



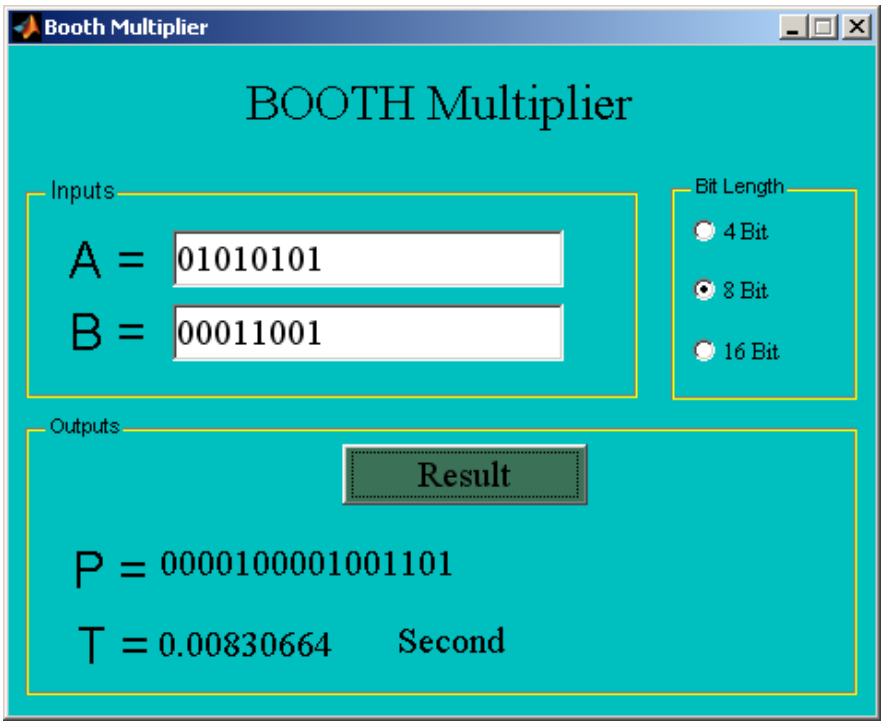
Şekil 4.9 : 16 bitlik Urdhva algoritmasının MATLAB simülasyon sonucu

4.2.2. Booth Çarpma Algoritmasının MATLAB Simülasyonları ve Performansları

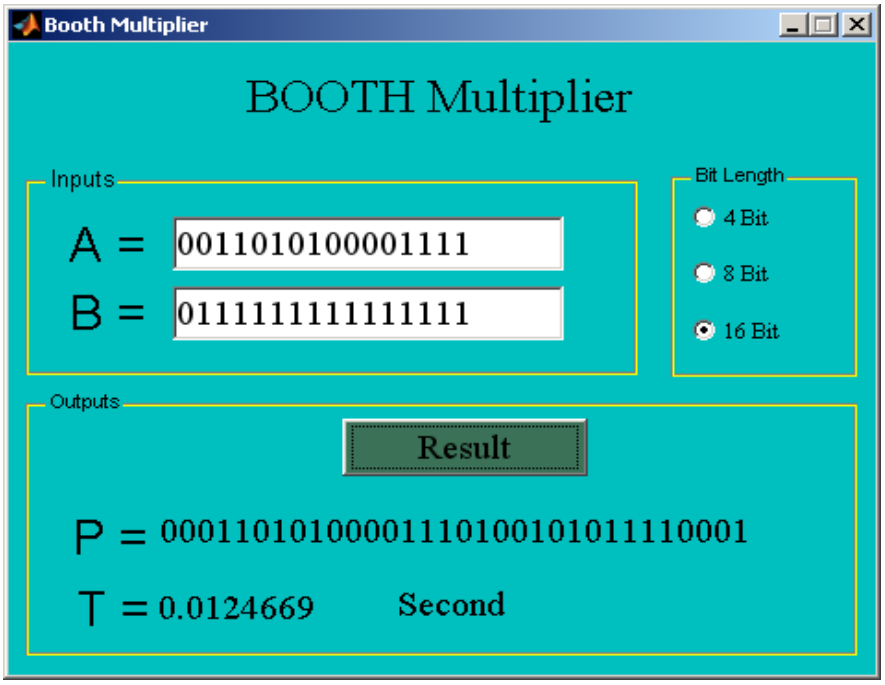
Booth çarpma algoritmasının MATLAB simülasyonları 4, 8 ve 16 bitlik operandlar için sırasıyla Şekil 4.10, Şekil 4.11 ve Şekil 4.12'de gösterilmiştir.



Şekil 4.10 : 4 bitlik Booth algoritmasının MATLAB simülasyon sonucu



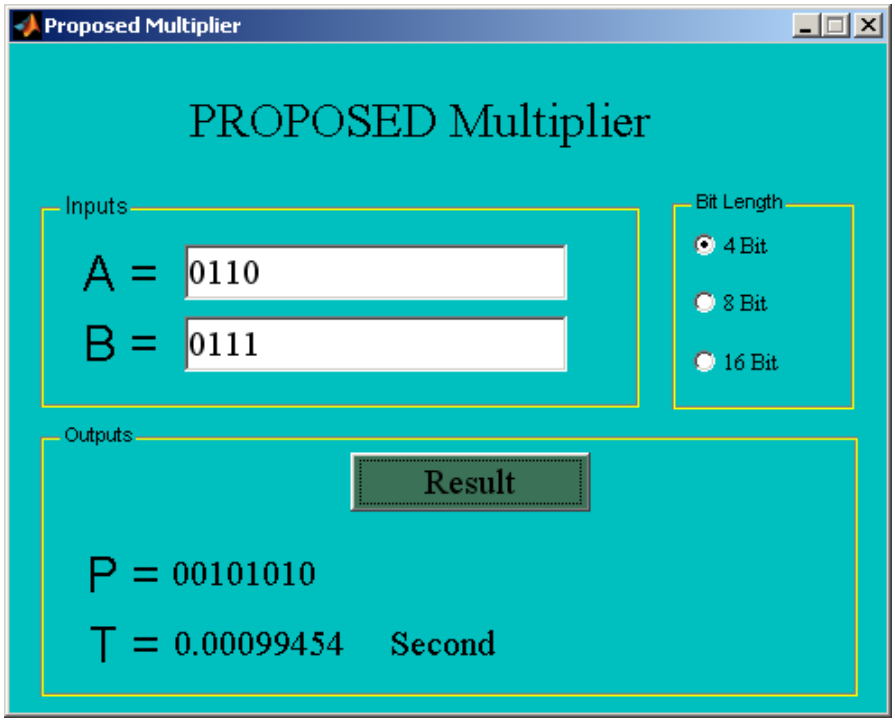
Şekil 4.11 : 8 bitlik Booth algoritmasının MATLAB simülasyon sonucu



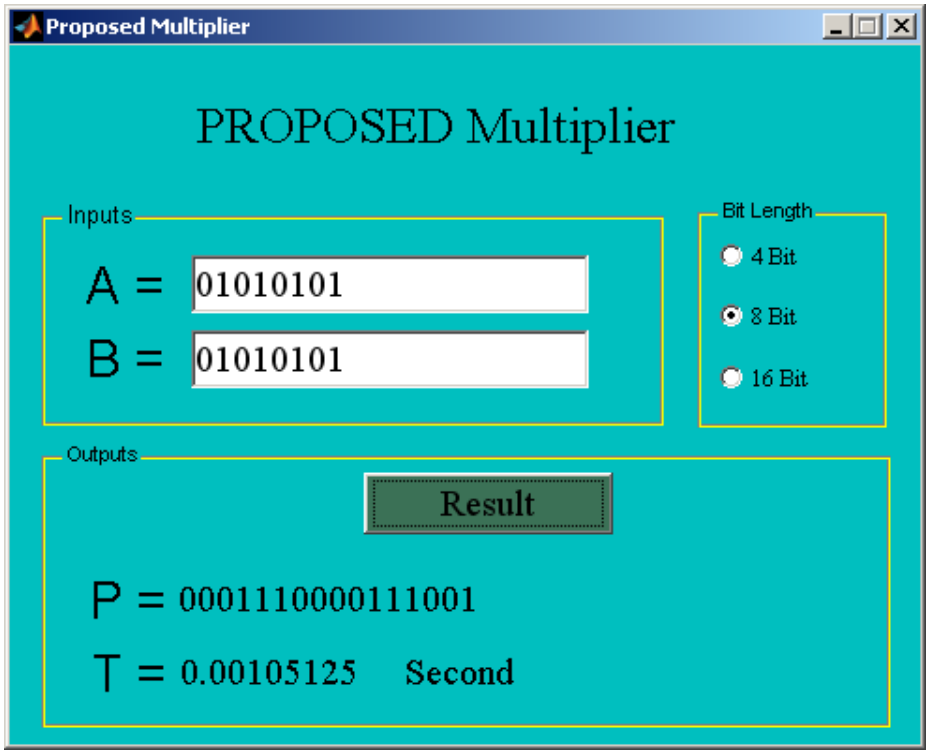
Şekil 4.12 : 16 bitlik Booth algoritmasının MATLAB simülasyon sonucu

4.2.3. Önerilen Çarpma Algoritmasının MATLAB Simülasyonları ve Performansları

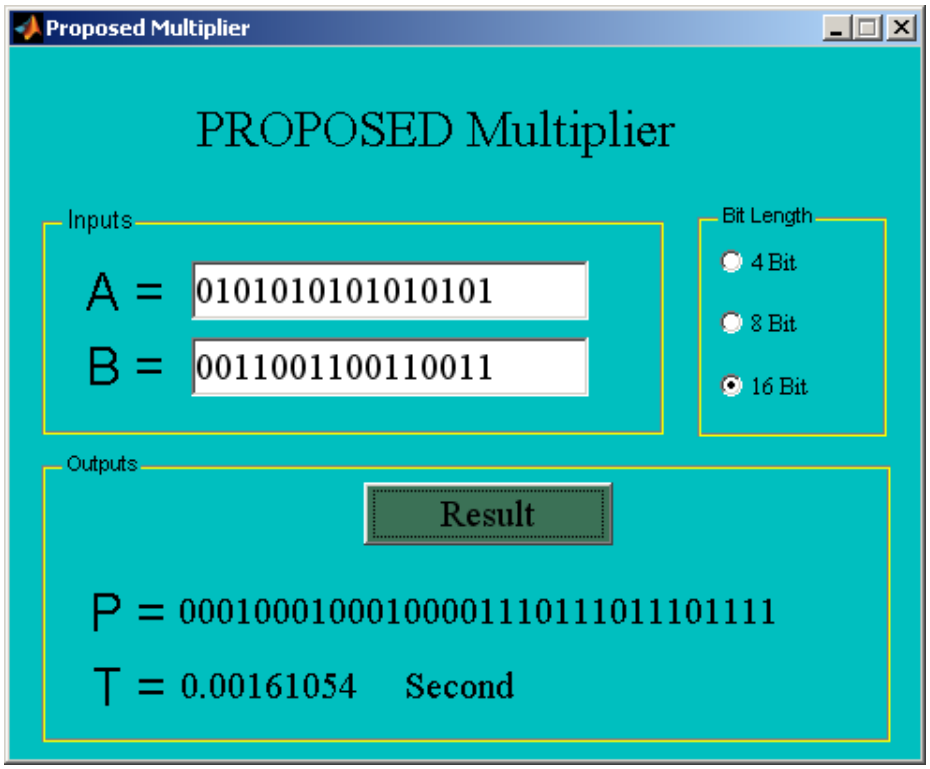
Önerilen çarpma algoritmasının MATLAB simülasyonları 4, 8 ve 16 bitlik operandlar için sırasıyla Şekil 4.13, Şekil 4.14 ve Şekil 4.15'te gösterilmiştir.



Şekil 4.13 : 4 bitlik Önerilen algoritmasının MATLAB simülasyon sonucu



Şekil 4.14 : 8 bitlik Önerilen algoritmasının MATLAB simülasyon sonucu

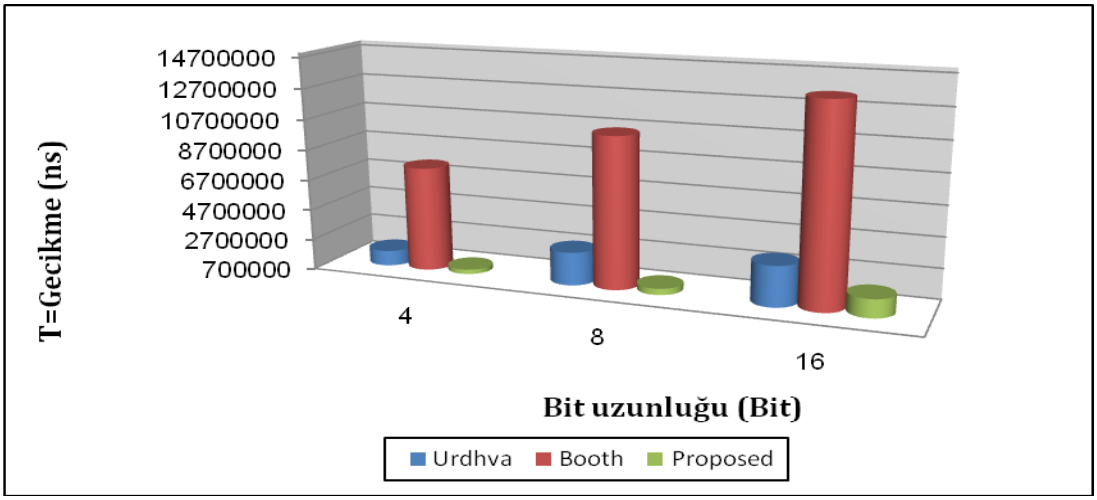


Şekil 4.15 : 16 bitlik Önerilen algoritmasının MATLAB simülasyon sonucu

4.2.4. Çarpma Algoritmalarının MATLAB Performans karşılaştırmaları

MATLAB Performans ölçütü olarak girişten çıkışa çarpma algoritmalarının yazılımsal gecikme süreleri esas alınmıştır. Çarpma işlemlerinde performans değerlendirmesi yapılan toplama işlemlerinin performansına göre yapılmaktadır. Çünkü çarpma işlemeleri toplama ve öteleme işlemleri ile gerçekleştirilmektedir. Bu işlemler içinde de, en fazla zaman harcayan işlem, toplama işlemidir. En kötü durumu düşünecek olursak, n bitlik iki sayının çarpılması için, n tane toplama işlemine gereksinim duyulmaktadır. (4,8,16) bitlik MATLAB Simülasyon sonuçlarına bakıldığında, Urdhva çarpma algoritmasının 4 bitlik sayılar için ortalama gecikmesi 0.0017 saniyedir,8 bitlik sayılar için ortalama gecikmesi 0.0028673 saniyedir, 16 bitlik sayılar için ortalama gecikmesi ise 0.0033442 saniyedir.Booth çarpma algoritmasının (4,8,16) bitlik sayılar için ortalama gecikmesi sırayla (0.007551 ,0.010576 ,0.013592) saniyedir.Önerilen çarpma algoritmasının (4,8,16) bitlik sayılar için ortalama gecikmesi ise, sırasıyla (0.00099621 ,0.00110931 ,0.00189944) saniyedir.

Bu sonuçlara ve grafiklara bakıldığında çarpma algoritmalarının en hızlısı Önerilen (Proposed) çarpma algoritmasıdır.En yavaş algoritma ise Booth algoritmasıdır.Şekil 4.16 'da çarpma algoritmalarının bit uzunluğuna bağlı yazılımsal gecikme süresi grafiği gösterilmektedir.



Şekil 4.16 : Çarpma devrelerinin yazılımsal gecikme değerlrinin karşılaştırılması

4.3. ÇARPMA ALGORİTMALARININ VHDL SİMÜLASYONLARI VE PERFORMANS ANALİZLERİ

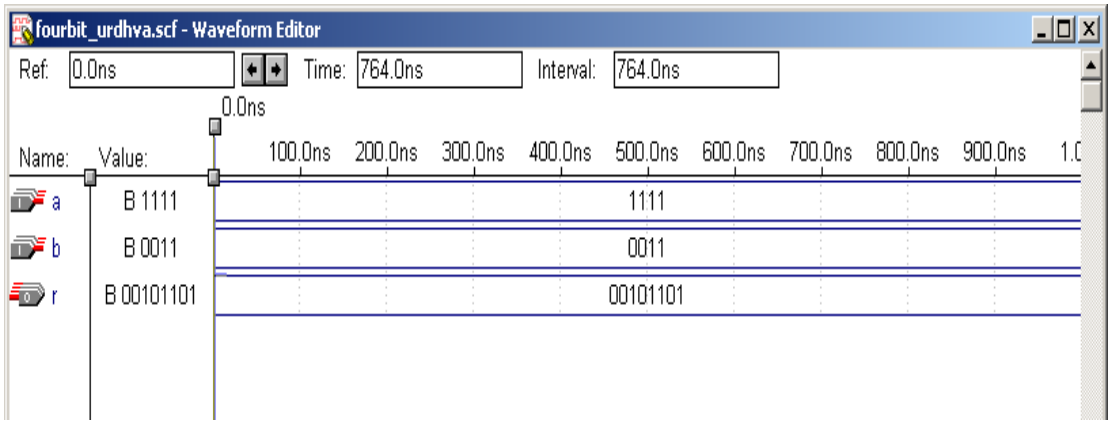
Bu bölümde, önerdiğimiz algoritmalara dayanarak geliştirilen donanımsal devre tasarımlarının VHDL aracığıyla simülasyonları yapılmıştır.Önerilen çarpma yöntemin devre performansı Urdhva ve Booth çarpma devreleri ile karşılaştırılmıştır.Tüm simülasyonlar MAX+plus II ortamında 3s100evq100-5 konfigürasyonu seçilerek yapılmıştır.Performans analizlerini gerçeklemek için ise, XILINX Spartan serisi Spartan 3E (XC3S100E, Paket VQ100, Hız -5) modeli kullanılmıştır.

4.3.1. Urdhva Çarpma Algoritmasının VHDL Simülasyonları ve Performansları

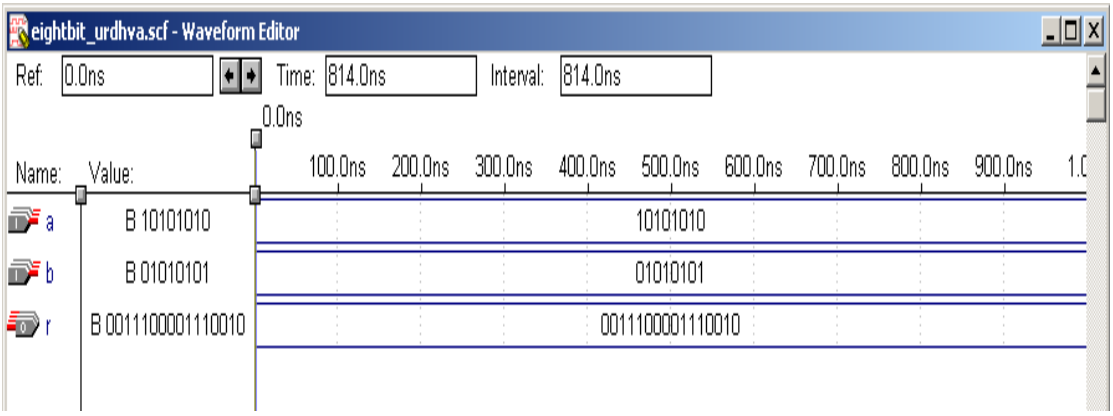
Urdhva çarpma algoritmasının (4, 8, 16) bitlik sayılarla VHDL ve MAX+plus kullanılarak elde edillen simülasyonları Şekil 4.17, 4.18, 4.19'daki grafiklerle gösterilmiştir. Burada, aritmetik çarpma devrelerinin fonksiyonel olarak doğrulanması görülmektedir.

Tablo 4.1 : Urdhva çarpma yönteminin donanımsal performansları

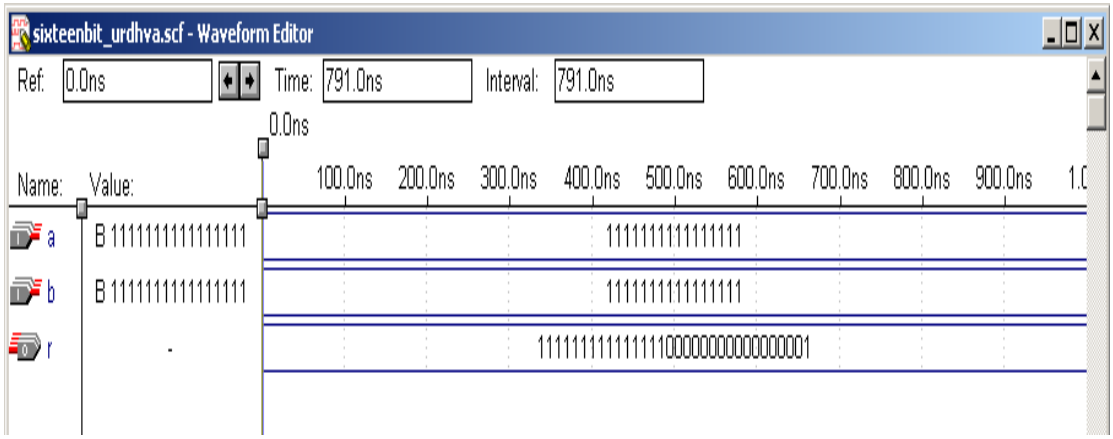
Bit Uzunluğu	**4 Bit**	**8 Bit**	**16 Bit**
Number of Slices (960)	16 1%	75 7%	394 41%
Number of Slice Flip Flops (1920)	0 0%	0 0%	0 0%
Number of 4 input LUTs (1920)	28 1%	133 6%	685 35%
Number of bonded IOBs (66)	16 24%	32 48%	64 96%
Number of GCLKs (24)	0 0%	0 0%	0 0%
Number of IOs	16	32	64
Maksimum Gecikme	11.005 ns	21.544 ns	40.731 ns
Toplam Kapı Sayısı	228	634	2621



Şekil 4.17 : (4 x 4 bit) Urdhva algoritmasının VHDL simülasyon sonucu



Şekil 4.18 : (8 x 8 bit) Urdhva algoritmasının VHDL simülasyon sonucu



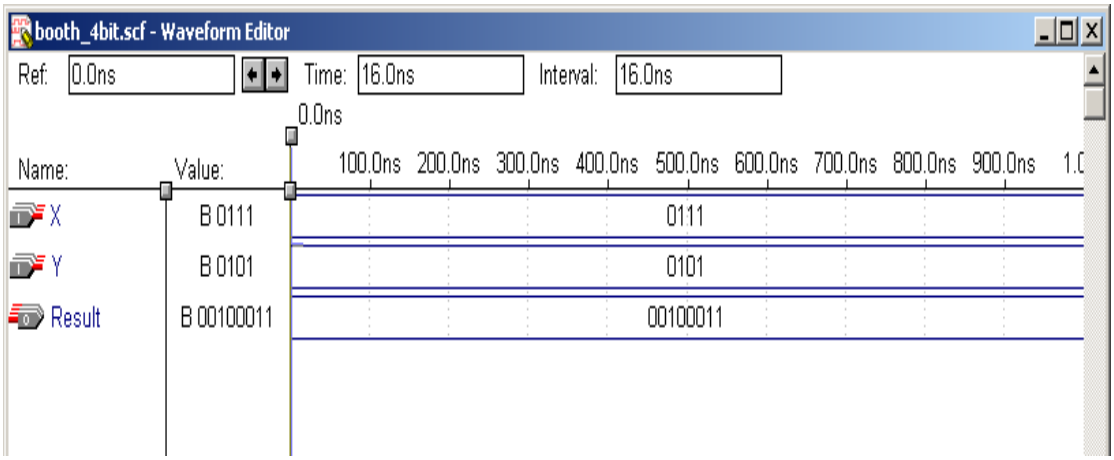
Şekil 4.19 : (16 x 16 bit) Urdhva algoritmasının VHDL simülasyon sonucu

4.3.2. Booth Çarpma Algoritmasının VHDL Simülasyonları ve Performansları

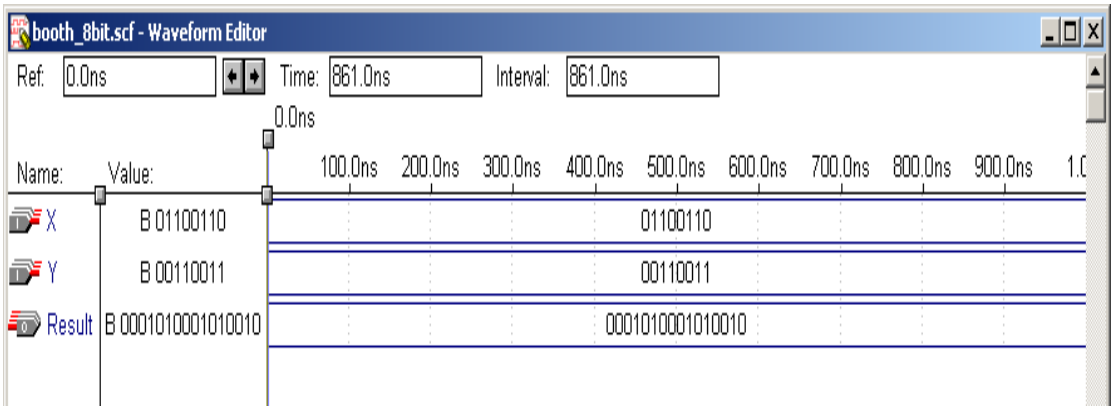
Booth çarpma algoritmasının (4, 8, 16) bitlik sayılarla VHDL ve MAX+plus kullanılarak elde edillen simülasyonları Şekil 4.20, 4.21, 4.22'deki grafiklerle gösterilmiştir.

Tablo 4.2 : Booth çarpma yönteminin donanımsal performansları

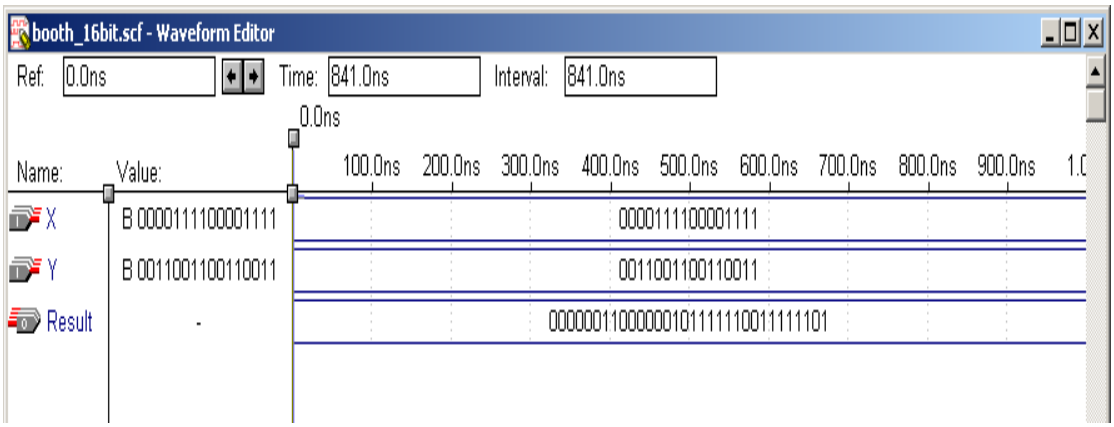
Bit Uzunluğu	**4 Bit**		**8 Bit**		**16 Bit**	
Number of Slices (960)	18	1%	93	9%	417	43%
Number of Slice Flip Flops (1920)	0	0%	0	0%	0	0%
Number of 4 input LUTs (1920)	32	1%	164	8%	732	38%
Number of bonded IOBs (66)	16	24%	32	48%	64	96%
Number of GCLKs (24)	0	0%	0	0%	0	0%
Number of IOs	16		32		64	
Maksimum Gecikme	12.767 ns		23.934 ns		41.883 ns	
Toplam Kapı Sayısı	217		592		2587	



Şekil 4.20 : (4x4 bit) Booth algoritmasının VHDL simülasyon sonucu



Şekil 4.21 : (8 x 8 bit) Booth algoritmasının VHDL simülasyon sonucu



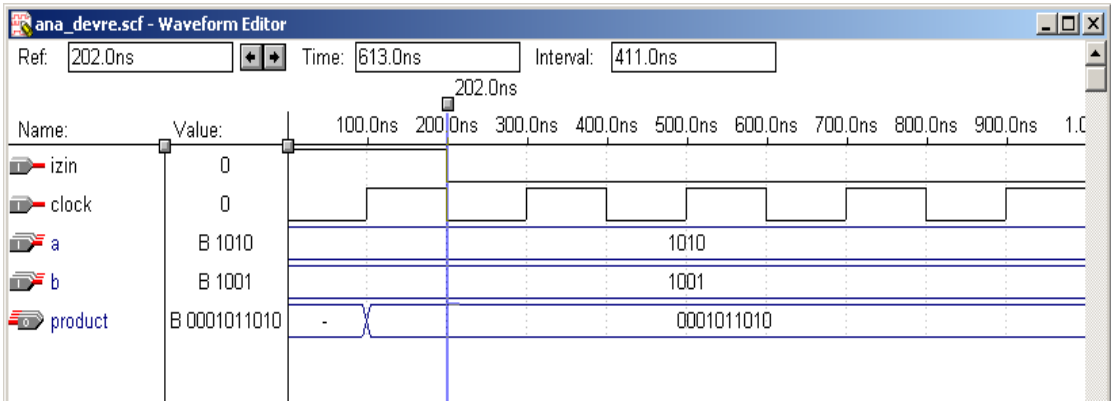
Şekil 4.22 : (16 x 16 bit) Booth algoritmasının VHDL simülasyon sonucu

4.3.3. Önerilen Çarpma Algoritmasının VHDL Simülasyonları ve Performansları

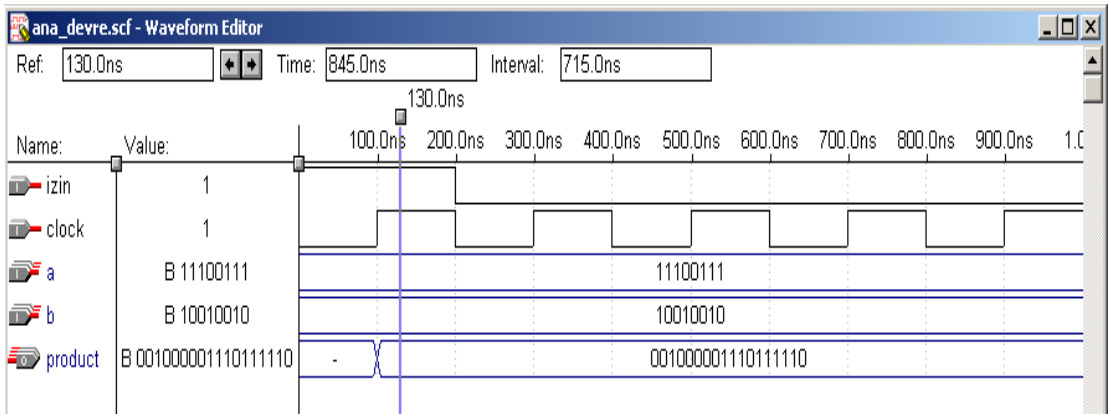
Önerilen çarpma algoritmasının (4, 8, 16) bitlik sayılarla VHDL kullanılarak elde edillen simülasyonları Şekil 4.23, 4.24, 4.25t'teki grafiklerle gösterilmiştir.

Tablo 4.3 : Önerilen çarpma yönteminin donanımsal performansları

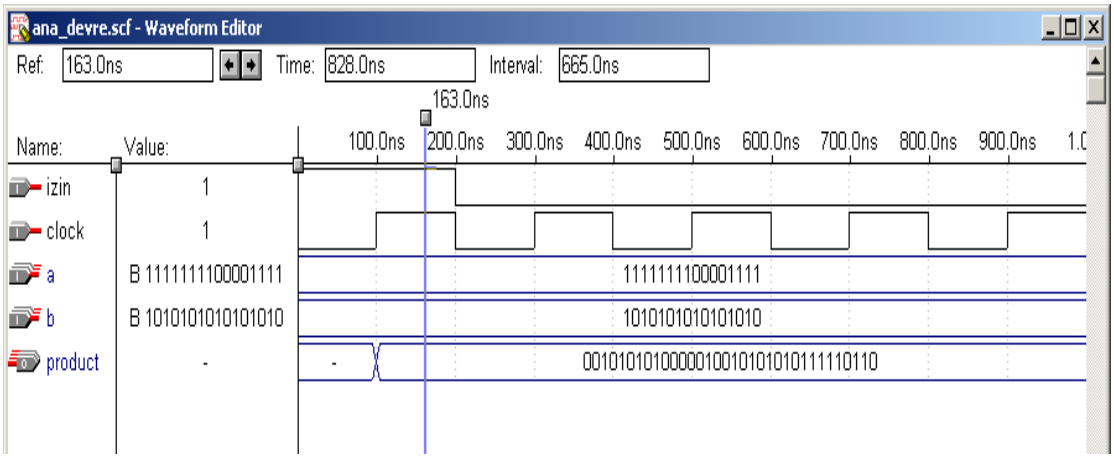
Bit Uzunluğu	**4 Bit**		**8 Bit**		**16 Bit**	
Number of Slices (960)	13	1%	47	4%	231	24%
Number of Slice Flip Flops (1920)	8	0%	15	0%	28	1%
Number of 4 input LUTs (1920)	27	1%	89	4%	412	21%
Number of bonded IOBs (66)	15	22%	27	40%	51	77%
Number of GCLKs (24)	2	8%	2	8%	2	8%
Number of IOs	18		34		66	
Maksimum Gecikme	6.947 ns		9.887 ns		28.821 ns	
Toplam Kapı Sayısı	260		1164		3116	



Şekil 4.23 : (4 x 4 bit) Önerilen algoritmasının VHDL simülasyon sonucu



Şekil 4.24 : (8 x 8 bit) Önerilen algoritmasının VHDL simülasyon sonucu



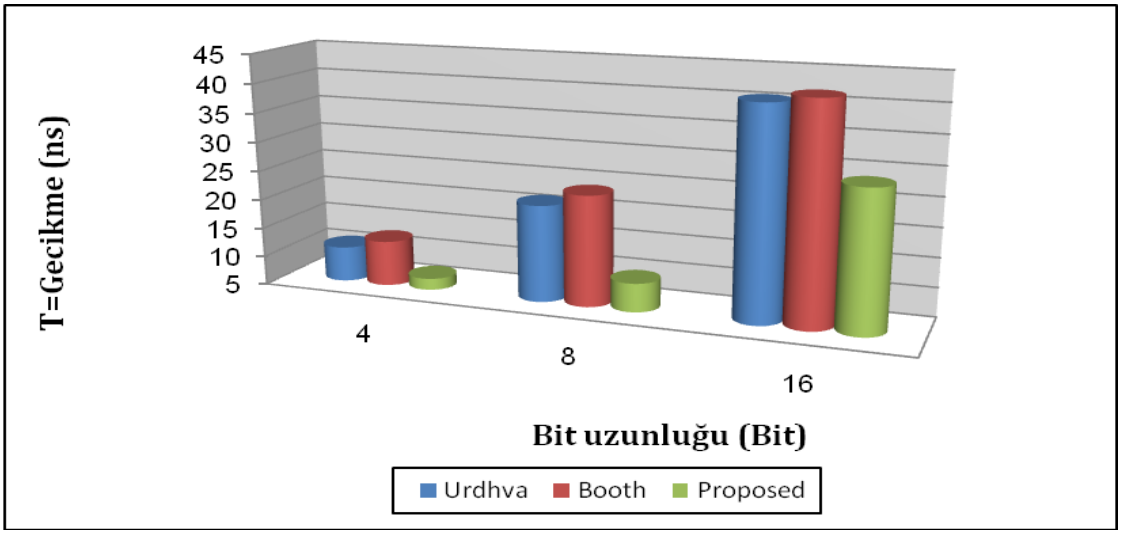
Şekil 4.25 : (16 x 16 bit) Önerilen algoritmasının VHDL simülasyon sonucu

4.3.4. Çarpma Yöntemlerinin Donanımsal Performans karşılaştırmaları

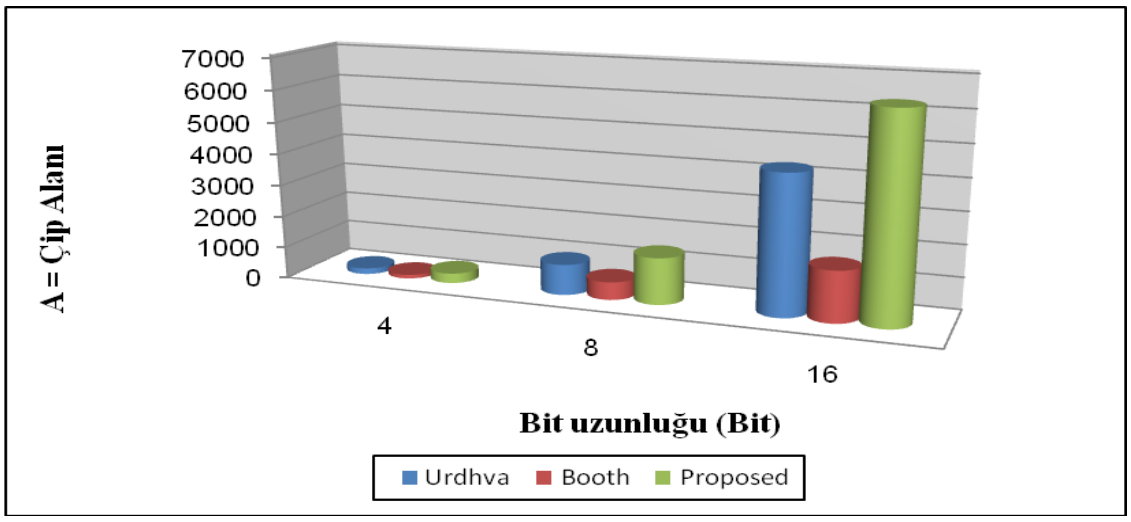
VHDL Performans ölçütü olarak girişten çıkışa çarpma algoritmaların en uzun gecikme süresi ve kullanılan toplam birim kapı sayısı (çip alanı) kriterleri esas alınmıştır. (4, 8, 16) bitlik VHDL Simülasyon sonuçlarına bakıldığında ise Urdhva çarpma algoritmasının 4 bitlik sayılar için gecikmesi 11.005 ns'dir,8 bitlik sayılar için gecikmesi 21.544 ns'dir, 16 bitlik sayılar için gecikmesi ise 40.731 ns'dir.Booth çarpma algoritmasının (4, 8, 16) bitlik sayılar için gecikmesi sırayla (12.767, 23.934, 41.883) ns'dir.Önerilen çarpma algoritmasının (4, 8, 16) bitlik sayılar için gecikmesi ise (6.947, 9.887, 28.821) ns'dir.

Çarpma algoritmalarının en hızlısı Önerilen(Proposed) Çarpma algoritmasıdır, fakat bu hız artışı ile beraber kullanılan fazla çip alanı ortaya çıkmaktadır. En yavaş algoritma olarak Booth çarpma algoritması daha yavaş çalışmaktadır.Ancak bu algoritma en az çip alanı gerektiren algoritmadır.Urdhva çarpma algoritması ise orta gecikme ve orta bir çip alanı gerektiren algoritmadır.

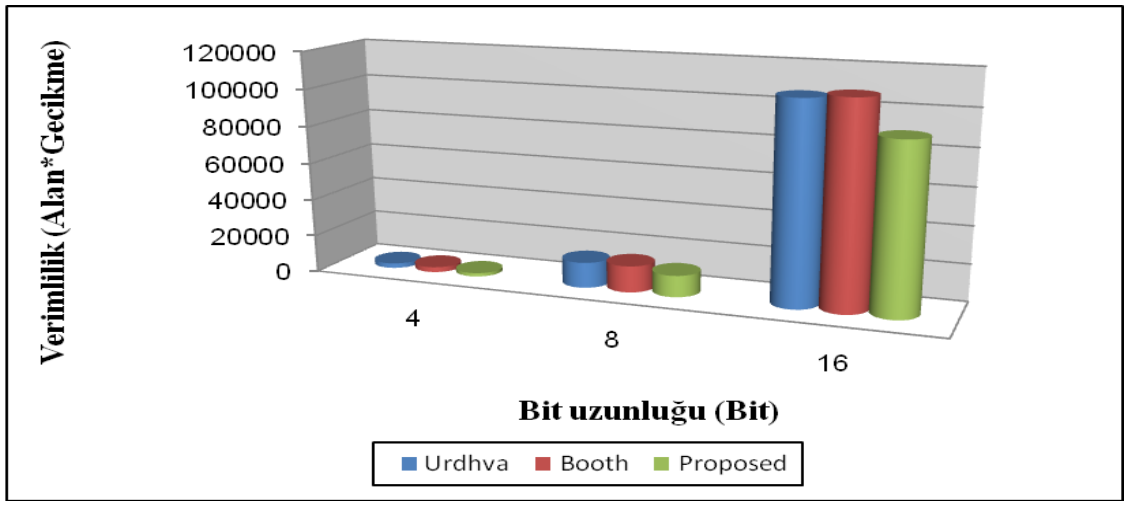
İşlem gecikme (T) grafiği oluşturulurken, algoritmaların iterasyon sayıları, oteleme sayıları, yeniden kodlama süreleri de göz önünde tutulmuştur.Şekil 4.26'da çarpma algoritmalarının bit uzunluğuna bağlı gecikme süresi, 4.27'de bit uzunluğuna bağlı gereksinim duyulan çip alanı, 4.28'de de bu iki değerin çarpımıyla elde edilmiş olan verimlilik AxT grafiği sunulmuştur.



Şekil 4.26 : Çarpma Algoritmalarının gecikme (T) grafiği



Şekil 4.27 : Çarpma Algoritmalarının çip alanı (A) grafiği

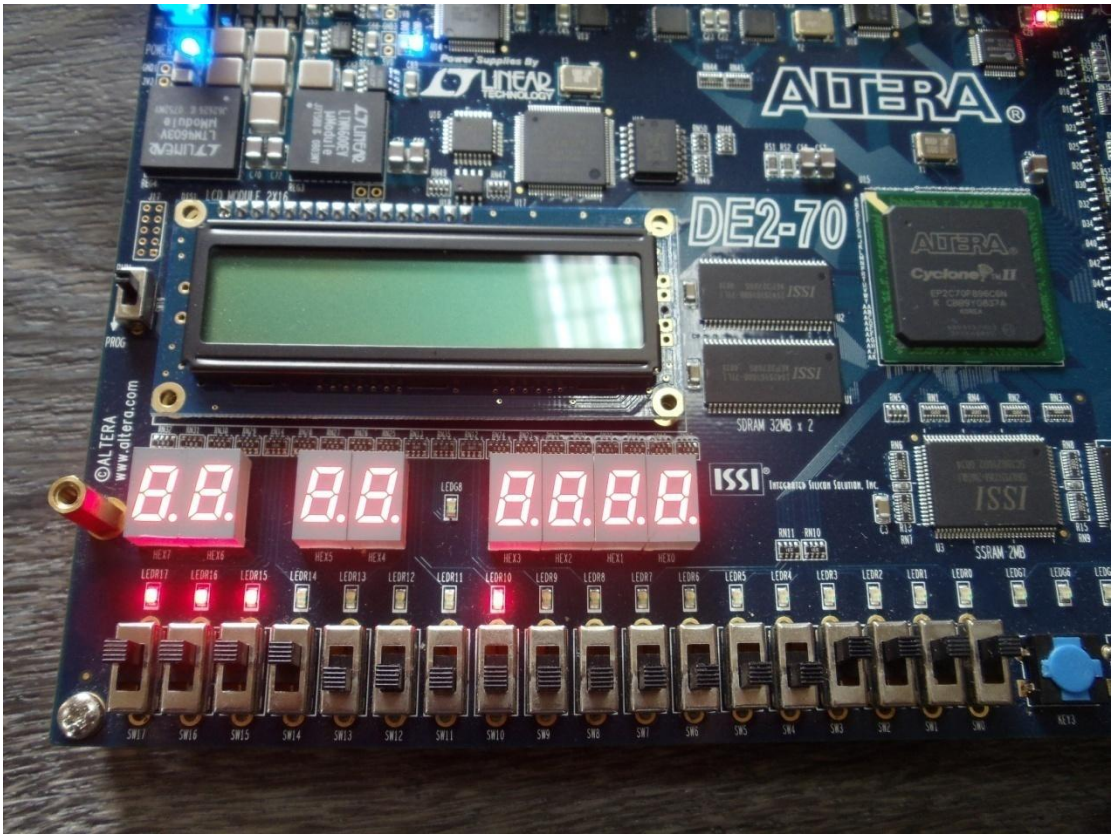


Şekil 4.28 : Çarpma Algoritmalarının verimlilik (AxT) grafiği

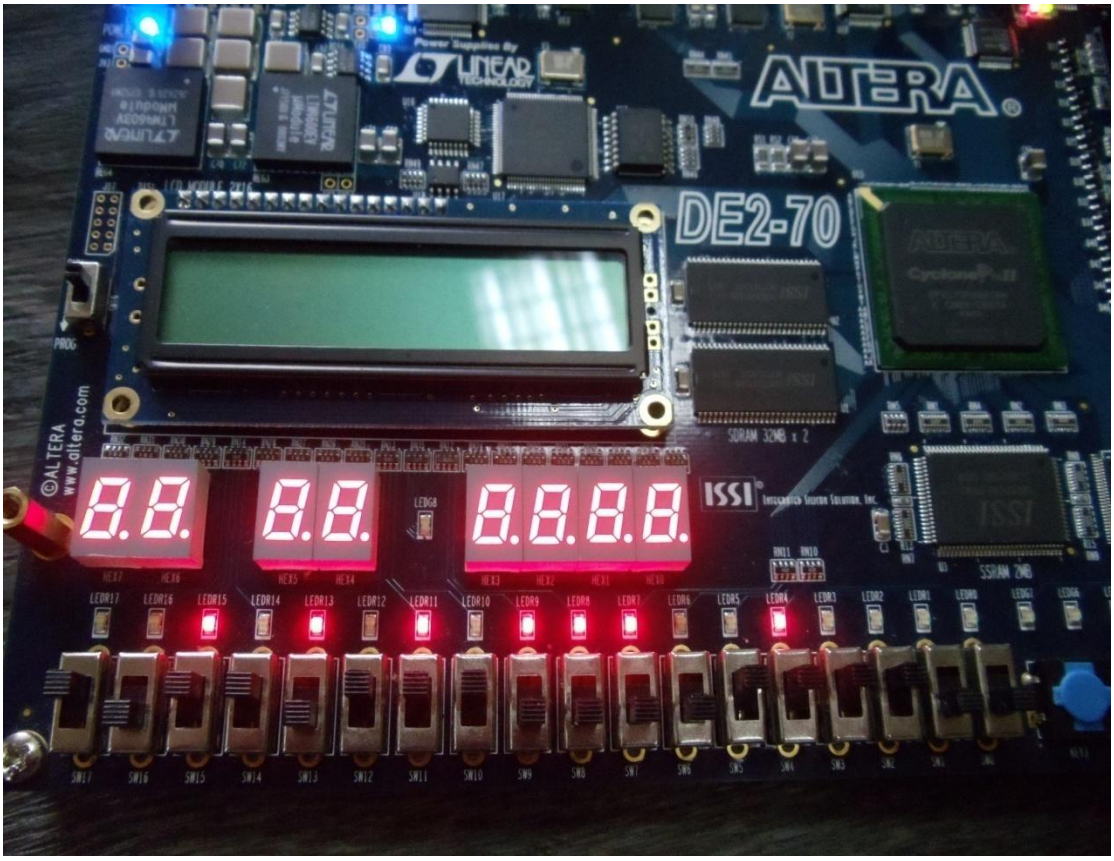
4.4. ÇARPMA ALGORİTMALARININ FPGA SENTEZLEMELERİ

Bu bölümde, Altera firmasının QUARTUS II (Sürüm 10.0 sp1) web edition aracını ve aynı firmanın Cyclone serisinin Cyclone II FPGA kiti modelini kullanılarak çarpma devrelerinin sentezlemeleri yapılmıştır.

Urdhva çarpma yöntemini kullanarak, 4 bitlik iki sayının çarpımı FPGA kiti üzerinde gerçeklenmesi Şekil 4.29'da görülmektedir.Burada görüldüğü gibi girişler (A,B) siyah anahtarlerden (switchs) verilerek, A sayısı kitin sol tarafından ilk 4 düğmeden, B sayısı sağ ilk 4 düğmeden girilerek, çarpmanın sonucu ise sol taraftan ilk 8 ledlerde görülmektedir.Yanan ledler çıkış bitin bir olduğu anlamına gelir.



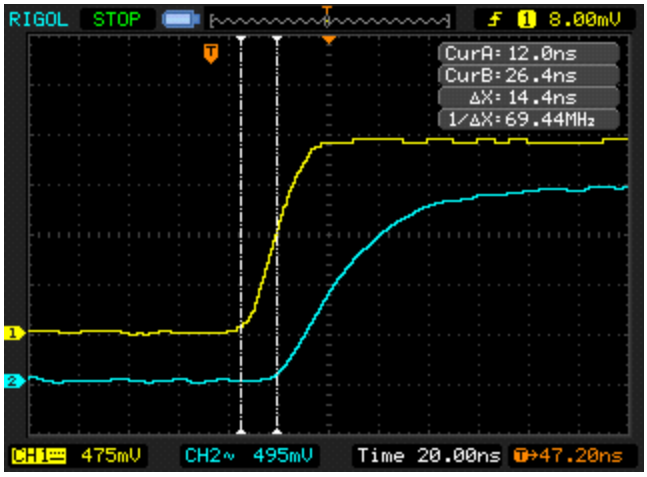
Şekil 4.29 : (4x4 bit) Urdhva çarpımının FPGA sentezlemesi



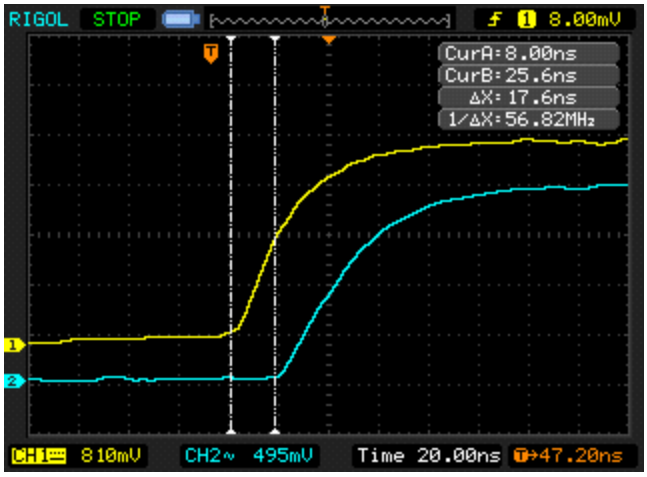
Şekil 4.30 : (8x8 bit) Önerilen çarpımının FPGA sentezlemesi

Şekil 4.30'da görüldüğü gibi, 8 bitlik iki sayının çarpımı Önerilen çarpma yöntemini kullanarak FPGA kiti üzerinde gerçekleştirilmiştir.Fakat girişler 8 bit olduğu için A sayısı kitin sol tarafından ilk 8 düğmeden, B sayısı sağ ilk 8 düğmeden girilerek, çarpmanın sonucu ise sol taraftan ilk 16 ledlerde görülmektedir.

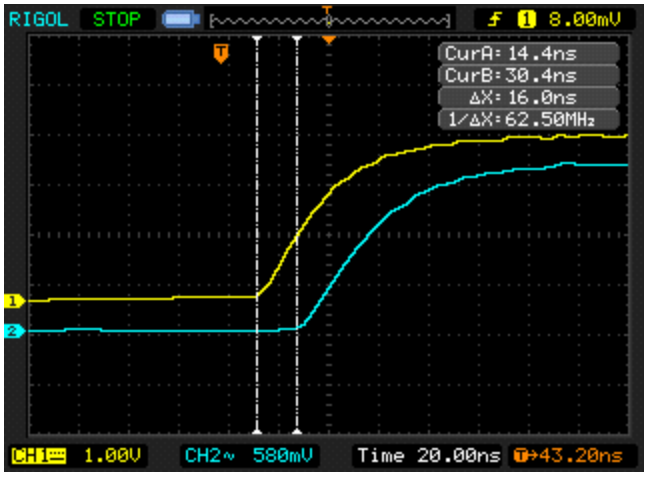
Çarpma devreleri FPGA kiti üzerinde sentezlenerek performansları elde edilmiştir. Performans ölçütü olarak girişten çıkışa en uzun gecikme süresi kriteri esas alınmıştır.Buradaki gecikme FPGA kiti üzerindeki gecikmeyi temsil etmektedir.Şekil 4.31, Şekil 4.32, Şekil 4.33, Şekil 4.34, Şekil 4.35, Şekil 4.36'da (4,8) bitlik çarpma devrelerinin Osilaskop görüntüleri grafik halinde gösterilmektedir. (4,8) bitlik devre performansları sonuçlarına bakıldığında Urdhva çarpma algoritmasının 4 bitlik sayılar için gecikmesi 14.4 ns'dir, 8 bitlik sayılar için gecikmesi ise 17.6 ns'dir.Booth çarpma algoritmasının (4, 8) bitlik sayılar için gecikmesi sırayla (16.0, 19.2) ns'dir.Önerilen çarpma algoritmasının (4, 8) bitlik sayılar için gecikmesi ise (10.4, 11.2) ns'dir.



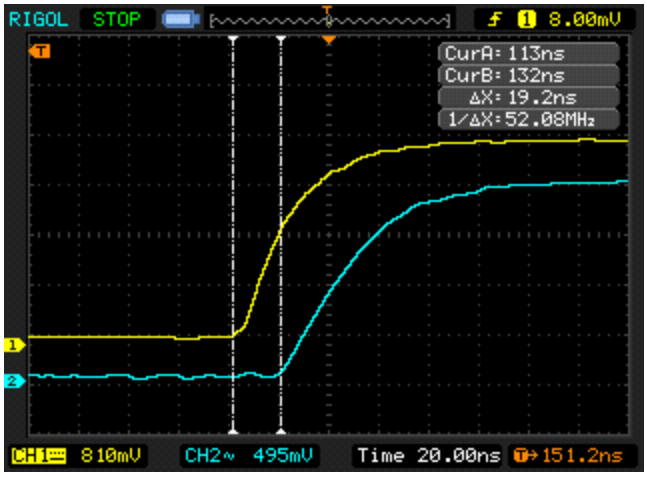
Şekil 4.31 : 4x4 bit Urdhva çarpma devresi giriş ve çıkış işaretleri dalga forumu



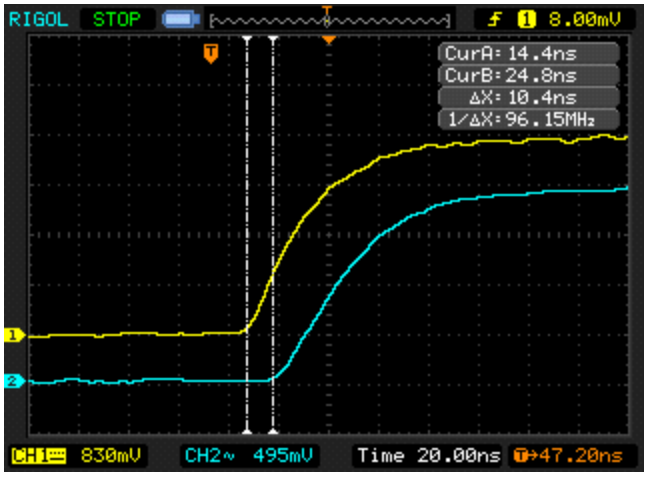
Şekil 4.32 : 8x8 bit Urdhva çarpma devresi giriş ve çıkış işaretleri dalga forumu



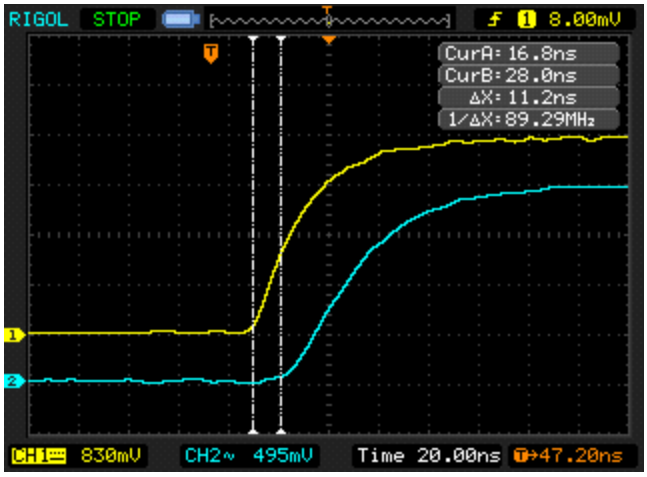
Şekil 4.33 : 4x4 bit Booth çarpma devresi giriş ve çıkış işaretleri dalga forumu



Şekil 4.34 : 8x8 bit Booth çarpma devresi giriş ve çıkış işaretleri dalga forumu



Şekil 4.35 : 4x4 bit Önerilen çarpma devresi giriş ve çıkış işaretleri dalga forumu



Şekil 4.36 : 8x8 bit Önerilen çarpma devresi giriş ve çıkış işaretli dalga forumu

5. TARTIŞMA VE SONUÇ

Bu tezde, etkin bit indirgeme metotları kullanan çarpma algoritmaları ele alınarak, çarpma işlemlerin donanımsal algoritmaları incelenip, VHDL dilinde simülasyonlarını gerçekleştirerek algoritmaların performans analizleri yapılmıştır. Ayrıca, bütün donanımsal çarpma devreleri FPGA kiti yardımıyla sentez edilerek devrelerin performansları belirlenmeye çalışılmıştır.

Tezde incelenen çarpma algoritmalarını gerçekleyen donanımsal devreler üzerinde, analitik irdeleme sonucu gecikme (girişten-çıkışa en uzun yol gecikmesi) ve kullanılan toplam standart kapı sayısını (birim kapı-unit gate) veren fonksiyonlar, giriş operandı bit uzunluğuna bağlı olarak elde edilmiştir. Ayrıca, devrelerin fonksiyonel doğrulanması ve en uzun gecikme süreleri VHDL kullanılarak yapılan simülasyonlarla belirlenmiştir. Ele alınan tüm aritmetik işlemleri gerçekleyen donanımsal devrelerin çip tasarımı halinde sarfedeceği enerjinin ölçüsü olarak, en uzun gecikme süresi ile kullanılan kapı sayılarının çarpımı performans analizinde bir kriter olarak seçilmiştir.

Çarpma devreleri içinde, en hızlı olanı, yani en kötü gecikme süresi en düşük, Önerilen çarpma devresi (Proposed) iken, en yavaş çalışan ise Booth çarpma devresi tespit edilmiştir. Diğer yandan, kapladığı çip alanı veya kullanılan birim kapı sayısı en az olan, yani en az maliyetle üretilebilen Booth devresi iken, en fazla maiyetli olanı Önerilen çarpma devresi olduğu görülmüştür. Toplam performans, yani çipte harcanan enerji miktari gözönüne alınırsa, Önerilen (Proposed) çarpma devresi en iyi; Booth çarpma devresi ise en kötü performansa sahip olduğu belirlenmiştir.Urdhva çarpma devresi ise orta gecikme ve aynı zamanda orta maliyetle bir performans sergilemektedir.

Tablo 5.1: Çarpma Algoritmaları VHDL Alan,Verimlilik ve İşlem Zamanları

Çarpıcı Türü	**Urdhva**			**Booth**			**Önerilen**		
Bit uzunluğu	**4bit**	**8bit**	**16bit**	**4bit**	**8bit**	**16bit**	**4bit**	**8bit**	**16bit**
Gecikme (ns)	11	22	41	13	24	42	7	10	29
Alan	228	634	2621	217	592	2587	260	1164	3116
Verimlilik	2508	13948	107461	2821	14208	108654	1820	11640	90364

Ayrıca, çarpma devrelerinin FPGA kiti üzerindeki performanslarını karşılaştırırken yine Önerilen çarpma devresi en hızlı, yani gecikme süresi en düşük bir algoritma olduğu belirlenmiştir. Özellikle, yüksek bit değerlerinde Önerilen çarpma devresinin gecikmesi, diğer yöntemlerle tasarlanan çarpma devrelerine göre daha hızlı küçüldüğü elde edilen grafiklerle ortaya çıkmıştır.Urdhva devresi ise orta bir gecikme ile yine iki algoritmanın arasında bir performans sergilemektedir. Buna karşılık, en yavaş çarpma devresi ise, Booth algoritmasına dayanan devre olduğu tespit edilmiştir .

Tablo 5.2 : Çarpma Algoritmaları FPGA İşlem Zamanları

Çarpıcı Türü	**Urdhva**		**Booth**		**Önerilen**	
Bit uzunluğu	**4bit**	**8bit**	**4bit**	**8bit**	**4bit**	**8bit**
Gecikme (ns)	14	18	16	19	10	11

Bu çalışmada, çarpma algoritmalarında performans ölçümünde daha tatmin edici verilere ulaşmak için daha büyük bit uzunluklu operandlar ile işlem yapmak gerekmektedir. Çarpma işlemlerini yaparken 4,8,16 bitlik sayılar yerine 32 bitlik, 64 bitlik sayıların kullanılması çok daha sağlıklı sonuçlar vermekle birlikte algoritmaların birbirleri arasında farkı da daha iyi ortaya çıkarır.

Simülasyonlar sırasında kullanmış olduğumuz yazılımların sınırlı kullanıma açık öğrenci versiyonlarından dolayı operand bit uzunlukları küçük tutulmuştur. VHDL profesyonel versiyonu satın alınarak sözkonusu devrelerin analizi daha büyük bit uzunlukları için de yapılması geleceğe dönük bir çalışma olarak önerilebilir.

KAYNAKLAR

1. WIKIPEDIA, 2011, *Multiplication* [online], http://en.wikipedia.org/wiki/Multiplication [Ziyaret Tarihi: 18 Mayıs 2011]

2. SİNGH, AMANDEEP, 2010, *Design And Hardware Realization Of a 16 Bit Vedic Arithmetic Unit*, Thesis (PhD), Thapar University.

3. SHRIPAD KULKARNI, 2007, Discrete Fourier Transform (DFT) by using Vedic Mathematics, *report, vedicmathsindia.blogspot.com*.

4. JAGADGURU SWAMI SRI BHARATI KRISHNA TIRTHJI MAHARAJA, 1986, *Vedic Mathematics*, Motilal Banarsidas,Varanasi, İndia.

5. HIMANSHU THAPLIYAL, SAURABH KOTIYAL and M. B SRINIVAS, 2005, Design and Analysis of A Novel Parallel Square and Cube Architecture Based On Ancient Indian Vedic Mathematics, *Centre for VLSI and Embedded System Technologies, International Institute of Information Technology, Hyderabad, IEEE*, 500019.

6. HONEY DURGA TIWARI, GANZORIG GANKHUYAG, CHAN MO KIM and YONG BEOM CHO, 2008, Multiplier Design Based On Ancient Indian Vedic Mathematics, *International SoC Design Conference*, II-66.

7. HARPREET SINGH DHILLON and ABHIJIT MITRA, 2008, A Reduced- Bit Multiplication Algorithm for Digital Arithmetics, *International Journal of Computational and Mathematical Sciences*, 64 - 69.

8. AKHTER SHAMIM, 2007, Vhdl Implementation Of Fast NxN Multiplier Based On Vedic Mathematic, *IEEE Trans. On Computers*, 472 – 473.

9. SREE N. SREENATH, 2003, The Vedic Math Guy, *Lotus, Indian -Community Monthly Newspaper, Cleveland,* 1 – 2.

10. KUMAR, VİNAY, 2009, *Analysis, Verification And FPGA Implementation Of Vedic Multiplier With Bist Capability,* Thesis (PhD), Thapar University.

11. PERHAMI, BEHROOZ, 2000, *Computer Arithmetic: Algorithms and Hardware Designs*, Oxford University Press.

12. ISRAEL, KOREN, 1993, *Computer Arithmetic Algorithms*, Englewood Cliffs, Prentice Halls.

13. BOOTH, A.D., 1951, A Signed Binary Multiplication Technique, *Quart J. Mech.Appl.Math.*, 4Part.2.

14. RUBINFIELD, L.P., 1975, A Proof of The Modified Booth's Algorithm for Multiplication, *IEEE Trans. On Computers*, C-24, pp. 1014-1015

15. WALLACE, C. S., 1964 , A Suggestion For Fast Multiplier, *IEEE Trans. Electronic Computer*, Vol. 13

16. ÖZBEY, RECEP SELAMİ. , 2004, *Bilgisayar Aritmetik Ünitelerinin Tasarımı İçin VHDL Tabanlı Kütüphane Geliştirilmesi,* Yüksek Lisans, İstanbul Üniversitesi

17. HANSELMAN D., LITTLEFIELD B., 1995, *MASTERING MATLAB*, Prentice – Hall Int, London, Inc.

18. UZUNOĞLU M., KIZIL A., ONAR Ö. Ç., 2002, *Kolay Anlatımı İle MATLAB 6.0- 6.5*, Türkmen Yayınevi, Türkiye.

19. ALTERA CORPERATION, *MAX Plus II Data Sheets* [online], http://www.altera.com [Ziyaret Tarihi: 9 Mayıs 2011]

20. WIKIPEDIA, 2011, *Field-programmable_gate_array* [online], http://en.wikipedia.org/wiki/Field-programmable_gate_array [Ziyaret Tarihi: 27 Mayıs 2011]

21. NASRI SULAIMAN, ZEYAD ASSI OBAID, M. H. MARHABAN AND M. N. HAMIDON, 2009, Design and Implementation of FPGA-Based Systems - A Review, *Australian Journal of Basic and Applied Sciences*, 3(4), 3575-3596.

EKLER

Bu çalışmada ek olarak tez ve tez içinde MATLAB ve VHDL simülasyonları yapılmış olan aritmetik çarpma algoritmaların tüm kaynak kodları, simülasyon sonuç formları bir cd içerisinde sunulmaktadır. Bunlara ek olarak programların yazılmış olduğu ücretsiz yazılım olan MAX-Plus ve kullanılan programlama dili olan VHDL ile ilgili dokumanlar da yer almaktadır. Ayrıca, tez içerisinde kullandığımız FPGA kiti ile tüm aritmetik çarpma algoritmaların simülasyon sonuç görüntüleri ve ilgili dokumanlar da yer almaktadır.

ÖZGEÇMİŞ

Sarmad MOHAMMED Ocak 1987'de IRAK Kerkük ili Musalla semtinde doğdu.İLk-orta-lise eğitimini aynı şehirde tamamladı. Lisans eğitimini 2004-2008 yılları arasında Kerkük Teknik Üniversitesi'nde tamamladı. 2008–2009 yılında Ankara'da Gazi Üniversitesi TÖMER (Türkçe Öğrenim, Araştırma ve Uygulama Merkezinde) düzenlenen Türkiye Türkçesi programını başarıyla tamamladı.Eylül 2009'da İstanbul Üniversitesi, Mühendislik Fakültesi, Bilgisayar Mühendisliği Ana Bilim Dalı'nda Yüksek Lisans eğitimine başladı.