T.C.
İSTANBUL ÜNİVERSİTESİ-CERRAHPAŞA
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

YÜKSEK LİSANS TEZİ

FPGA TABANLI 64 BİT ARİTMETİK MANTIK BİRİMİ TASARIMI

Nuray SAĞLAM BEDİR

DANIŞMAN
Prof. Dr. Fırat KAÇAR

Elektrik-Elektronik Mühendisliği Anabilim Dalı

Elektrik-Elektronik Mühendisliği Programı

İSTANBUL-2018

Bu çalışma 14.12.2018 Tarihinde aşağıdaki jüri tarafından Elektrik-Elektronik Mühendisliği Anabilim Dalı, Elektrik-Elektronik Mühendisliği Programı Yüksek Lisans Tezi olarak kabul edilmiştir.

TEZ JÜRİSİ

Prof. Dr. Fırat KAÇAR
İstanbul Üniversitesi-Cerrahpaşa
Mühendislik Fakültesi

Doç. Dr. Yasin ÖZÇELEP
İstanbul Üniversitesi-Cerrahpaşa
Mühendislik Fakültesi

Dr.Öğr.Üyesi Muhammed Emin BAŞAK
Yıldız Teknik Üniversitesi
Gemi İnşaatı ve Denizcilik Fakültesi

20.04.2016 tarihli Resmi Gazete'de yayımlanan Lisansüstü Eğitim ve Öğretim Yönetmeliğinin 9/2 ve 22/2 maddeleri gereğince; Bu Lisansüstü teze, İstanbul Üniversitesi-Cerrahpaşa'nın abonesi olduğu intihal yazılım programı kullanılarak Lisansüstü Eğitim Enstitüsü'nün belirlemiş olduğu ölçütlere uygun rapor alınmıştır.

ÖNSÖZ

Dijital elektronik sektöründeki yüksek teknolojik gelişimeler hızlı ürün geliştirilmesi ve prototipleşmesini beraberinde getirmiştir. Dijital elektronik dünyasında hızlı ve esnek ürün geliştirmek ve ihtiyaçlara yönelik çözümler sunmak için pek çok araç geliştirilmiştir. Geliştirilen bu araçların en önemlilerinden biri FPGA'lardır. Dijital olarak yapılabilen tüm tasarımların tek bir entegre üzerinde geliştirilebilmesine olanak sağlaması FPGA teknolojisinin önemini artırmaktadır.

Günümüzde dijital sistemler neredeyse tüm özel ve profesyonel faaliyetlerimizde kullanılmaktadır. Her dijital sistemin bünyesinde sistemin beynini oluşturan merkezi bir işlem birim(CPU) bulunur. CPU'da yer alan veri işleme birimi Aritmetik Mantık Birimi(ALU) olarak adlandırılır ve bu birim CPU'nun kalbidir.

Bu çalışma CPU'nun kalbi olan ALU'nun FPGA teknolojisi kullanılarak gerçeklenmesini ele almaktadır.

Yapılan çalışmanın hayata geçirilmesi ve Yüksek Lisans eğitimim süresince gösterdiği her türlü destek ve yardımlarından dolayı çok değerli hocam ve danışmanım Prof. Dr. Fırat Kaçar'a teşekkürü bir borç bilirim.

Ayrıca, hayatımın her anında olduğu gibi Yüksek Lisans eğitimim süresince de desteğini bir an olsun eksik etmeyen ve her daim yanımda duran sevgili eşime ve aileme çok teşekkür ederim.

Aralık 2018 Nuray SAĞLAM BEDİR

İÇİNDEKİLER

Sayfa No

ÖNSÖZ ..... **iv**
İÇİNDEKİLER ..... **v**
ŞEKİL LİSTESİ ..... **viii**
TABLO LİSTESİ ..... **xiii**
SİMGE VE KISALTMA LİSTESİ ..... **xiv**
ÖZET ..... **xvii**
SUMMARY ..... **xviii**
1. GİRİŞ **1**
2. GENEL KISIMLAR **3**
2.1 MİKROİŞLEMCİ 3
2.1.1. Kısa Tarihçe 3
2.1.2. Mikroişlemci Yapısı ve Çalışma Şekli 4
2.1.2.1. Kontrol Birimi *6*
2.1.2.2. Kaydediciler (Registers) *6*
2.1.2.3. Aritmetik Mantık Birimi *8*
2.1.3. Mikroişlemci Performansı 8
2.1.3.1. Bilgisayar Çalışma Hızı (Clock Speed) *9*
2.1.3.2. Sözcük Boyutu (Word Size) *9*
2.1.3.3. Yol Genişliği (Bus Width) *10*
2.1.3.4. İş Hattı (Pipelining) *10*
2.1.3.5. RISC(Reduced Instruction Set Computer) İşlem *11*
2.1.3.6. Çoklu İşlem(Multiprocessing) *11*
2.1.3.7. Çok çekirdeklilik (Multicore) *11*
2.1.4. Bellek Hiyerarşisi 13
2.1.4.1. RAM (Random Access Memory) *14*
2.1.4.2. ROM (Read-Only Memory) *17*
2.2. ALANDA PROGRAMLANABİLİR KAPI DİZGELERİ (FPGA) 21
2.2.1. Programlanabilir Mantık Cihazlarının Gelişimi 21
2.2.1.1. SPLD(Simple Programmable Logic Devices) *22*
2.2.1.2. CPLD (Complex Programmable Logic Devices) *25*

2.2.2. ASIC(Application Specific Integrated Circuits) 27
2.2.2.1. Full Custom *28*
2.2.2.2. Standard Cell *28*
2.2.2.3. Gate Array *29*
2.2.2.4. Structured ASIC *29*
2.2.3. FPGA (Field Programmable Gate Arrays) 30
2.2.3.1. FPGA Mimarisi *31*
2.2.3.3. Giriş/Çıkış Blokları *35*
2.2.3.4. Alıcı/Verici Blokları *36*
2.2.3.5. Programlanabilir Ara Bağlantılar *36*
2.2.3.6. RAM Blokları *37*
2.2.3.7. FPGA Pinleri *37*
2.2.3.8. Global İç Bağlantılar ve Saat Pinleri *38*
2.2.4. FPGA Programlama Teknolojileri 38
2.2.4.1. SRAM Tabanlı Programlama Teknolojisi *39*
2.2.4.2. SRAM/Flash Tabanlı Programlama Teknolojisi *40*
2.2.4.3. Anti-fuse Tabanlı Programlama Teknolojisi *40*
2.2.4.4. EPROM ve EEPROM Tabanlı Programlama Teknolojisi *41*
2.2.4.5. Flash Tabanlı Programlama Teknolojisi *41*
3. MALZEME VE YÖNTEM 42
3.1. ALTERA CYCLONE® II MİMARİSİ 42
3.1.1. Lojik Element Yapıları 43
3.1.2. Lojik Dizi Blokları(LAB) 46
3.1.3. PLL(Phase Lock Loop) Yapıları 50
3.1.4. Gömülü Bellekler 51
3.1.5. Gömülü Çarpıcılar 52
3.1.6. I/O Yapısı ve Özellikleri 54
3.1.7. Cyclone II – EP2C70F896C6 FPGA Yapısı 56
3.1.8. Cyclone II FPGA Konfigürasyonu 56
3.1.9. DE2-70 Geliştirme Kartı 58
3.2. FPGA PROGRAMLAMA DİLLERİ 59
3.2.1. Verilog HDL 59
3.2.2.VHDL (Very High-Speed Integrated Circuit Hardware Description Language) 60
3.2.2.1. Tasarım Birimleri *60*

2.3.2.2. İfade Türleri *66*
3.2.2.3. Veri Nesneleri *67*
3.2.2.4. Veri Türleri *68*
3.2.2.5. Operatörler *70*
3.2.2.6.Önemli Yapılar *71*
4. BULGULAR **80**
4.1. 64 BİT MANTIK BİRİMİ TASARIMI 81
4.1.1. 64 Bit Mantık Birimi Bulguları 83
4.2. 64 BİT ARİTMETİK BİRİM TASARIMI 84
4.2.1. 64 Bit İşaretli Toplayıcı Algoritması 84
4.2.2. 64 Bit İşaretli Çıkarıcı Algoritması 93
4.2.3. 64 Bit İşaretli Çarpıcı Algoritması 96
4.2.4. 64 Bit İşaretli Bölücü Algoritması 102
4.2.5. 64 Bit İşaretli Mod Algoritması 106
4.2.6. 64 Bit Aritmetik Birim Bulguları 109
4.3. 64 BİT KAYDIRMA-DÖNDÜRME BİRİMİ TASARIMI 110
4.3.1. Aritmetik Sağa Kaydırma 110
4.3.2. Aritmetik Sola Kaydırma 112
4.3.3. Lojik Sağa Kaydırma 112
4.3.4. Lojik Sola Kaydırma 113
4.3.5. Sağa Döndürme 113
4.3.6. Sola Döndürme 114
4.3.7. 64 Bit Kaydırma-Döndürme Birimi Bulguları 114
4.4. 64 BİT ARİTMETİK MANTIK BİRİMİ 115
5. TARTIŞMA VE SONUÇ **118**
KAYNAKLAR **121**
ÖZGEÇMİŞ **123**

ŞEKİL LİSTESİ

Sayfa No

Şekil 1.1: Genel Bir Sistemin Blok Diyagramı 2

Şekil 2.1: Merkezi işlem biriminin diğer bilimler ile ilişkisi 5

Şekil 2.2: Kontrol Birimi Fonksiyonları 6

Şekil 2.3: ALU blok diyagramı 8

Şekil 2.4: Pipelining'siz Makine Döngüsü 10

Şekil 2.5: Pipelining'li Makine Dögüsü 11

Şekil 2.6: Tek Çekirdekli CPU yapısı 12

Şekil 2.7: Çok Çekirdekli CPU yapısı 12

Şekil 2.8: Bellek Hiyerarşisi 13

Şekil 2.9: a) 1 Bitlik SRAM hücresi, b) Bellek hücresinin CMOS devresi 15

Şekil 2.10: 64 Bit SRAM 15

Şekil 2.11: 4 Transistörlü DRAM hücresi 16

Şekil 2.12: Ferroelektrik kristal yapısı 16

Şekil 2.13: NAND tabanlı ROM yapısı 18

Şekil 2.14: NOR tabanlı ROM yapısı 19

Şekil 2.15: PROM yapısı 20

Şekil 2.16: a) Geleneksel MOS b) Floating gate MOS 20

Şekil 2.17: EEPROM Hücresi (FLOTOX) 21

Şekil 2.18: Programlanabilir Mantık Gelişimi 22

Şekil 2.19: 3 girişli, 3 çıkışlı önceden programlanmış PROM yapısı 23

Şekil 2.20: Sigorta bağlantılı teknoloji 23

Şekil 2.21: 3 giriş, 3 çıkışlı programlanmamış PLA yapısı 24

Şekil 2.22: 3 giriş, 3 çıkışlı programlanmamış PAL yapısı 24

Şekil 2.23: Sırası ile PROM, PAL ve PLA yapısı 25

Şekil 2.24: a) Basit PAL yapısı(SPLD) b) Karmaşık PLD yapısı(CPLD).....26

Şekil 2.25: CPLD iç bağlantıları ve LAB'lar.....26

Şekil 2.26: a) MAX 7000 CPLD içyapısı, b) MAX 7000 macrocell yapısı 27

Şekil 2.27: PLDs ve ASICs.....28

Şekil 2.28: Gate Array yaklaşımlı yonga görünümü.....29

Şekil 2.29: Basit gate array temel hücreleri 29

Şekil 2.30: Yapısal ASIC temelleri.....30

Şekil 2.31: Yapısal ASIC genel görünümü.....30

Şekil 2.32: Modern FPGA yapısı.....32

Şekil 2.33: FPGA Temel Lojik Elementi ve Kümesi.....33

Şekil 2.34: LUT giriş sayısına göre alan, routing alanı, fpga alanı ve gecikme değişimi.....34

Şekil 2.35: 3-LUT ve 4-LUT yapısı.....35

Şekil 2.36: Programlanabilir ara bağlantılar 36

Şekil 2.37: Sırası ile programlama öncesi ve sonrası Anti-Fuse yapısı 40

Şekil 3.1: Cyclone II mimarili EP2C20 etegresi blok diyagramı.....42

Şekil 3.2: Cyclone II Lojik Element Yapısı 44

Şekil 3.3: Normal modda LE yapısı.....46

Şekil 3.4: Aritmetik modda LE yapısı.....46

Şekil 3.5: Cyclone II mimarisinde LAB yapısı.....47

Şekil 3.6: Direkt link ve register zinciri bağlantısı 48

Şekil 3.7: R4 satır ara bağlantıları.....49

Şekil 3.8: Register zinciri ve C4 ara bağlantısı 50

Şekil 3.9: PLL yapılarının FPGA içerisindeki konumları.....51

Şekil 3.10: M4K Bellek Bloğu LAB satır arayüzü 52

Şekil 3.11: Çarpıcı Blok mimarisi.....53

Şekil 3.12: Çarpıcıları Yapılarının Routing Arayüzü 54

Şekil 3.13: IO elementi yapısı.....55
Şekil 3.14: I/O Bloklarının Arabağlantılara Bağlantısı.....55
Şekil 3.15: AS Programlama Metodu 57
Şekil 3.16: JTAG Programlama Metodu.....57
Şekil 3.17: DE2-70 Geliştirme Kartı.....59
Şekil 3.18: Bildirim türlerinin birbiri ile ilişkisi 61
Şekil 3.19: Veri Türleri 69
Şekil 4.1: VHDL tabanlı tasarım akışı 80
Şekil 4.2: 64 Bit VE Bileşeni Derleme Raporu 82
Şekil 4.3: 64 Bit VE Bileşeni RTL Şeması 82
Şekil 4.4: 64 Bit VE Bileşeni Benzetim Sonucu.....82
Şekil 4.5: 64 Bit Mantık Birimi Derleme Raporu.....83
Şekil 4.6: 64 Bit Mantık Birimi RTL Şeması 83
Şekil 4.7: 64 Bit Mantık Birimi Benzetim sonucu.....84
Şekil 4.8: Klasik toplayıcıların gecikmesi ve kapladığı alan karşılaştırması.....85
Şekil 4.9: Full_adder bileşeni RTL yapısı.....86
Şekil 4.10: 2 bitlik COSA RTL şeması.....87
Şekil 4.11: 4:2 MUX yapısı 87
Şekil 4.12: 2 Bitlik COSA toplayıcı benzetim sonucu.....88
Şekil 4.13: 64 Bit İşaretsiz Toplayıcı Bileşeni Derleme Raporu 89
Şekil 4.14: 64 Bitlik Toplayıcı Bileşeni RTL Şeması.....90
Şekil 4.15: 64 Bitlik Toplayıcı Bileşeni Benzetim Sonucu.....90
Şekil 4.16: 64 Bit İşaretli Toplayıcı Bileşeni Derleme Raporu 92
Şekil 4.17: İşaretli Sayılar için 64 Bit Toplayıcı Bileşeni RTL Şeması.....93
Şekil 4.18: İşaretli Sayılar için 64 Bit Toplayıcı Bileşeni Benzetim Sonucu 93
Şekil 4.19: 64 Bit Çıkarıcı Bileşeni RTL Şeması 94
Şekil 4.20: 64 Bit Çıkarıcı Bileşeni Benzetim Sonucu 95

Şekil 4.21: Tasarımın Derleme Raporu.....95
Şekil 4.22: 64 İşaretli Sayılar için 64 Bit Çıkarıcı Bileşeni RTL Şeması.....95
Şekil 4.23: 64 İşaretli Sayılar için 64 Bit Çıkarıcı Bileşeni Benzetim Sonucu.....95
Şekil 4.24: Booth's Çarpma Algoritması Akışı.....96
Şekil 4.25: 64 Bit Çarpıcı Bileşeni RTL Şeması.....101
Şekil 4.26: Tasarımın Derleme Raporu.....101
Şekil 4.27: 64 Bit Çarpıcı Bileşeni Benzetim Sonucu.....101
Şekil 4.28: İşaretsiz Bölücü Bileşeni Algoritma Akışı.....102
Şekil 4.29: 64 Bit İşaretsiz Bölücü Bileşeni Derleme Raporu.....103
Şekil 4.30: 64 Bit İşaretsiz Bölücü Bileşeni RTL Şeması.....103
Şekil 4.31: 64 Bit İşaretsiz Bölücü Bileşeni Benzetim Sonucu.....103
Şekil 4.32: 64 Bit İşaretli Bölücü Bileşeni Derleme Raporu.....105
Şekil 4.33: 64 Bit İşaretli Bölücü Bileşeni RTL Şeması.....105
Şekil 4.34: 64 Bit işaretli Bölücü Bileşeni Benzetim Sonucu.....105
Şekil 4.35: 64 Bit İşaretsiz Mod Alıcı Bileşen Derleme Raporu.....106
Şekil 4.36: 64 Bit İşaretsiz Mod Alıcı Bileşen Benzetim Sonucu.....106
Şekil 4.37: 64 Bit İşaretli Mod Alma Bileşeni Derleme Raporu.....108
Şekil 4.38: 64 Bit İşaretli Mod Alma Bileşeni RTL Şeması.....108
Şekil 4.39: 64 Bit İşaretli Mod Alma Bileşeni Benzetim Sonucu.....108
Şekil 4.40: 64 Bit Aritmetik Birim Derleme Raporu.....109
Şekil 4.41: 64 Bit Aritmetik Birim RTL Şeması.....109
Şekil 4.42: 64 Bit Aritmetik Birim Benzetim Sonucu.....110
Şekil 4.43: Aritmetik Kaydırma Bileşeni Derleme Raporu.....111
Şekil 4.44: Aritmetik Sağa Kaydırma Bileşeni Benzetim sonucu.....111
Şekil 4.45: Aritmetik Sola Kaydırma Bileşeni Benzetim sonucu.....112
Şekil 4.46: Lojik Sağa Kaydırma Bileşeni Derleme Raporu.....112
Şekil 4.47: Lojik Sağa Kaydırma Bileşeni Benzetim Sonucu.....112

Şekil 4.48: Lojik Sola Kaydırma Bileşeni Benzetim Sonucu 113

Şekil 4.49: Sağa Döndürme Bileşeni Derleme Raporu 113

Şekil 4.50: Sağa Döndürme Bileşeni RTL şeması 114

Şekil 4.51: Sağa Döndürme Bileşeni Benzetim Sonucu 114

Şekil 4.52: Sola Döndürme Bileşeni Benzetim Sonucu 114

Şekil 4.53: 64 Bit Kaydırma Döndürme Birimi Derleme Raporu 115

Şekil 4.54: 64 Bir Kaydırma ve Döndürme Birimi RTL Şeması 115

Şekil 4.55: 64 Bit Kaydırma ve Döndürme Birimi Benzetim Sonucu 115

Şekil 4.56: 64 Bit Aritmetik Mantık Birimi Derleme Raporu 117

Şekil 4.57: 64 Bit Aritmetik Mantık Birimi RTL Şeması 117

Şekil 4.58: 64 Bit Aritmetik Mantık Birimi Benzetim Sonuçları 117

TABLO LİSTESİ

Sayfa No

Tablo 2.1: 2000'li yıllara kadar piyasaya sürülen Intel işlemciler ve özellikleri 3

Tablo 2.2: İşlemci Derecelendirme Örneği 9

Tablo 2.3: RAM teknolojileri karşılaştırma tablosu 17

Tablo 2.4: Basit PLD'ler ve Karmaşık PLD'lerin karşılaştırılması 27

Tablo 2.5: FPGA programlama teknolojileri karşılaştırması 39

Tablo 3.1: Çarpıcılarda *Sign* gösterimi 54

Tablo 3.2: Operatörler 70

Tablo 4.1: 64 Bit ALU Talimat ve İşlem Tablosu 116

SİMGE VE KISALTMA LİSTESİ

Simgeler **Açıklama**

SiO_2 **:** Silisyum dioksit

Kısaltmalar **Açıklama**

ALU : Arithmetic Logic Unit
API : Appliation Programming Interface
AS : Active Serial
ASIC : Aplication Sipesific Integrated Circuits
CLA : Carry Lookahead Adder
CLB : Configurable Logic Block
COSA : Conditional Sum Adder
CMOS : Complementary Metal Oxide Semiconductor
CPLD : Complex PLD
CPU : Central Processing Unit
CRAM : Configurable RAM
CSKA : Carry Skip Adder
CSLA : Carry Select Adder
DCT : Discrete Cosine Transform
DRAM : Dynamic Random Access Memory
DSP : Digital Signal Processing
EPCS : Electronic Prescriptions for Controlled Substances
EEPROM : Electrically Erasable Programmable Read Only Memory
EPROM : Erasable Programmable Read Only Memory
FA : Full Adder
FeRAM : Ferroelectric Random Access Memory
FFT : Fast Fourier Transform
FIFO : Firts In First Out
FIR : Finite Impuls Respond
FPGA : Field Programmable Gate Arrays
GAL : Generic Array Logic

GB : Gigabyte
GHz : Gigahertz
HDL : Hardware Discription Language
IEEE : Institute of Electrical and Electronics Engineers
IOE : Input Output Element
JTAG : Joint Test Action Group
KB : Kilobyte
KHz : Kilohertz
LAB : Logic Array Block
LE : Lojik Element
LPM : Library of Parameterized Modules
LSB : Less Significant Bit
LUT : Look-Up Table
MHz : Megahertz
MİB : Merkezi İşlem Birimi
ms : Milisaniye
MSB : Most Significant Bit
MUX : Multiplexer
NMOS : N kanal CMOS
nJ : Nanojoule
ns : Nanosaniye
VHDL : Very High Speed Integrated Circuit Hardware Description Language
RISC : Reduced Insturaction Set Computer
RAM : Random Access Memory
ROM : Read Only Memory
SRAM : Static Random Access Memory
PAL : Programmable Array Logic
PCI : Peripheral Component Interconnect
PIA : Programmable Interconnect Array
PLA : Programmable Logic Array
PLD : Programmable Logic Device
PLL : Phase Lock Loop
PMOS : P kanal CMOS
PROM : Programmable Read Only Memory

RCA : Ripple Carry Adder

RTL : Register Transfer Level

SPLD : Simple PLD

SSRAM : Synchronous Static Random Access Memory

VCO : Voltage Controlled Oscillator

ÖZET

FPGA TABANLI 64 BİT ARİTMETİK MANTIK BİRİMİ TASARIMI

YÜKSEK LİSANS TEZİ

Nuray SAĞLAM BEDİR

İstanbul Üniversitesi-Cerrahpaşa

Lisansüstü Eğitim Enstitüsü

Elektrik-Elektronik Mühendisliği Anabilim Dalı

Danışman : Prof. Dr. Fırat KAÇAR

II. Danışman : Unvan Ad SOYAD

Günümüz teknolojisi gerçek zamanlı gömülü sistemler için uygun veri iletişim arayüzleri geliştirilmesine büyük ihtiyaç duymaktadır. Alanda programlanabilir Kapı Dizileri (FPGA) programlanabilen çeşitli kaynakları ile etkin gömülü sistem tasarımlarının gerçekleştirilmesini büyük ölçüde desteklemektedir. Programlanmaları donanım tanımlama dilleri ile gerçekleştirilmektedir.

Bu çalışmada Merkezi İşlem Birimi(CPU)'nin en önemli yapısı olan, çeşitli aritmetiksel ve mantıksal işlemlerin gerçekleştirildiği, işaretli sayıların işlenmesini sağlayan 64 Bit İşletimli Aritmetik Mantık Birimi, VHDL donanım tanımlama dili ve Cyclone II FPGA mimarisi kullanılarak tasarlanmıştır. Tasarım yöntemi olarak "buttom-up" benimsenmiş ve modelleme yönteminde davranışsal ve yapısal modelleme yöntemleri kullanılmıştır. Tasarım sonucunda teorik bilgi ve uygulama sonuçları karşılaştırılmış ve bulgular değerlendirilmiştir.

Aralık 2018, 141 sayfa.

Anahtar kelimeler: FPGA, VHDL, Aritmetik Mantık Birimi Tasarımı, Cyclone II Mimarisi

SUMMARY

DESIGN OF FPGA BASED 64 BIT ARITHMETIC LOGIC UNIT

M.Sc. THESIS

Nuray SAĞLAM BEDİR

Istanbul University-Cerrahpasa

Institute of Graduate Studies

Department of Electrical and Electronic Engineering

Supervisor : Prof. Dr. Fırat KAÇAR

Co-Supervisor : Academic Title Name SURNAME

In the present day technology, there is an immense need of developing suitable data communication interfaces for real time embedded systems. Field Programmable Gate Array(FPGA) offers various resources, which can be programmed for building up an efficient embedded system. FPGA programming is performed with the usage of hardware description languages. In this study, the 64-bit Arithmetic Logic Unit, which is the most important structure of the Central Processing Unit (CPU), has been designed for processing signed numbers by using the VHDL hardware description language and Cyclone II FPGA architecture. As a design method, buttom-up method and behavioral and structural modeling methods are used as modeling method. At the end of the study, theoretical knowledge and application results were compared and the findings were evaluated.

December 2018, 141 pages.

Keywords: FPGA, VHDL, Arithmetic Logic Unit Design, Cyclone II FPGA Architecture

1. GİRİŞ

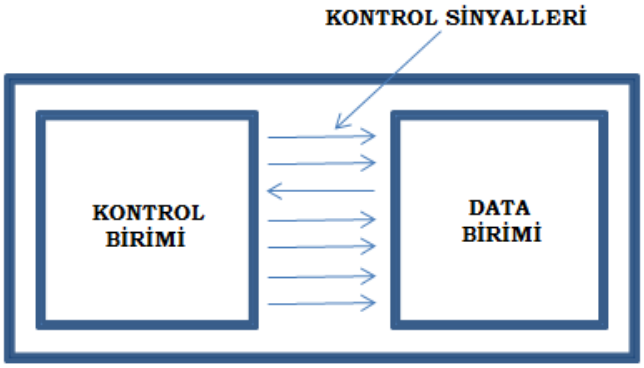
Günümüzde dizüstü bilgisayarlar, tablet, bilgisayarlar gibi elektronik aletlere olan talebin artması yüksek hızlı mikroişlemci tasarımını zorunlu hale getirmektedir. Herhangi bir mikroişlemcinin hızı, merkezi işlem birimi(CPU)'nin ana birimi olan Aritmetik Mantık Birimi(ALU) içindeki herhangi bir görevin yerine getirilme süresi ile doğrudan ilişkilidir. ALU içerisinde toplayıcı ve çarpıcılar bu süreyi doğrudan etkiler. [1-3-5-7]

Aritmetik Mantık Birimi işlemcinin ana birimidir. ALU, veri üzerindeki aritmetiksel veya mantıksal işlemlerin gerçekleştirilmesinde hayati rol oynar. Bilgisayar sistemindeki, register, memory, kontrol birimi, giriş/çıkış birimi gibi diğer birimler temel olarak ALU birimine veri getirmek ve ALU biriminde işlenen veriyi geri almakla yükümlüdür. ALU içerisinde toplama, çıkarma, çarpma bölme gibi temel aritmetik işlemler, and, or, xor, not gibi temel mantıksal işlemler ve sağa-sola kaydırma, sağa-sola döndürme gibi temel kaydırma işlemleri gerçekleştirilir. [1-3-5-7]

Mikroişlemcilerde performans kıstası olarak alınan diğer konulardan ikisi ise güç tüketimi ve tasarım alanıdır. Teknolojinin her geçen gün gelişmesi beraberinde karmaşık sistemleri getirmiştir. Elektronik cihazlara eklenen her yeni özellik sistemin karmaşıklaşmasına ve giderek büyümesine sebep olmuştur. Büyüyen ve karmaşıklaşan sistem yüksek güç tüketimine neden olacaktır. Bahsedilen büyüme ile birlikte de sistemin kapladığı alanda artama ortaya çıkacaktır. Bunlar bir mikroişlemci yapısı için istenmeyen durumlardır.

İşlemcilerin geliştirilmesine yönelik yapılan akademik araştırmalar; tasarımın kapladığı alanın küçültülmesi, düşük güç tüketimi ve işlem hızını geliştirmeye yöneliktir.

Çoğu mühendislik tasarımı faydalı bir işlemi yerine getirmek için bir araya getirilmiş çeşitli elemanlardan oluşan sistem olarak görülebilir. Bu elemanlar arasındaki etkileşim basit ya da karışık olabilir. Sistemlerin çoğu ya tamamıyla elektroniktir ya da izleme ve kontrol amaçlı elektronik alt sistemler içermektedir. Dolayısı ile herhangi bir sistem veri birimi ve kontrol birimine ayrılabilir. Veri birimi veri öğeleri üzerinde toplama, çıkarma, çarpma gibi işlemler yapabilir. Kontrol birimi ise kontrol sinyalleri kullanarak data biriminde gerçekleşen işlemleri kontrol eder. [1-3-5-7]



Şekil 1.1: Genel Bir Sistemin Blok Diyagramı

Data ve kontrol birimlerinin tamamı sadece yazılım kullanarak, sadece donanım kullanarak ya da Yazılım/Donanım birleşik tasarım yöntemi kullanılarak tasarlanabilir. Tasarım yönteminin seçimi alan, hız, güç, maliyet gibi çeşitli kısıtlamalara göre yapılabilir. Yazılım tasarımı ile daha az alanı işgal eden ve düşük hızda çalışan sistemler tasarlanabilir. Yüksek hızda çalışan bir sistemin tasarımı için zamanlama(timing) sorunlarının dikkate alınması gerekmektedir. Yüksek hızlı sistemler için donanım tasarım yöntemi bir çözümdür fakat sistem yazılım tasarım yöntemine kıyasla daha fazla alan işgal edecektir. [1-3-5-7]

Bu çalışma ile CPU'nun data birimi olan Aritmetik Mantık Birimi donanım ve yazılım birleşik tasarım yöntemi kullanılarak Alanda Programlanabilir Kapı Dizgileri(FPGA) ile tasarlanacaktır. Sistem davranışı VHDL donanım tanımlama dili kullanılarak ifade edilecektir.

Çalışmanın temel hedefi, 19 ayrı işlem yapabilen 64 Bit Aritmetik Mantık Birimini VHDL donanım tanımlama dili ile davranışsal olarak tasarlamak ve FPGA kullanarak gerçeklemektir. Tasarım şimdiye kadar yapılan çoğu çalışmadan farklı olarak işaretli sayıların işlenmesini sağlayacak ve tasarımın toplayıcı devresinde elde dalgalı toplayıcı yerine koşullu toplayıcı yapısı kullanılacaktır.

2. GENEL KISIMLAR

2.1 MİKROİŞLEMCİ

2.1.1. Kısa Tarihçe

İlk tek yongalı işlemci 1971 yılında Intel tarafından üretilmiştir. Intel 4004 işlemcisi 4-bitlik işlem yapma kapasitesine sahipti. Çalışma frekansı 740KHz ve program hafızası 4KB idi. İşlemcilerin elektronik cihazlarda kullanılması ile günümüzün elektronik cihazları olan masaüstü bilgisayarlar, dizüstü bilgisayarlar, akıllı telefonlar, tabletler, mp3 çalarlar, çamaşır makineleri, bulaşık makineleri, LCD televizyonlar gibi akıllı sistemler geliştirildi.[1-13]

Geçmişten günümüze doğru gelindiğinde sırası ile 1971 yılında Intel 4004, 1974 yılında Intel 8080, 1979 yılında Intel 8088, 1982 yılında Intel 80286, 1985 yılında Intel 80386, 1989 yılında Intel 80486, 1993 yılında Intel Pentium, 1997 yılında Intel Pentium II, 1999 yılında Intel Pentium III, 2000 yılında Intel Pentium IV piyasaya sürülmüştür. Intel 2000 yılından sonra sırasıyla Celeron, Pentium M, Core II Duo ve i3, i5, i7 ve i9 serilerini piyasaya sürmüştür.

Tablo 2.1: 2000'li yıllara kadar piyasaya sürülen Intel işlemciler ve özellikleri[13]

	Tarih	**Transistör**	**Mikron**	**Saat Frekansı**	**Veri Genişliği**	**MIPS**
8080	1974	6,000	6	2 MHz	8 bit	0,64
8088	1979	29,000	3	5 MHz	16 bit-8 bit	0,33
80286	1982	134,000	1,5	6 MHz	16 bit	1
80386	1985	275,000	1,5	16 MHz	32 bit	5
80486	1989	1,200,000	1	25 MHz	32 bit	20
Pentium	1993	3,100,000	0,8	60 MHz	32 bit-64 bit	100
Pentium II	1997	7,500,000	0,35	233 MHz	32 bit-64 bit	~300
Pentium III	1999	9,500,000	0,25	450 MHz	32 bit-64 bit	~510
Pentium IV	2000	42,000,000	0,18	1,5 MHz	32 bit-64 bit	~1,700

Intel firmasının günümüzde en büyük rakibi olan AMD firması ise ilk mikroişlemcisi olan AM386'yı 1991 yılında tanıtmıştır. 2000'li yıllara gelene kadar ise K5, K6, K6-II, K6-III ve Athlon işlemci serilerini piyasaya sürmüştür. 2000'li yıllarda ise Duro, Opteron, Sempron serilerini piyasaya sürmüştür. İlk çift çekirdekli işlemcisini 2005 yılında Athlon 64 X2 olarak piyasaya sürmüştür. Daha sonra ise Phenom işlemci serisi üretilmeye başlanmıştır.

2.1.2. Mikroişlemci Yapısı ve Çalışma Şekli

Mikroişlemci, bilgisayar birimlerinin çalışmasını ve bu birimler arasındaki veri (data) akışını kontrol eden, veri işleme (verileri değerlendirip yeni veriler üretme) görevlerini yerine getiren donanım birimidir. MİB (merkezi işlem birimi), CPU(central processing unit) olarak da adlandırılır. [3-5-7]

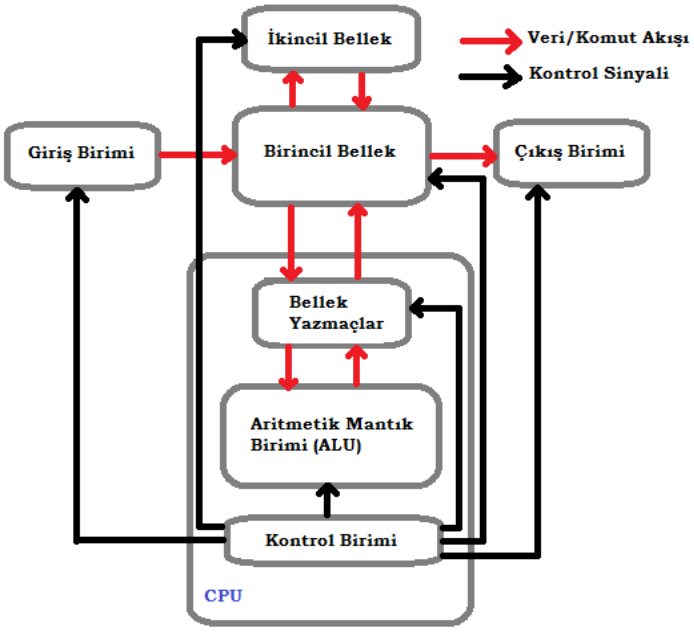
Bir mikroişlemci bulunduran sistemde yapılmak istenen tüm işlemler CPU tarafından gerçekleştirilir. Mikroişlemciler, açma kapama anahtarı gibi çalışan milyonlarca transistordan oluşmaktadır. Bu anahtarların programlanma durumuna göre elektrik sinyalleri bunların üzerinden akar. Bu sinyaller, bilgisayarın yaptığı tüm işleri toplama, çıkarma, çarpma ve bölme gibi temel matematiksel işlemlere indirir. İşlemci de bu işlemleri en basit sayma sistemi olan ikilik sayma sistemini kullanarak yapar. Buna göre "1" elektrik sinyalinin olduğunu yani yüksek gerilimi (high voltage) "0" ise elektrik sinyalinin olmadığını yani düşük gerilimi(low voltage) ifade eder. [1-3-5-7]

Bu sayı grupları üzerinde işlem yapmak için işlemci içerisinde bir takım komut listesinden ibaret bir program mevcuttur. Bu komutlar işlemciye hangi işlemin yapılacağı yönünde emir verebildiği gibi klavyeden girilen tercihlere göre bir takım komut satırını atlayıp (şartlı dallanma - conditional branch) diğer komut satırlarını işlemeye de devam edebilir. Temel olarak, mikroişlemcinin yaptığı iş, bitler üzerinde işlem yapmak üzere komutları çalıştırmaktır.

Bir mikroişlemci bulunduran sistemi için, mikroişlemcinin yanı sıra yardımcı elemanlara ihtiyaç duyulur. Bunlar: Input (Giriş) birimi; Output (Çıkış) birimi; Memory (bellek) birimidir.

CPU, Giriş/Çıkış ve Bellek birimlerinin oluşturduğu sisteme mikrobilgisayar adı da verilir. Giriş/Çıkış ve Bellek elemanları mikrobilgisayar kartı üzerinde bir yerde CPU chip'inden

bağımsız olarak yerleştirilmiş çiplerden ve elektronik devre elemanlarından oluşur. Aralarındaki iletişimler ise yollar aracılığı ile sağlanır (Adres bus, Data bus, Control bus). Şekil 2.1'de mikroişlemcinin(CPU) giriş/çıkış birimleri ve bellek birimi ilişkisi gösterilmiştir.



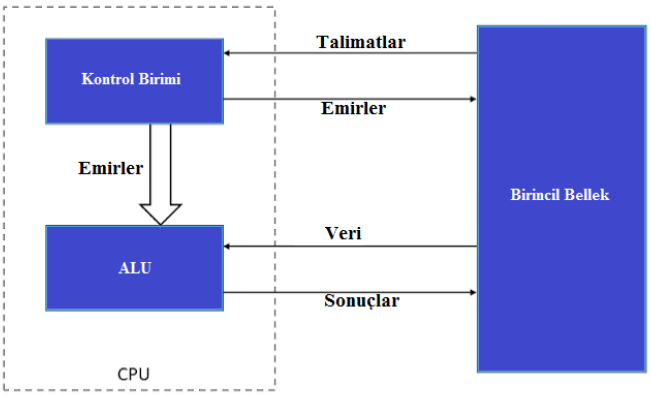
Şekil 2.1: Merkezi işlem biriminin diğer bilimler ile ilişkisi

Giriş birimlerinden alınan talimat birincil bellekte program bölümünde komut olarak tutulur. CPU içerisindeki kontrol birimi bu komutu birincil bellekten alır(fetching) ve kodu CPU içerisindeki geçici depolama birimine iletir daha sonra bu kodu çözer(decoding) ve yapılacak işlem için Aritmetik Mantık Birimine talimat verir(execute). Aritmetik Mantık birimi kodu talimata göre işler ve işlem çıktılarına yazmaçlara gönderir. Kontrol biriminin müdahalesi ile yazmaçlardaki çıktılar birincil bellekteki belirli adreslere gönderilir(store). Birincil belleğe gelen işlenmiş kod çıktısı yine kontrol biriminin müdahalesi ile çıkış birimine ulaştırılır. Tüm bu işlemler bir makine çevrimi içerisinde gerçekleşir. Makine çevrimi CPU içerisindeki kontrol biriminin yürüttüğü dört adımlık işlemden oluşur. Bunlar fetch, decode, execute, store olarak adlandırılır. Kısaca bu dört işlem, kodun getirilmesi, çözülmesi, işlenmesi ve depolanması ile ifade edilir. Örneğin Microsoft Word programında çalışırken klavye giriş biriminden alınan input'un CPU da işlendikten sonra çıkış birimi olan ekranda işlenmiş kod çıktısının ifade edilmesi CPU'da yürütülen fetch-decode-execute-store işlemlerinin sonucunda gerçekleşir. [1-3-5-7-14]

2.1.2.1. Kontrol Birimi

CPU içerisinde bulunan kontrol birimi tüm bilgisayar sistemindeki koordinasyondan sorunludur. Bilgisayar sisteminde bulunan tüm dahili ve harici sistemlerin kontrolünü sağlayan kontrol sinyallerinin üretildiği birimdir. Bununla birlikte CPU'nun en önemli birimi olan Aritmetik Mantık Birimi'nin işlemlerini gerçekleştirmesi için gerekli olan talimatları üretir. Ayrıca bilgisayar sistemindeki veri akışını kontrol sinyalleri vasıtası ile kontrol eder.

Makine döngüsünün adımları bu birim tarafından gerçekleştirilir. Birincil bellekten talimatların alınıp getirilmesi, bu talimatların çözülüp ALU'ya yürütme talimatının verilmesi ve aritmetik mantık biriminin çıktılarının birincil bellekte depolanması kontrol biriminin ürettiği sinyaller ile yürütülür. Aynı zamanda gerekli durumlarda işlemcinin durdurulması ve sistemin senkronizasyonu bu birim ile sağlanır.



Şekil 2.2: Kontrol Birimi Fonksiyonları

2.1.2.2. Kaydediciler (Registers)

CPU, basit hesaplamaları hızlı bir şekilde yapmak için tasarlanmıştır. Performansı arttırmak için, bu hesapların sonuçlarının hafızaya gönderilmeleri ve saklanması yerine yerel olarak saklanması arzu edilmektedir. Böylece, her CPU, her biri tek bir kelime tutabilen küçük bir dizi yüksek hızlı yazmaçlarla (registers) donatılmıştır.

Birincil belleğe erişim yavaş gerçekleşen bir iştir. Örneğin CPU birincil bellekte *a* adresinde tutulan içeriği almak istediğinde sırasıyla şu süreçler gerçekleşir:

- *a* adres bilgisi CPU'dan birincil belleğe gider.

- Birincil belleğin direk-erişim lojiği adresi *a* olan bellek yazmacını seçer.
- Birincil bellekteki *a* adresli yazmaç içeriği CPU'ya geri gider.

Yukarıdaki hizmeti, gidiş-dönüş ve adres arama işlemleri olmadan, kaydedicilerde de gerçekleştirmektedir. Yazmaç olarak da adlandırılan bu kaydediciler fiziksel olarak CPU içerisinde yer alırlar, bu sebeple onlara erişim neredeyse anlıktır. Ayrıca yazmaçlar birincil bellekte bulunan milyonlarca bellek hücresine kıyasla çok daha azdırlar.

Farklı CPU'lar farklı amaçlar için farklı türlerdeki farklı sayıda yazmaç kullanır. Bazı bilgisayar mimarilerinde her bir yazmaç birden fazla amaca hizmet edebilir:

Veri Yazmaçları: Bu yazmaçlar CPU'ya kısa süreli bellek hizmeti verir. Örneğin $(a - b) * c$ hesabı yapılırken önce $(a - b)$ hesabı yapılmalı ve sonucu c ile çarpılmak üzere daha sonra hatırlanmalıdır. $(a - b)$ işlemi sonucu daha sonra kullanılmak üzere geçici olarak CPU içerisindeki veri yazmaçlarında saklanır.

Adresleme Yazmaçları: CPU verileri okumak ve yazmak için sürekli olarak belleğe erişmesi gerekmektedir. Bu işlemlerin her birinde CPU'nun hangi bellek sözcüğüne erişmesi gerektiğini bilmesi gerekir, yani erişilecek bellek sözcüğünün adresinin bilinmesi gerekir. Bazı durumlarda bu adres talimatın bir parçası iken bazı durumlarda ise kendisinden önceki talimatın yürütülmesine bağlıdır. İkinci durumda adres daha sonra kullanılmak üzere geçici olarak bir yazmaçta tutulmalıdır. Adres tutan bu yazmaçlara adresleme yazmaçları denir.

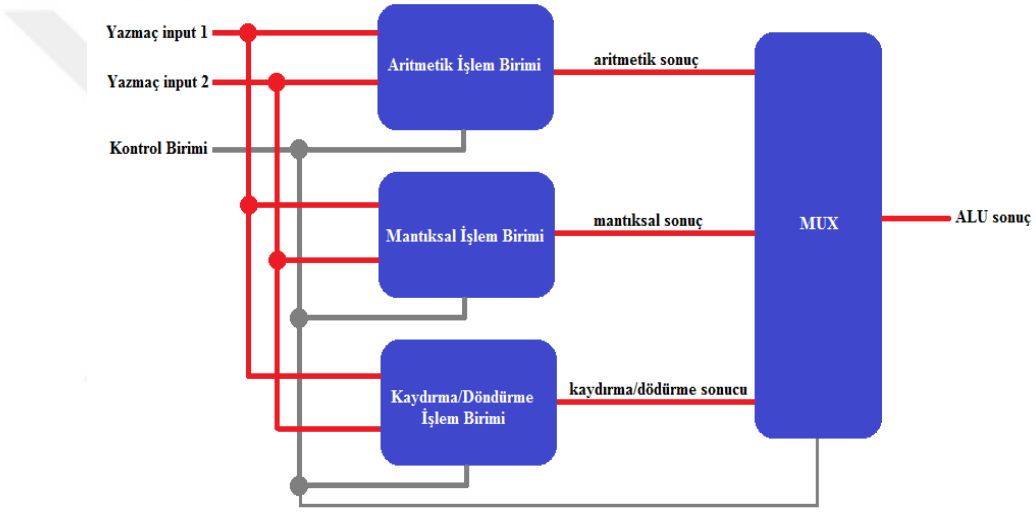
*Program Sayıcı Yazmaçlar: CPU b*ir programı yürütürken, talimat belleğinden alınması gereken bir sonraki talimatın adresini tutması gerekmektedir. Bu adres, program sayacı veya PC adı verilen özel bir yazmaçta tutulur. CPU mevcutta işlediği talimata göre program sayacının içeriğini iki şekilde günceller. Birincisi eğer mevcut talimat içerisinde herhangi bir adresten veri alma direktifi içermiyorsa program sayacı içeriği bir sonraki talimatın adresini gösterecek şekilde artırılır. Eğer mevcut program bir *n* adresine gitme direktifi içeriyorsa CPU program sayacının içeriğine bu *n* adresini yükler.

Burada verilen yazmaçlar tüm CPU'larda görülen yazmaç tipleridir. Kendi içlerinde alt kategorilere de ayrılırlar. Alt kategorilerde yer alan bazı önemli yazmaçlar, talimat yazmaçları (instruction registers), bayrak yazmaçları (flag registers), I/O tampon yazmaçları (I/O buffer

registers), I/O adres yazmaçları, bellek tampon yazmaçları (memory buffer register) ve ALU da gerçekleşen işlem sonuçlarını tutan akümülatör şeklindedir.[1-3-5-7-14]

2.1.2.3. Aritmetik Mantık Birimi

Aritmetik mantık birimi(ALU) merkezi işlem biriminin veri işleme merkezidir. CPU, bellekten aldığı talimatlar doğrultusunda aldığı veriyi çözer ve işlemesi için ALU'ya talimat verir. Farklı bilgisayarlarda farklı aritmetik mantık birimleri olmasına karşın her ALU mutlaka toplama, çıkarma, çarpma, bölme gibi temel matematiksel işlemlerin yapıldığı aritmetik ve and, or, xor gibi temel mantık işlemlerinin yapıldığı mantık birimini içerir.



Şekil 2.3: ALU blok diyagramı

ALU işlenecek veriyi CPU içerisindeki veri yazmaçlarından alır. ALU içerisinde işlenen veri tekrar depolanmak üzere data yazmaçlarına gönderilir. Daha sonra işlenen veri kontrol biriminin talimatı ve adresleme yazmaçlarındaki verinin gönderileceği adres bilgisine göre birincil belleğe gönderilir.

2.1.3. Mikroişlemci Performansı

Makine çevriminin ekinliği mikroişlemcinin performansını belirler. Temel bir makine çevriminin etkinliği CPU'nun çeşitli performans özellikleri ile saptanır. Bunlar bilgisayar çalışma hızı(clock speed), sözcük boyutu(word size), yol genişliği(bus width) gibi özellikleri ve iş hattı (pipelining), RISC(reduced instruction set computer) işlem, çoklu işlem ve çok çekirdeklilik gibi teknikleri içerir.[1-14]

2.1.3.1. Bilgisayar Çalışma Hızı (Clock Speed)

Dahili bir sistem saati makine çevriminin senkronizasyonunu sağlar. Her bir saat darbesinde CPU başka bir komutu yürütür. Kısaca saat hızı, CPU'nun temel talimatlarını gerçekleştirme hızıdır. Bilgisayarlar talimatları çok hızlı yürütür ve bu hızlar hertz ile ölçülür. Saniyede gerçekleşen 1 döngü 1 hertz olarak ölçülür. Saniyede gerçekleşen döngü sayısı saat hızı olarak ifade edilir. Mikrobilgisayarların saat hızı MHz(megahertz) ya da GHz(gigahertz) ile ölçülür. Bu da saniyede milyonlarca ya da milyarlarca döngü demektir. Daha hızlı elektronik depolama ile birleştirilen yeni yonga seti tasarımları, CPU performansını yalnızca saat hızına göre derecelendirmeyi zorlaştırmaktadır. Dahili saat hızını artırmak, CPU'nun performansını iyileştirme yöntemlerinden sadece birisidir. İşlemcinin gücünden tam anlamıyla yararlanmak için diğer performans özelliklerinin de optimize edilmesi gerekmektedir.[14]

2.1.3.2. Sözcük Boyutu (Word Size)

CPU gücüne katkıda bulunan önemli bir diğer faktör, işlemci aracılığıyla bir makine döngüsünde taşınan veri veya talimatların miktarıdır. Sözcük boyutu, işlemcinin bir makine döngüsünde bir birim olarak işleyebileceği bit grubunu ifade eder. Bir 64-bit işlemci, bir kerede 64 bit (veya 8 bayt) veriyi, 32-bit (4 byte) işlemciye göre açıkça bir avantaj olarak işleyebilir. Tablo 2.2 de farklı işlemciler için saat hızı ve sözcük boyutunu gösteren bilgiler verilmiştir.[14]

Tablo 2.2: İşlemci Derecelendirme Örneği[14]

Mikroişlemci	**Transistor**	**Saat Hızı**	**Sözcük Boyutu**
Intel Pentium IV	42 milyon	2.4 GHz	32 bit
AMD Athlon64	105.9 milyon	2.2 GHz	32 bit
AMD K10	758 milyon	1.6 GHz	64 bit
Intel i7 Quad	731 milyon	2.93 GHz	64 bit
Intel Atom Z540	47 milyon	1.86 GHz	64 bit
Intel i7 Six Core	2.2 milyar	3.9 GHz	64 bit

2.1.3.3. Yol Genişliği (Bus Width)

Diğer bir performans faktörü, işlemci ve bellek arasındaki sistem veri yolunun genişliğidir. Veri yolu, veri akışı için bir elektronik yoldur. Sistem veri yolu ise, CPU, bellek ve diğer sistem bileşenleri arasındaki elektronik yoldur. İşlemcinin belleğe iki veri yolu bağlantısı vardır: veri yolu ve adres veriyolu. Veri yolu, bellek ve CPU arasındaki gerçek verileri taşıyan yolların kümesidir. 64 bit veri yolu, 8 bayt veriyi 1 makine döngüsünde işlemciye taşıyabilir. Veri yolu genişliği, optimum performans için CPU'nun kelime boyutuna uygun olmalıdır.

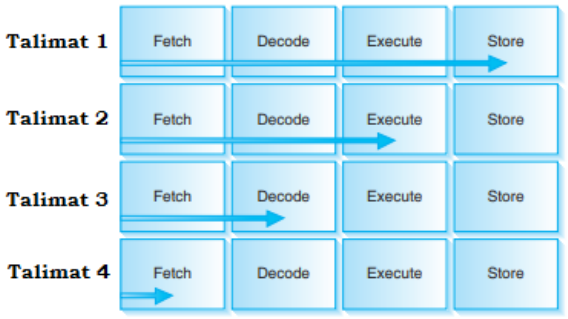
Adres veriyolu, verilerin bellekteki konumları hakkında bilgi taşıyan elektronik yoldur. Adres veriyolunun genişliği, bir işlemcinin algılayabileceği potansiyel belleği belirler. Daha büyük adres verilyolları, işlemcinin daha fazla belleği ele alabileceği anlamına gelir. 64 bit adres veriyolu kullanan işlemciler, 16 GB (2^{64}) bellek potansiyeline erişebilir. İşlemcinin geniş bir adres alanına erişme yeteneği, multimedya uygulamaları için önemlidir. Dijital video, ses ve görüntüler büyük veri dosyaları üretmektedir. Daha büyük adres veriyolları ile multimedya dosyaları sabit sürücü, CD-ROM'lar, DVD'ler veya diğer çevresel depolama aygıtlarından alınıp verinin daha hızlı işlendiği elektronik belleğe taşınmasını sağlayarak bu multimedya dosyaların daha hızlı görüntülenmesine ve işlenmesine destek olur. [14]

2.1.3.4. İş Hattı (Pipelining)

Intel, Motorola ve IBM gibi mikroişlemci üreticileri, CPU'nun hızını en üst düzeye çıkarmak için ek teknikler kullanmaktadırlar. Pipelining ile bir talimat okunduktan sonra okunan komut işlenmeye başlandığı anda başka bir talimat da okunmaya başlanır ve böylelikle CPU'nun verimliliği artırılmaktadır. Pipelining kullanılarak, makine döngüsünün farklı adımları aynı anda çeşitli talimatlar üzerinde gerçekleştirilebilir. Bu, CPU'nun boşta kalma süresini azaltır ve talimatların gerçekleştirilme hızını artırır. [14]



Şekil 2.4: Pipelining'siz Makine Döngüsü[14]



Şekil 2.5: Pipelining'li Makine Dögüsü[14]

2.1.3.5. RISC(Reduced Instruction Set Computer) İşlem

RISC (azaltılmış komut setli bilgisayar) tekniği kullanılan yongalar da CPU'nun genel hızını artırır. RISC yongaları karmaşık gömülü mikrokod komutlarını ortadan kaldırır ve bunları işlemci tarafından kullanılan en yaygın komutları yürütmek için daha basit bir talimat kümesiyle değiştirir. Bu ortak komutlar daha az sıklıkta fakat daha karmaşık işlem görevlerini yürütmek için birleştirilir. Bu, rutin operasyonların daha hızlı işlenmesine neden olur. RISC teknolojisi birçok mikroişlemcide bulunur ve kullanımı hızla cep telefonu işlemcilerine kadar genişlemiştir. [1-14]

2.1.3.6. Çoklu İşlem(Multiprocessing)

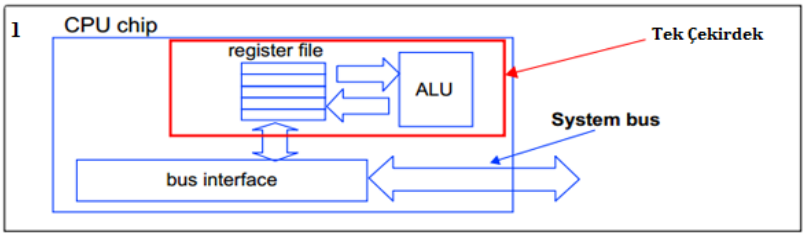
Çoklu işlem de sistem performansını artırabilir. Çoklu işlem, sistem kartındaki komutları aynı anda yürüten çoklu işlemcilerin birleşimidir. Çoklu işlemede çeşitli yaklaşımlar vardır. Bilgisayarlar, bir sistemin genel performansını artırmak için CPU ile birlikte çalışan bir matematik işlemcisi veya grafik işlemcisine sahip olabilir. Apple G5 bilgisayarında, 64-bit yongalı çift işlemciler vardır. Bu işlemciler CPU'yu hızlandırır ve bir süper bilgisayarla karşılaştırılabilecek işlem gücüne ulaşılmasını sağlar. Multimedya gelişimi, aynı zamanda tek bir görevde birden fazla işlemcinin aynı anda çalışması için bir bağlantı tekniği olan paralel işlemeyi de kullanabilir. Paralel işleme genellikle görselleştirme olarak bilinen 3-D animasyonunun son aşamasını hızlandırmak için kullanılır. [1-14]

2.1.3.7. Çok çekirdeklilik (Multicore)

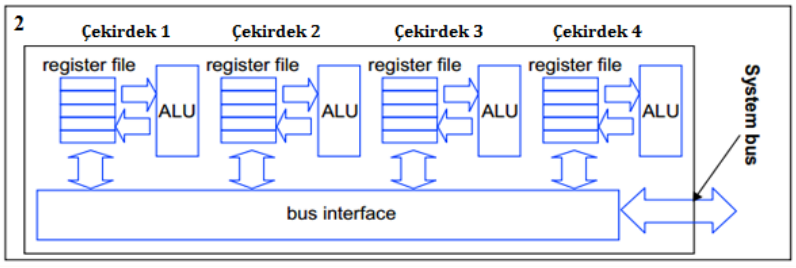
Çok çekirdekli işlemciler, kişisel bilgisayarlarda çoklu işlem yapmak için giderek daha da popülerleşen bir yaklaşımdır. Bir işlemci çekirdeği, CPU'nun "hesaplama motoru", yani

doğrudan talimatları uygulayan bileşenidir. Eski işlemciler tek çekirdekli olarak tasarlanmışlardır. Bu işlemcilerde CPU, pipeline'daki her komutu mümkün olduğunca hızlı bir şekilde getirip çalıştıran tek bir hesaplama motoru etrafında inşa edilmiştir. Daha sonraları işleme kapasitesini arttırmak için mühendisler saat hızlarını artırmışlar ve CPU'ya transistör ekleyen başka iyileştirmeler yapmışlardır. Yapılan bu değişiklikler neticesinde çip üzerindeki ısıyı artmış ve mevcut bilgi-işlem talepleri için gerekli performans kazanımlarını elde edememişlerdir. Üreticiler, ısıyı düşürürken ve enerji verimliliğini artırırken işlem hızını arttırmak için farklı bir strateji arayışına girmiş ve sonucunda çok çekirdekli mimari ortaya koyulmuştur. Bu mimaride CPU içerisinde bulunan her çekirdeğin ayrı ALU birimleri ve önbellekleri bulunmaktadır. [1-14]

Çok çekirdekli teknoloji, iki veya daha fazla işlemci çekirdeğini ve önbelleği tek bir tümleşik devrede birleştirmiştir. Çift çekirdekli bir işlemci iki yürütme çekirdeği ve iki L2 seviye önbelleği içermektedir.



Şekil 2.6: Tek Çekirdekli CPU yapısı[15]



Şekil 2.7: Çok Çekirdekli CPU yapısı[15]

Bu iki "çekirdek" aynı anda farklı görevleri yerine getirmek için birlikte çalışır. Giderek daha da karmaşıklaşan işlemlerde, bilgisayar yanıt süresini daha da geliştirmek için sisteme daha fazla çekirdek eklenebilmektedir. Çok çekirdekli işlemciler, bir işlemci çekirdeği bir görevi ele alırken ikinci bir görevin başka bir çekirdek tarafından yürütülmesi gibi çoklu görev ortamlarında bilgisayar performansını önemli ölçüde artırabilir. Örneğin, bir çekirdek DVD

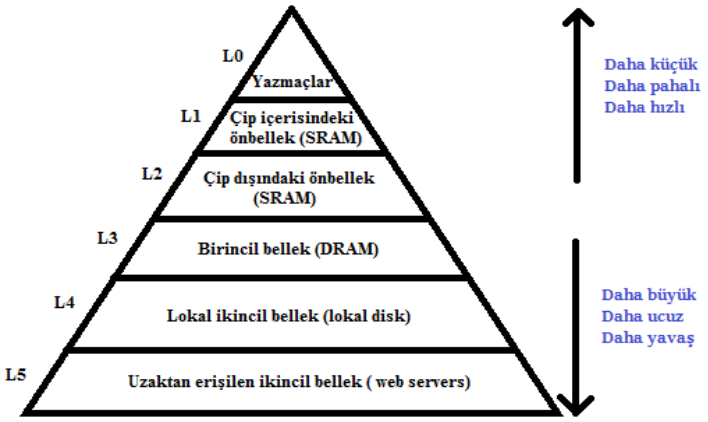
yazma işlemini yürütürken Photoshop'ta görüntü düzenleme işlemi ikinci çekirdeğe yönlendirilebilir. Çok çekirdekli bir işlemci her iki görevi de daha hızlı yürütür çünkü her bir çekirdek ayrı bir görevi ele alabilmektedir.

CPU'nun getirme/yürütme (fetch/execute) döngüsü, hızı artırmak ve daha karmaşık veri türleri ile uğraşmak için çeşitli teknikler kullanılarak optimize edilmiştir. Saat hızını, sözcük boyutunu ve sistem veri yolunu artırarak, kontrol ünitesi ve ALU verileri daha hızlı işleyebilmektedir. Küme komut işleme, çoklu işlem ve çok çekirdekli yongalardaki yeni gelişmeler, gittikçe karmaşıklaşan komutlarla başa çıkabilecek daha hızlı işlemciler üretmektedir. [1-14]

2.1.4. Bellek Hiyerarşisi

Bellek sistemi doğrusal bir bayt dizisidir ve CPU, her bellek konumuna sabit bir sürede erişebilir. Bu, şimdiye kadar etkili bir model olsa da, modern sistemlerin gerçekte nasıl çalıştığını yansıtmamaktadır.

Pratikte, bir bellek sistemi, farklı kapasitelere, maliyetlere ve erişim sürelerine sahip bir depolama cihazları hiyerarşisidir. CPU'da bulunan yazmaçlar en sık kullanılan verileri tutar. CPU'nun yakınındaki küçük ve hızlı cache bellekleri, nispeten daha yavaş olan ana bellekteki veri ve talimatlar için depolama alanı işlevi görür. Ana bellekler ise network aracılığı ile bağlanan harici makinelerin büyük ve yavaş disklerdeki veriler için depolama alanı işlevi görür. CPU'dan uzaklaştıkça bellek hızı düşerken bellek boyutu artar.[1-3-5-7-14]



Şekil 2.8: Bellek Hiyerarşisi

Her seviyedeki bellek tipi kendisinden bir sonraki bellekten alınan veriler için depolama alanı oluşturur. L0 ve L1 seviye bellekler CPU çekirdeğinde yer alır. Kapasiteleri kilobayt seviyelerindedir. L2 seviye bellekler çekirdek dışında fakat CPU içerisinde yer alır. Kapasiteleri megabayt seviyelerindedir. Diğer bellekler CPU dışındadır ve kapasiteleri sırasıyla gigabayt, terabayt ve petabayt seviyelerindedir. L0, L1, L2 ve L3 seviyelerindeki bellek hızları 1- 30ns iken diğer seviyelerdeki bellek hızları 1-100ms arasında değişir. L0, L1, L2 ve L3 seviyelerindeki bellekler uçucu hafızaya sahipken diğer seviyelerdeki bellekler uçucu olmayan belleklerdir. [1-3-5-7-14]

Bellek yapılarını anlamak için RAM (random access memory) ve ROM (read-only memory) yapılarının incelenmesi gerekmektedir.

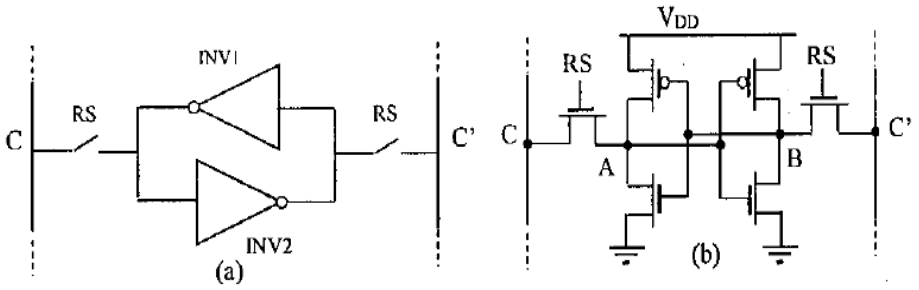
2.1.4.1. RAM (Random Access Memory)

RAM (Rastgele Erişimli Bellek), işletim sisteminin, uygulama programlarının ve mevcutta kullanılan verilerin bilgisayar işlemcisi tarafından hızla erişilebilecek şekilde saklandığı bellektir. Sabit disk, disket ve CD-ROM gibi belleklere nazaran RAM'e veri yazmak ya da RAM'dan veri okumak çok daha hızlıdır. Fakat veri sistemde elektrik olduğu sürece RAM'de tutulur. Elektrik kesintisinde, RAM'de tutulan veri kaybolur. Sisteme tekrar elektrik verildiğinde işletim sistemi ve diğer dosyalar hard disk üzerinden tekrar RAM'e yüklenir.

Yapı olarak satır ve sütun kod çözücüler ile erişilebilen hücrelerden oluşur ve her bir hücre 1 bitlik veri saklar. Her bir hücreden veri okunabildiği ve yazılabildiği için bu belleklere yaz-oku bellek de denilmektedir. Bu belleklerin salt okunur belleklere göre dezavantajı uçucu olmalarıdır. Bu yüzden bu belleğin kullanıldığı cihazlarda besleme kesildiğinde RAM içerisinde verinin silinmemesi için enerji sağlamaya devam edecek ilave bir pil kullanılır. Ferroelektrik RAM gibi bazı özel üretim teknolojileri kullanılarak uçucu olmayan RAM bellekleri de üretilmektedir. [1-3-5-7]

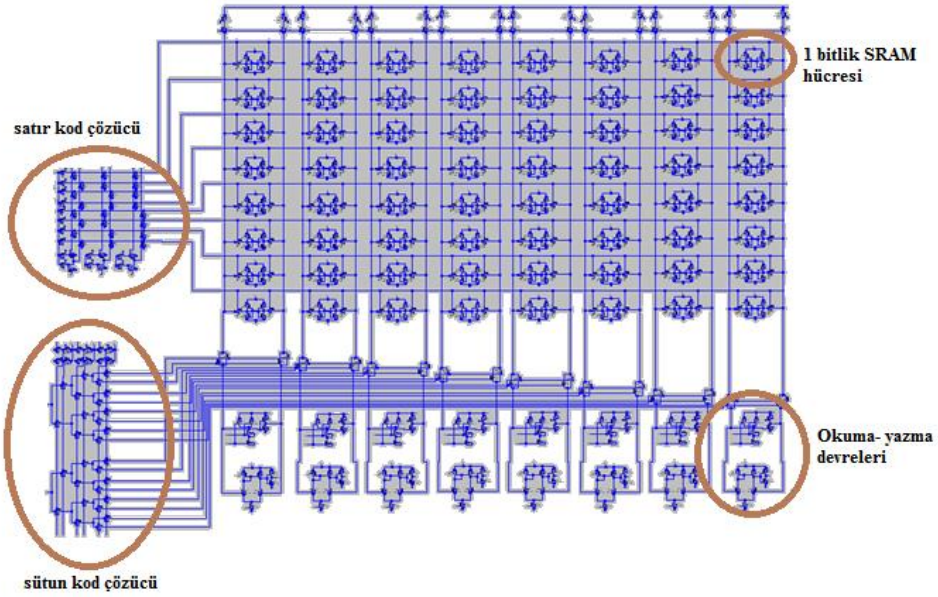
Çalışma şekillerine göre rastgele erişimli bellekler SRAM (statik RAM), DRAM (dinamik RAM) ve FeRAM olmak üzere üçe ayrılır.

SRAM (statik rastgele erişimli bellek): mikroişlemci içerisindeki L1 ve L2 seviye önbellekler SRAM tabanlıdır. Çok hızlı çalışırlar. Bir SRAM hücresi yapısal olarak, pozitif geri beslemeli iki eviricinin sırt sırta bağlanmasıyla oluşturulmuştur. Hücrede 4 ila 6 transistor bulunabilir.



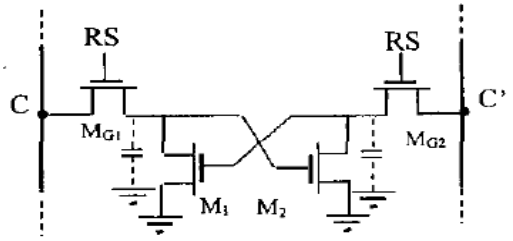
Şekil 2.9: a) 1 Bitlik SRAM hücresi, **b)** Bellek hücresinin CMOS devresi[2]

Şekilde RS satır seçimini yani satır kod çözücüden gelen bilgiyi, C ise sütun seçimini yani sütun kod çözücüden gelen bilgiyi ifade eder. C ve C' birbirinin evriğidir. RS girişi '1' olduğu durumda bellek hücresinin çıkışları bu hatlara bağlanmış olur. Yatay hatlar kelime hatları düşey hatlar ise bit hatları olarak adlandırılır. Aşağıda 64 bitlik bir SRAM verilmiştir.[2]



Şekil 2.10: 64 Bit SRAM

DRAM (dinamik rastgele erişimli bellek): SRAM bellekleri çok hızlı olmalarına rağmen çok pahalıdırlar. Tümleştirme yoğunluğu artırılarak belirli bir kırmık alanına daha büyük bellek kapasitesi elde etmek amacıyla bu belleklerin yerine dinamik rastgele erişimli bellekler kullanılır. DRAM hızları SRAM'lere oranla daha düşüktür ve yapı içerisinde kapasitör bulunduğu için tazeleme süreçlerine ihtiyaç duyarlar.

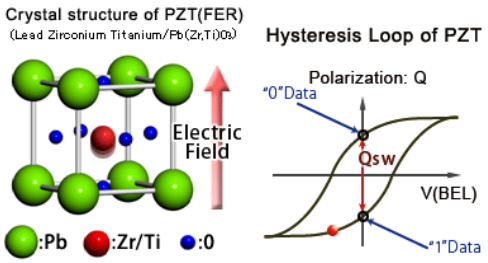


Şekil 2.11: 4 Transistörlü DRAM hücresi[2]

Statik RAM hücresinde verinin saklanması transistörler aracılığı ile bazı düğümlerin "1" veya "0" seviyelerine zorlanması ile gerçekleşirken dinamik RAM hücrelerinde ise veri kapasitör üzerinde biriktirilen yük yardımıyla sağlanmaktadır. Kondansatör üzerinde pozitif bir gerilim var ise(kondansatör dolu) saklanan veri "1", kondansatör üzerindeki gerilim sıfır ise (kondansatör boş) saklanan veri "0" dır. Fakat transistörler üzerindeki kaçak akım sebebi ile kondansatör üzerindeki yük boşalır dolayısı ile veri kaybolur. Bunun engellenmesi için veri tazeleme işlemi gerçekleştirilir. Ayrıca dinamik RAM üzerinde veri okunma esnasında da veri kaybı riski olduğundan okuma işlemi ardından verinin eski haline getirilmesi için de ayrı bir işlem yapılır. Bu tazeleme işlemleri bellek hızını düşürmektedir. [2]

FeRAM (Ferroelektrik rastgele erişimli bellek): SRAM ve DRAM bellekleri besleme kesildiğinde ve kondansatörler boşaldığında sakladıkları veriyi kaybederler. Bu da bu bellekleri uçucu hafızalı yapar.

Ferroelektrik RAM'de DRAM yapısına çok benzer bir yapı kullanılmıştır. Yapıda bir transistör ve bir kondansatör bulunur. DRAM'den farklı olarak kondansatörün dielektrik malzemesi olarak paraelektrik malzeme (SiO_2) yerine ferroelektrik malzeme kullanılır.



Şekil 2.12: Ferroelektrik kristal yapısı[16]

Ferroelektrik kristale bir elektrik alan uygulandığında kristalin merkezindeki atomlar elektrik alan yönünde hareket ederler. Atom kristalin içinde hareket ederken, bir enerji bariyerinden

geçer ve bu da bir yük artışına sebep olur. Bu yük artışı ile iç devre bellekte verinin tutulmasını sağlar. Elektrik alan ortadan kaldırıldığında merkez atomlar bulunduğu noktada kalır ve hafıza durumu sabit olarak korunur. Bu da herhangi bir tazeleme işlemine gerek kalmadan FeRAM'in kalıcı hafızaya sahip olmasını sağlar.

Tablo 2.3: RAM teknolojileri karşılaştırma tablosu

	SRAM	**DRAM**	**FeRAM**
Bellek Tipi	Uçucu	Uçucu	Kalıcı
Veri Yazma Süresi	85 ns	70ns	100ns
Veri Yazma Gerilimi	85ns	3.3V	100ns
Veri Okuma Süresi	3nJ	70ns	1nJ
Veri Koruma Akımı	Var	Var	Yok
Alan/Hücre	3.3V	2	2-5V
Güç Tüketimi	4	4nJ	4

2.1.4.2. ROM (Read-Only Memory)

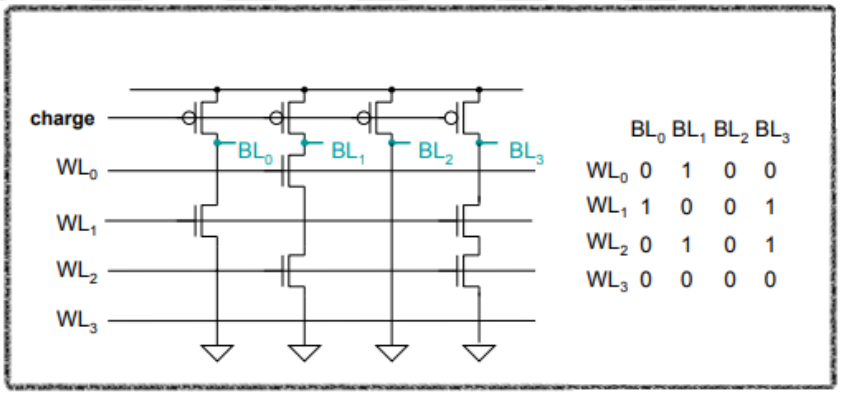
Salt okunur bellekler bilgisayar ve diğer elektronik cihazlar için sabit değerlerin, değişmesi istenmeyen kontrol bilgilerinin ve bazı komutların saklanması için kullanılan veri saklama alanıdırlar. Salt okunur bellek olarak da adlandırılırlar. ROM'lar metal maske aşamasında fabrikada programlandıkları için daha sonra programlanmaları mümkün olamamaktadır.

ROM tek seferlik programlanmış ve uçucu olmayan hafıza yapısına sahip belleklerdir. Tekrar programlanmaları ya da içeriklerinin değiştirilmesi mümkün değildir.

Yapısındaki lojik kapı tipine göre iki farklı ROM yapısı vardır. Bunlar NOR (not or) tabanlı salt okunur bellek ve NAND (not and) tabanlı salt okunur belleklerdir.

Nand bazlı belleklerde, yapı her bir sütunun satır girişlerine göre NAND işlemi gerçekleştirmektedir. Geçitleri topraklanmış olan PMOS transistorlar paralel bağlı NMOS'ların oluşturduğu NAND kapılarının aktif yükleridir. Satır sütun dizisinin kesişme noktalarının bazılarında bir NMOS transistoru bulunurken bazılarında ise kısa devre

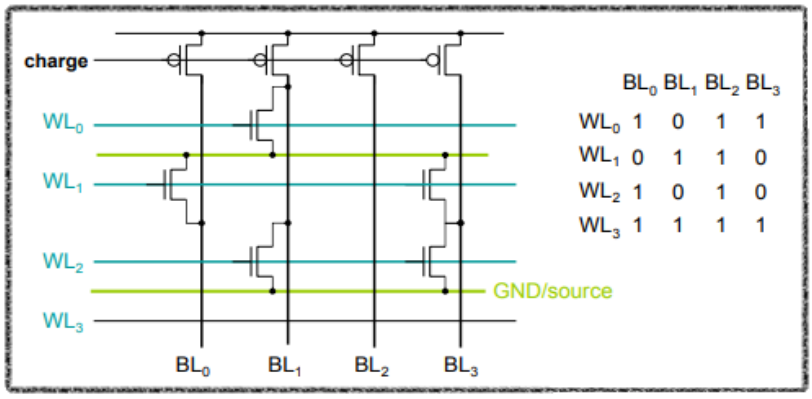
olmamaktadır. Herhangi bir satır girişi, diğer satırlar lojik-1 iken, lojik-0 seviyesine çekildiğinde, geçitleri o satıra bağlı bulunan transistorlar kesime geçer ve sütun kesişme noktasını lojik-1 seviyesine çıkarır, kesişme noktasında NMOS bulunmayan sütunlar ise lojik-0'de kalır. Başka bir ifadeyle lojik-1 bilgisi için kesişme noktasında transistor bulunması, lojik-0 bilgisi için kesişme noktasında transistor bulunmaması gerekir. Programlama ise bir ROM yapısı olduğu için fabrikasında metal maske aşamasında yapılmaktadır. "1" olması gereken yerlerde transistor bağlantısı yakılarak sağlanırken, "0" olması geren yerlerde transistor bağlantıları yapılmaz. ROM çekirdeğinde transistorlar seri bağlandıkları için sütunların alçak seviye(lojik-0) değerlerini belli bir değerin altında tutulması açısında satır sayısı çok artırılamaz. Bu nedenle yapı çok büyütülemez ve çok girişli ROM çekirdekleri NOR tabanlı sağlanır.[2]



Şekil 2.13: NAND tabanlı ROM yapısı[17]

NOR tabanlı salt okunur belleklerde ise yapı her bir sütunun satır girişlerine göre NOR işlemi yapmaktadır. Geçitleri topraklanmış olan PMOS transistorlar paralel bağlı NMOS'ların oluşturduğu NOR kapılarının aktif yükleri olup, bu halleriyle sözde NMOS NOR kapılarını oluştururlar. Satır sütun dizisinin kesişme noktalarının bazılarında bir NMOS transistoru bulunurken bazılarında ise mevcut olmamaktadır. Herhangi bir satır girişi lojik-1 seviyesine çıkarıldığında, geçitleri o satıra bağlı bulunan transistorlar iletime geçer ve sütun kesişme noktasını lojik-0 seviyesine çeker, kesişme noktasında NMOS bulunmayan sütunlar ise lojik-1'de kalır. Başka bir ifadeyle lojik-1 bilgisi için kesişme noktasında transistor bulunmaması, lojik-0 bilgisi için kesişme noktasında transistor bulunması gerekir. Programlama ise bir ROM yapısı olduğu için fabrikasında metal maske aşamasında yapılmaktadır. "1" olması gereken yerlerde transistor bağlantısı yapılmazken, "0" olması geren yerlerde transistor

bağlantıları yakılarak sağlanır. Yapının dezavantajı ise seçilen kelime hattında iletime giren transistorların neden statik güç kaybıdır.[2]

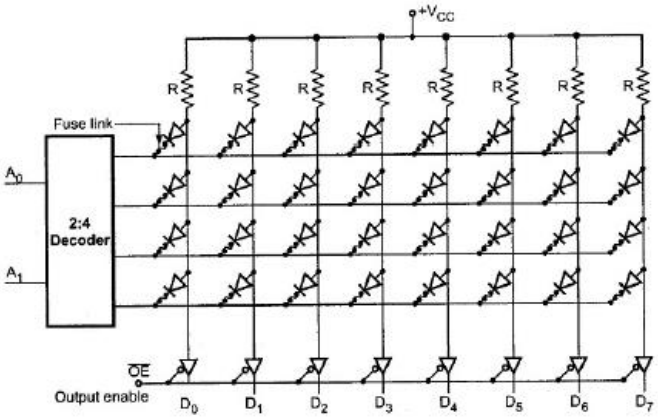


Şekil 2.14: NOR tabanlı ROM yapısı[17]

WL (Word line) ve BL (bit line) satır ve sütun çözücülerinden bilgi alırlar. Aynı anda sadece tek bir WL seçilebilmektedir.

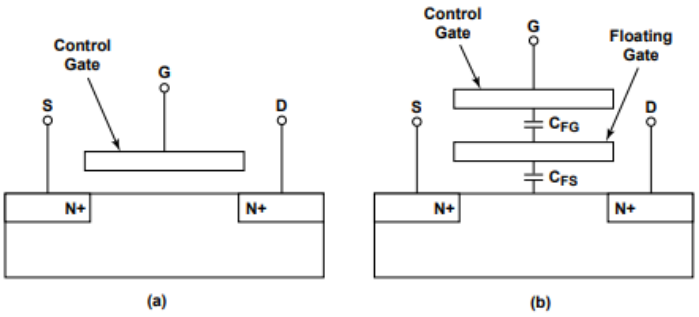
PROM (programlanabilir salt okunur bellek): ROM yapısından çok küçük bir farklılıkla satır-sütun kesişim noktalarına konan diyot-transistor gibi çeşitli aktif elemanlar sayesinde gerçekleştirilen yapılardır. Programla artık kullanıcı tarafından yapılacak olsa da sadece 1 defa olmaktadır. İşlem istenilen satır-sütun kesişimindeki elemanların belli bir gerilimin üzerine çıkarılarak yakılması sonucu olmaktadır. Yakma işlemi geri döndürülemez bir işlemdir. Diğer kesişim bölgelerindeki transistorlar ise sağlam bırakılır ve o bölgede transistor yokmuş gibi işlem yapılır.

PROM yapısındaki programlanması istenilen kesişim bitine yüksek voltajlı bir "pulse" uygulanması ile sağlanmaktadır(6V / 2nm kalınlığında Oksit veya 30 MV/cm). Transistorun geçidindeki pozitif voltaj, geçidin altındaki "substratta" bir "inversiyon" kanalı oluşturarak oksit boyunca bir tünel akımının akmasına neden olur. Akım, okside ek girdiler oluşturarak oksit içerisindeki akımı arttırır ve sonunda oksidi eritir ve geçitten tabana iletken bir kanal oluşturur. İletken kanalı oluşturmak için gereken akım yaklaşık 100µA / 100nm2'dir ve yaklaşık 100µs veya daha az bir süre gerekir.[2]



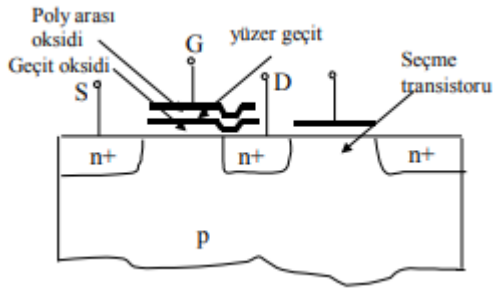
Şekil 2.15: PROM yapısı[2]

EPROM (silinebilir-programlanabilir salt okunur bellek): EPROM bellekler elektriksel olarak programlanabilen ve istenildiğinde bazı yöntemler ile silinebilen salt okunur belleklerdir. Zamanın bilgiyi depo edip elektrik beslemesi kesilse dahi bilgiyi kaybetmeme niteliğindeki aygıtları arasında en kullanışlı olanı olmuştur. Klasik bir ROM veya bir PROM'dan farklı olarak satır ve sütun kesişmelerinde aktif eleman olarak sıradan bir NMOS yerine bir FGMOS teknolojisi kullanılmasıyla oluşturulmuşlarıdır. EPROM hücresindeki bilgi FGMOS teknolojisi kullanılarak saklanmıştır. Saklanan bilginin silinmesi için FGMOS'un yüzer geçidinin tuttuğu yükün uzaklaştırılması gerekir. Bu amaçla yüzer geçide dalga boyu yani 400nm'den daha az(2537 Angstrom)Ultraviyole ışının yaklaşık 20 ila 30 dakika boyunca uygulanması gerekir. Bunun gerçekleşebilmesi için EPROM'un ROM kılıfı üzerinde kuvartz bir pencere oluşturulmuştur. Ancak silinme işlemi sırasında EPROM bulunduğu devreden çıkarılmalıdır, bu önemli bir dezavantajdır. Bu silme işlemini gerçekleştirmek üzere "EPROM Eraser" cihazları mevcuttur.[2]



Şekil 2.16: a) Geleneksel MOS **b)** Floating gate MOS[2]

EEPROM (elektrikle silinebilir-programlanabilir salt okunur bellek): EPROM belleklerin kullanıcıları için en büyük dezavantajı bulundukları devreden sökülmeden silme işleminin gerçekleştirilememesiydi. Bunun için bilgi saklayan transistor hücreleri modifiye edilerek elektriksel olarak silinebilen EEPROM(E^2PROM) geliştirilmiştir.



Şekil 2.17: EEPROM Hücresi (FLOTOX)[18]

Flotox hücresinin yapısında bir adet tek geçitli seçme transistoru ve bir adet çift geçitli saklama transistoru içermektedir. EEPROM'un tek dezavantajı ise EPROM'a göre daha fazla yer kaplamasıdır. Bu yapılarda yükün geçit ve taban arasında taşınmasını sağlayan birden çok mekanizma olsa da en çok kullanılan yöntemler Sıcak Taşıyıcı, Fowler-Nordheim Tünelleme Etkisi(Kaliteli SiO2 gerektirir) ve Polyoksit(Daha düşük kalitede SiO2 yeterlidir) İletimi yöntemleridir.[2]

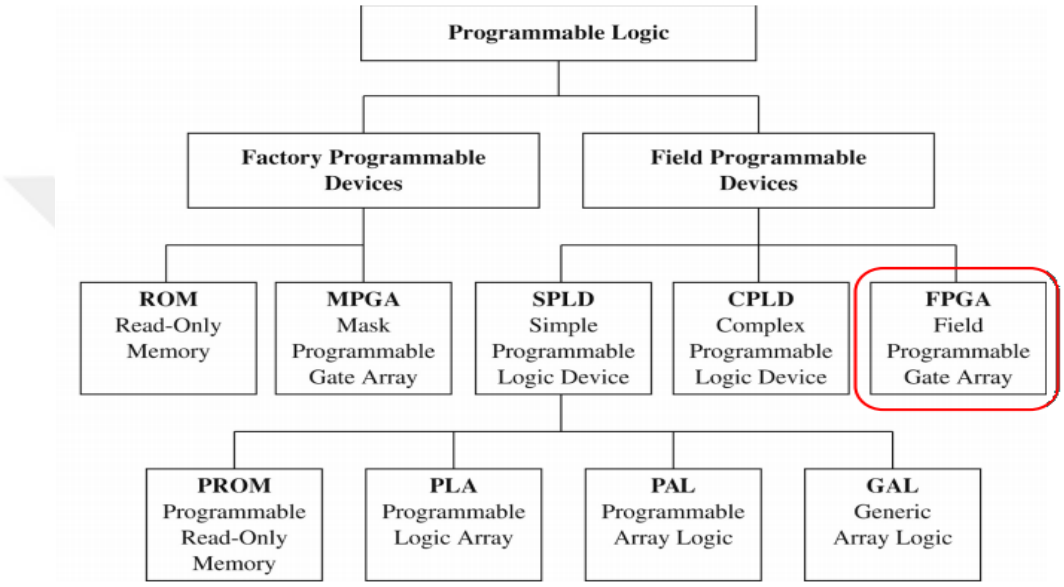
2.2. ALANDA PROGRAMLANABİLİR KAPI DİZGELERİ (FPGA)

2.2.1. Programlanabilir Mantık Cihazlarının Gelişimi

İlk programlanır gömülü devreler genel olarak "programmable logic device (PLD)" olarak tanıtılır. İlk programlanabilir bileşenler, 1970 yılında geliştirilen ve basit yapılı olan PROM'lardır. Fakat programlanır mantık cihazları olarak PLD' ler kabul edilir.

Programlanabilir mantık, salt okunur belleklerin geliştirilmesi ile başlar. FGMOS teknolojisinin ROM belleklerde kullanılması ile programlanabilen ve silinebilen salt okunur bellekler geliştirilmiştir. Programlama süreci iki farklı tipte gerçekleştirilir. Bunlardan ilki fabrikalarda üretim aşamasında gerçekleştirilen programlamadır ve bu programla yazılan veriler kalıcıdır, değiştirilemezler. ROM ve MROM(mask programmable read only memory) bu yöntemle programlanırlar. İkinci yöntem ise alanda programlama yöntemidir. Bu yöntem birçok farklı tekniği içinde barındırmaktadır. En ilkel alanda programlanabilen mantık devresi

PROM belleklerdir. Daha sonra sırası ile PLA (programmable logic array), PAL(programmable array logic) ve GAL (generic array logic) geliştirilmiştir. Bu dört alanda programlanabilen mantık dizileri, SPLD (simple programmable logic device) grubuna dahil edilmektedir. Basit programlanabilir mantık cihazlarının ardından sırası ile CPLD (complex programmable logic devices) ve FPGA (field programmable gate array) alanda programlanabilir cihazlar geliştirilmiştir.[31]

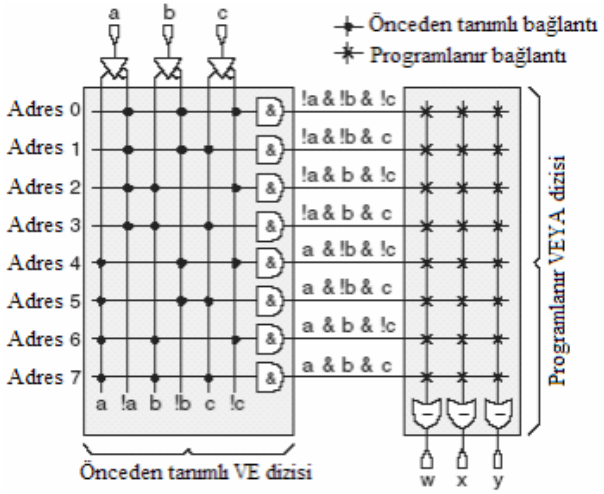


Şekil 2.18: Programlanabilir Mantık Gelişimi[19]

2.2.1.1. SPLD(Simple Programmable Logic Devices)

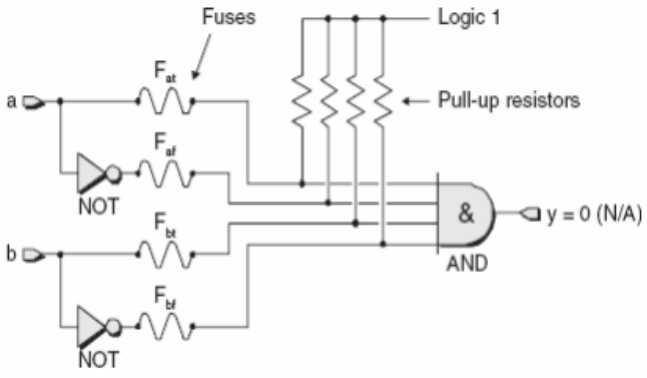
İçerdiği lojik kapı yoğunluğu birkaç yüzü aşmayan PROM, PLA, PAL ve GAL yapılarını kapsamaktadır. 1970'li yıllarda kullanılmaya başlanmıştır. İster sahada ister fabrikada programlansın bu yapılar sadece bir kez programlanabilen VE/VEYA kapılarından oluşmaktadır. SPLD'ler içerisindeki programlanabilir mantık kapılarının yeri sabit ve birbirine fiziksel olarak bağlıdırlar. Programlamaları bağlantı noktalarında bulunan sigortaların patlatılmasıyla yapılmaktaydı ve patlatılan sigortaların geri dönüşü yoktu. Boole lojiğinde SOP(som of product) formundaki mantık denklemleri için kullanılmaktadırlar. Basit yapılarından dolayı yüksek performans ile çalışırlar. SPLD'lerdeki hücrelerin yapımında genellikle sigorta, EPROM, EEPROM veya FLASH gibi değiştirilemeyen bellek hücreleri kullanılır. En ucuz programlanabilir mantık devreleridirler.[20-22-31]

PROM (programmable read only memory): En basit PLD çeşididir. Sabit VE kapısı fonksiyonlarının VEYA kapı fonksiyonlarını sürmesi şeklinde oluşturulmuştur.



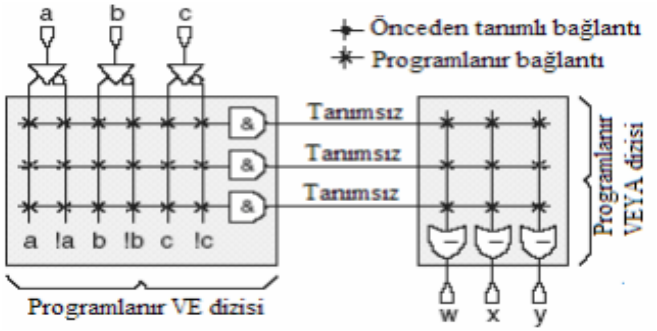
Şekil 2.19: 3 girişli, 3 çıkışlı önceden programlanmış PROM yapısı[20]

Her VE fonksiyonu a, b ve c girişlerinden gelen değerler veya bu değerlerin tümleyenlerinden üçü ile bağlantılıdır. Yine her VEYA fonksiyonu VE dizisi çıkışlarından gelen sekiz girişe sahiptir. Yukarıdaki yapı ile 3 giriş 3 çıkışlı tüm fonksiyon birleşimleri sağlanabilir. Uygulamada çok daha fazla sayıda giriş ve çıkışlara sahiptirler. İstenilen fonksiyona göre VEYA kapıları programlanır. VEYA dizisindeki programlanır bağlantılar eritilir bağlantı(sigorta), EPROM ve E^2PROM hücresi olabilir.[20-22-31]



Şekil 2.20: Sigorta bağlantılı teknoloji[20]

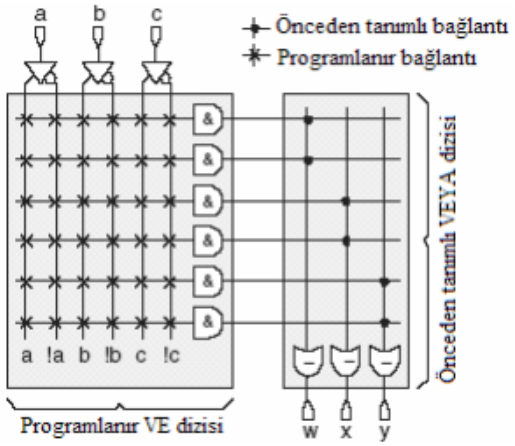
PLA (programmable logic array): PROM teknolojisindeki sadece veya kapılarının programlanmasına izin verilen kısıtlama ortadan kaldırılarak 1975 yılında PLA yapıları ortaya konulmuştur. VE ve VEYA kapılarının ikisinin de programlanmasına imkân vermesi sebebiyle PLA yapıları basit PLD' ler içerisinde kullanıcıya en fazla esneklik sağlayan yapılardır.



Şekil 2.21: 3 giriş, 3 çıkışlı programlanmamış PLA yapısı[20]

Önceden tanımlı karşıtlarına kıyasla programlanır bağlantılardan geçişler daha uzun sürdüğü için PROM'lara göre daha yavaştırlar.

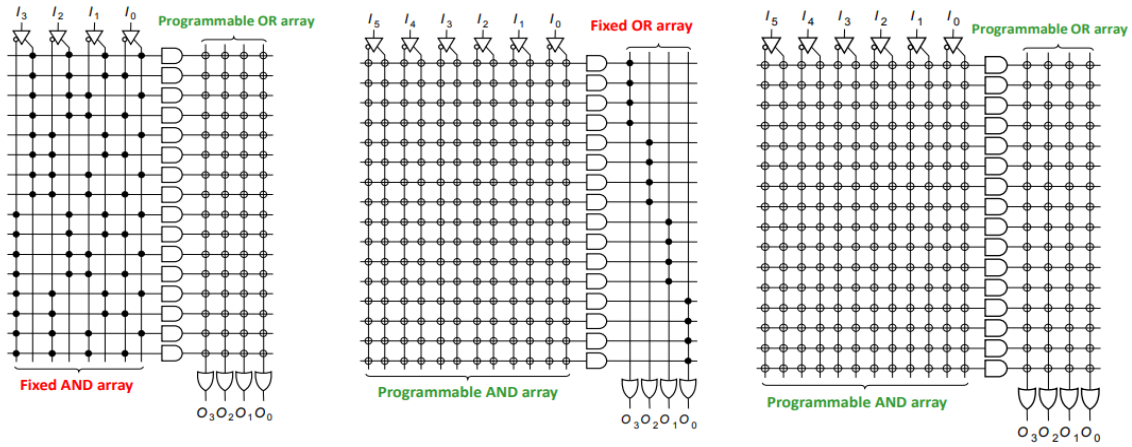
PAL (programmable array logic): Yapı olarak PROM'ların tam tersidir. VEYA kapıları yerine VE kapıları programlanabilmektedir ve VEYA kapıları önceden programlanmıştır.



Şekil 2.22: 3 giriş, 3 çıkışlı programlanmamış PAL yapısı[20]

Bu yapılar PLA' lere göre daha hızlıdırlar. Fakat VEYA kapılarından kısıtlı sayıda verinin çıkışa ulaşması dez avantajlarıdır.

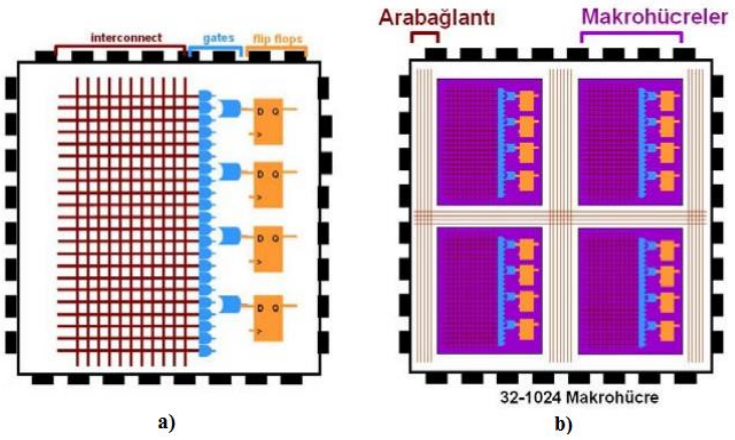
GAL (Generic Array Logic): PAL yapılarına benzemektedirler ve veri elektriksel olarak silinebilir. Üreticisi Lattice Semiconductor'dür. Programlanmaları kısıtlıdır.



Şekil 2.23: Sırası ile PROM, PAL ve PLA yapısı[20]

2.2.1.2. CPLD (Complex Programmable Logic Devices)

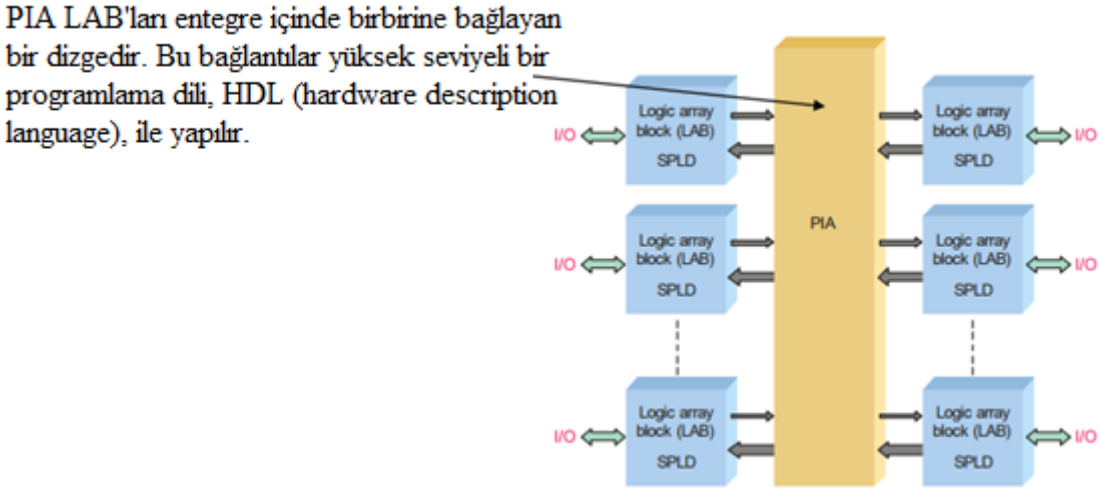
Artan kapasite ihtiyacını karşılamak amacı ile PLD yapılarının birleştirilmesi ile CPLD'ler 1984 yılında Altera firması tarafından geliştirilmiştir. SPLD'lerde olduğu gibi CMOS ve EPROM teknolojisi kullanılmıştır. Kapasiteleri SPLD'lerin 2 ila 64 katıdır. CPLD'lerde yüzlerce hücre vardır. Bu hücreler her CPLD modeline göre, 8 ile 16 hücre arasında değişen bir grubu bir fonksiyon bloğunda toparlanmış ve içerisinde hepsine direk bağlantıları yapılmıştır. Bu fonksiyon blokları arasında da iletişim sağlanmıştır. Fakat CPLD'lerdeki fonksiyon blokları arasındaki bu iletişim türünde kullanılan mantık, dolayısıyla da CPLD'nin hızı üretici firmasına göre değişir. CPLD'lerin temel mantığı matris şeklinde anahtarlama yapabilen bir birime dayanır. Bu birimin anahtarlaması sonucu işlem yapılacak lojik blok seçilir. CPLD'lerin birkaç PAL ünitesini kapsaması ve ABEL, CuPL, PALASM gibi bazı SPLD geliştirme dillerini de desteklemesi piyasanın CPLD kullanımına geçişini hızlandırmıştır.[22-31]



Şekil 2.24: a) Basit PAL yapısı(SPLD) **b)** Karmaşık PLD yapısı(CPLD)[21]

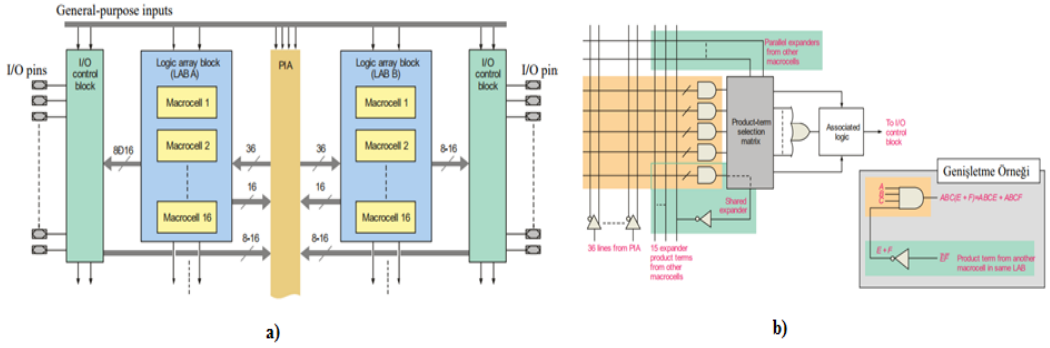
Üretici firmaya ve cihaz ailesine göre CPLD programlanır anahtarları EPROM, E^2PROM, FLASH veya SRAM hücre tekniğini kullanabilir.

CPLD'ler birden fazla PAL veya SPLD yapısının bir araya getirilerek oluşturulmuştur. CPLD içerisindeki bu PLD yapılarına LAB(logical array block) adı verilir ve CPLD içerisinde bulunan LAB'lar birbirleri ile haberleşecek şekilde iç bağlantılar ile birbirine bağlanmıştır. Bu iç bağlantılara PIA(programlanabilir ara bağlantı dizisi) adı verilir. Klasik CPLD LAB'ları içerisinde LUT (look up table) yapıları yerine VE/VEYA kapı dizileri bulunur. Klasik tip CPLD'lerin terkedilme sebebi ihtiyaç duyulan mantık hücresi sayısı artıkça, tek bir global ara bağlantı tipi olduğu için bağlantı sayısının katlanarak artmasıdır.[22-31]



Şekil 2.25: CPLD iç bağlantıları ve LAB'lar[22]

CPLD içyapısı içerdiği elemanlara göre değişmektedir. Aşağıdaki şekilde ALTERA firmasının MAX 7000 CPLD serisinin içyapısı ve macrocell'lerin yapısı verilmiştir.



Şekil 2.26: a) MAX 7000 CPLD içyapısı, **b)** MAX 7000 macrocell yapısı[22]

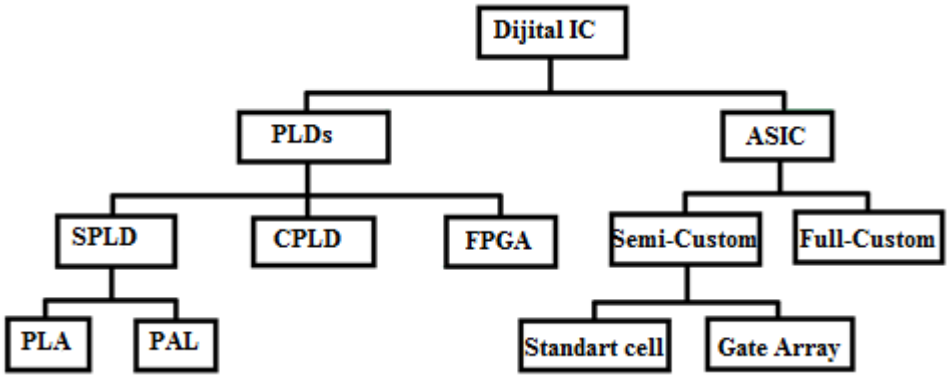
Tablo 2.4: Basit PLD'ler ve Karmaşık PLD'lerin karşılaştırılması

Kriter	**SPLD**	**CPLD**
Propagasyon Gecikmesi	Yüksek hız - Genellikle 3.5ns	Yüksek hız – Genellikle 10ns
Yoğunluk	100 - birkaç yüz arasında	Birkaç yüz -birkaç bin (25.000)
Teknoloji	Bipolar ve CMOS	CMOS
Tasarım Kolaylığı	Çok kolay	Orta kolaylık
Komplekslik	Basit mimari	Orta ve zor mimari
Frekans	200 MHz	100 MHz
Programlanabilir I/Os	Var	Var

2.2.2. ASIC(Application Specific Integrated Circuits)

ASIC'ler genel amaçlı tüm devrelerin aksine belirli bir kullanım amacına hizmet etmek için özel olarak üretilmiş silikon yongalardır. Seri üretimde lojistik ve malzeme masrafları düşüktür. Düşük güç tüketimi ve optimum performans sağlarlar. Tedarik zinciri yönetimi kolaydır çünkü tek bir şirket tasarımı yapar. Fikri mülkiyet korunur. Sistem boyutunu düşürür. FPGA'lere nazaran lojik fonksiyon başına transistor kullanımı daha azdır. Dezavantajları ise tasarım maliyetleri yüksektir. Farklı görevleri yerine getirmek üzere yazılımsal olarak programlanamazlar ve tasarım değişikliği yapılamaz. Tasarım araçları pahalıdır. Tasarım

planına ihtiyaç duyulur. Tasarım riski taşır, herhangi bir problem neticesinde revizyona ihtiyaç duyulabilmektedir. Üretimi aylar sürer.



Şekil 2.27: PLDs ve ASICs

ASIC kendi içerisinde dört ana başlıkta incelenir. Bunlar sırası ile Gate Array, Structured ASIC, Standart Cell ve Full Custom uygulamalarıdır.

2.2.2.1. Full Custom

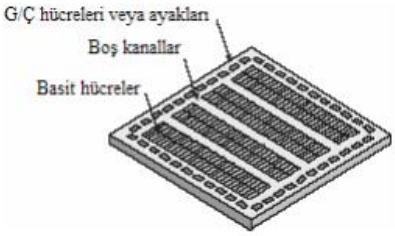
Tasarlanan devrelerin hızı ve karmaşıklığı artıkça tasarımcının üretim sürecine müdahalesi de artar. Yüksek hızlı sistemler ile gizliliği önemli olan devreler Full-Custom yöntemi ile tasarlanır. Bu yöntemde üreticinin gönderdiği kütüphaneler kullanılmaz. Üreticiden metal bağlantı yüzeyleri sayısı, iki kanal arası izin verilen en kısa mesafe gibi teknolojik bilgiler alınır. Üreticiye masklar dışında herhangi bir bilgi gönderilmediği ve üreticinin kütüphaneleri kullanmadığı için tasarımın çözülme olasılığı yoktur. Tasarımı ve üretimi, uzun süren bir yöntemdir. [23]

2.2.2.2. Standard Cell

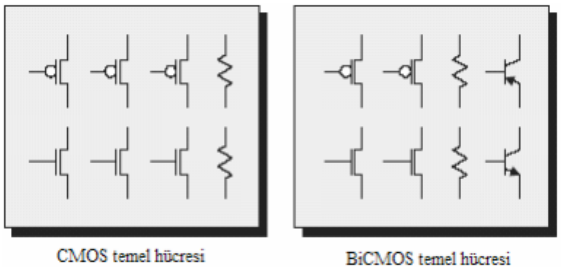
Bu yöntemde de üreticinin kütüphaneleri kullanmakla birlikte yonga üzerindeki hücrelerin konumu tasarımcı tarafından belirlenir. Gate array yöntemi gibi temel hücre yaklaşımı yoktur. Kütüphane içinde yer alan modeller üzerinde her hangi bir değişiklik yapılmamakla birlikte, yonga üzerinde oluşturulacak serimde tasarımcının söz sahibi olması, üretim sonuçlarının benzetim sonuçlarına daha yakın olmasını sağlar. Sistem gizliliği daha fazladır. Ancak yine de üreticinin modellerinin kullanılmış olması, çok zor olmakla birlikte, sistemin çözülme olasılığını gündeme getirebilmektedir. [23]

2.2.2.3. Gate Array

Bağlantısız transistör ve direnç içeren temel hücreler mantığına dayanır. Her ASIC sağlayıcısı özel birim hücrelerinde en uygun bileşen dağılımını belirler.



Şekil 2.28: Gate Array yaklaşımlı yonga görünümü[20]



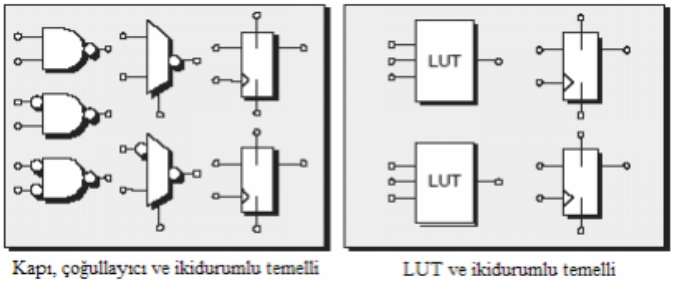
Şekil 2.29: Basit gate array temel hücreleri[20]

En kısa sürede sonuç alınabilen, tasarımı ve üretimi kolay olan ASIC tasarım ve üretim yöntemidir. Üreticiye masklarla birlikte devre şemaları da verildiği için tasarım gizliliği en az olan yöntemdir. Ancak tasarım ve üretim süreçlerinin kısalığı, ASIC tasarımına yeni başlayanlar için, sonuçları kısa bir süre içinde görmek açısından, bu yöntemi çekici bir duruma getirmektedir.

Olumsuz tarafları, çoğu tasarım önemli miktarda içsel kaynağı kullanmadan bırakır. Bağlantı düzenlemeleri kısıtlanır ve en iyi durum sağlanamaz. Bu da tasarımın başarımını ve güç tüketimini olumsuz etkiler.[23]

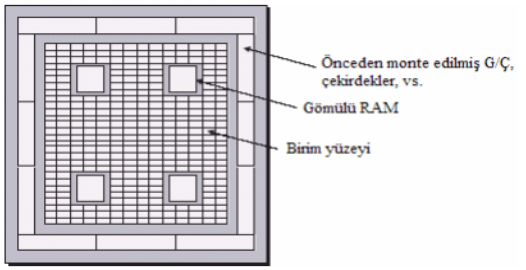
2.2.2.4. Structured ASIC

On yıllık geçmişine rağmen son iki-üç yıldır ASIC tasarım maliyeti ve geliştirme zaman değerlerini düşürme çabalarının sonucunda kullanılmaktadır. Her aygıtta “module” (birim) veya “tile” (kiremit) olarak isimlendirilen temel öğeler vardır. Aşağıdaki şekilde öğe içerikleri genel olarak verilmiştir.



Şekil 2.30: Yapısal ASIC temelleri[20]

Çoğu yapısal ASIC mimarisi için özelleşmiş iki veya üç metalleme katmanı yeterlidir. Bu sayede tasarımı oluşturma maliyeti ve zamanı önemli ölçüde azalır. Ancak yapısal ASIC, Standard Cell yöntemine göre aynı fonksiyonu gerçeklemek için silikon alanı ve güç tüketimi açılarından yaklaşık üç kat fazlasına gereksinim duyar.[23]



Şekil 2.31: Yapısal ASIC genel görünümü[20]

2.2.3. FPGA (Field Programmable Gate Arrays)

İlk FPGA 1984 yılında Xilinx firması tarafından geliştirilmiştir. Gelişimi 1980 yıllara gelinmeden önceki CPLD yapılarına dayanmaktadır. CPLD'lerden farklı bir mimarileri vardır ve CPLD'lerden çok daha esnektirler. CPLD içerisindeki lojik kapılar birkaç bin ila on binler ile sınırlı iken FPGA içerisindeki lojik kapılar on binlerden milyonlara ulaşmaktadır. CPLD'lerden bir diğer önemli farklı ise yapılarında gömülü bellek yapıları ve üst düzey gömülü fonksiyonlara (toplama, çıkarma gibi) sahip olmalarıdır.

Alanda programlanabilir kapı dizileri PLD kategorisinden alanda programlanabilir mantık devreleri grubundadır. Bazı kaynaklarda FPGA'ler UPASIC (user programmable ASIC) olarak gösterilir. FPGA'ler aslında PLD'ler ile ASIC'ler arasındaki teknoloji boşluğunu doldurmaktadır. PLD'ler programlanabilir olmalarına karşılık karmaşık sistemlerde kullanışlı değillerdir. ASIC'ler ise karmaşık sistemlerde yüksek performansla çalışmalarına karşın

programlanabilir değillerdir. Bu iki teknoloji arasında hem karmaşık sistemlerin tasarımına el verecek hem de kullanıcı tarafından programlanabilirken aynı zamanda tasarım değişikliğine müsaade edecek şekilde yeniden programlanabilecek sistem ihtiyacını FPGA teknolojisi karşılamaktadır.

2.2.3.1. FPGA Mimarisi

FPGA ve ASIC yapılarını karşılaştırdığımızda, ASIC ile yapılan tasarımı bir zincire dizilmiş farklı renk ve şekillerdeki boncukların bütünü olarak düşünebiliriz. Bu zincirdeki bir boncuğun yerini değiştirmek istediğimizde o boncuğa ulaşmak için zincirdeki tüm boncukları çıkarmamız gerekir. Bu da tasarımı baştan yapmamız anlamına gelir. FPGA ile yapılan tasarımları ise birbirine geçirilmiş legolara benzetebiliriz. Eğer bu lego bloğunda bir kısmı değiştirmek isterseniz. Birbirine geçirilmiş Legolardan sadece çıkarmak istediğiniz parçayı çıkarır geri kalan kısmı olduğu gibi tutabilirsiniz. Bu benzetimden yola çıkacak olursak FPGA ile ASIC arasındaki temel ve en önemli farkın tasarım değişimi ya da düzeltilmesindeki esneklik olduğunu görebiliriz. FPGA'ler sahada programlanabilir yapıları sayesinde ASIC'lere oranla çok daha esnektirler.

ASIC'ler FPGA'lerden çok daha hızlıdırlar, daha az alan kaplarlar ve daha az güç tüketirler. Buna karşılık olarak FPGA'ler ile akla gelebilecek her türlü dijital tasarım yapılabilir, yapılan tasarımlar değiştirilebilir. Start up maliyetleri çok daha düşüktür. Tasarım çok kısa sürede tamamlanır. Simülasyonu, analizi ve testi kolay ve ucuzdur.

FPGA'in bazı önemli özellikleri kullanımını popüler kılmıştır. Bu özelliklerin bazıları, paralel işlem yapma yeteneği, sahada programlanabilme, tasarımın test edilip doğrulanabilmesi ve FPGA'e işlemci gömülebilmesidir.

Paralel işlem yapma kabiliyeti: Aynı anda birden fazla işlem yapma anlamına gelen bu özellik, FPGA'in diğer entegrelere nazaran daha hızlı çalışmasında etkili olmuştur. Sıradan entegreler aynı anda birden fazla işlemi ya hiç yapamaz ya da sınırlı düzeyde yapabilir. FPGA'de uygulamaya ve kapasiteye bağlı olarak, birbirinden bağımsız onlarca belki yüzlerce işlem aynı anda gerçekleştirilebilir. Özellikle gerçek zamanlı ve yüksek çözünürlüklü bir video görüntüsü üzerinde filtreleme işlemini düşünürsek, video görüntüsü binlerce resmin bir araya gelmesinden oluşmuştur. Bu görüntünün filtrelenmesi her bir resmin tek tek alınması filtrelenmesi ve gönderilmesi gerekir. Bu işlem standart entegrelerde her resim için sırası ile

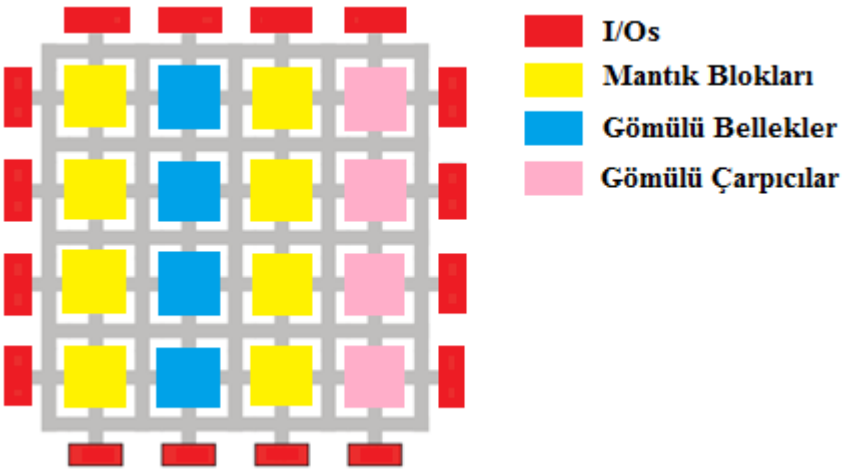
gerçekleşir. Yani bir resim alınıp filtrelenip gönderilmeden ikince resimde herhangi bir işlem başlamaz. FPGA'de ise birinci resim alındıktan sonra filtrelenme işlemine geçtiğinde aynı anda ikinci resim alınır. İlk resim gönderilmeye başlandığında ikinci resim filtrelenmeye ve üçüncü resimde alınmaya başlanır. Yani bir döngüde kaç işlem adımı var ise o kadar resimde farklı aşamalardaki işlemlerden geçer.

Sahada programlanabilme: En önemli özelliklerinden biri olan sahada programlanabilme özelliği ile FPGA çalıştığı ortamda programlanabilmektedir. FPGA'ler nihai ürün haline geldikten sonra da defalarca programlanabilmektedir. Bu özellik zaman ve maliyet açısından çok önemlidir.

Tasarımın test edilip doğrulanabilmesi: Tasarım FPGA'e yüklenmeden önce simülasyonu yapılıp problem tespiti yapılabilir. Bazı özel araçlar ile tasarım FPGA'e yüklendikten sonra da test edilip hata giderimi yapılabilir. Tabi bu özellik sadece FPGA'e has değildir.

FPGA'e işlemci gömülmesi: FPGA içerisine soft olarak işlemci gömülebilmektedir. Bu işlemciler C/C++ gibi diller ile programlanabilmektedirler. İşlemci gömüldükten sonra geriye kalan FPGA kısımları paralel işlemler için kullanılabilirler. Bu sayede hem sıralı işlemler için kolaylık sağlanır hem de FPGA'in paralel işlem yapma kabiliyetinden faydalanılır.

Modern FPGA yapılarında programlanabilen ara bağlantılar ve lojik bloklar, bellek yapıları, çoğullayıcılar ve giriş çıkış blokları bulunur.



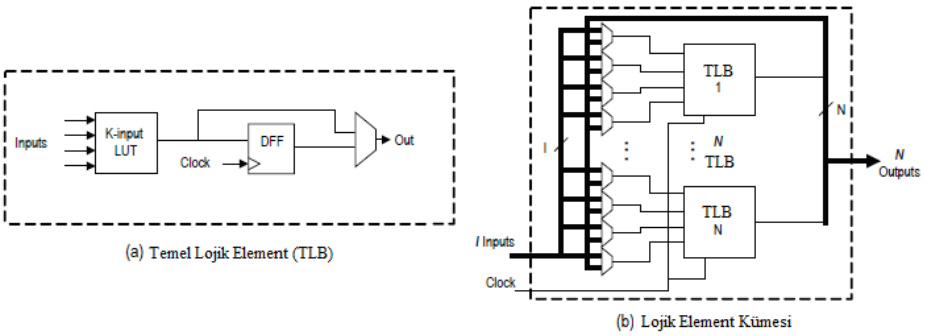
Şekil 2.32: Modern FPGA yapısı

FPGA'de "programlanabilir" terimi, silikon fabrikasyonu tamamlandıktan sonra FPGA yongası üzerine herhangi bir fonksiyonun yazılımsal olarak eklenmesini ifade eder.

2.2.3.2. Mantık Blokları

Temel FPGA'ler mantıksal fonksiyonları gerçekleştirmek üzere programlanabilen mantık bloklarından, bu mantık bloklarını birbirine bağlayan programlanabilir iç bağlantılardan ve çip dışı bağlantıları yapmak için I/O bloklarından oluşur.

FPGA'daki bir mantıksal bloğun amacı, dijital mantık sistemlerinde kullanılan temel hesaplama ve depolama öğelerini sağlamaktır. Orijinal kapı dizilerinde kullanıldığı gibi, bu kabiliyeti sağlamanın en basit ve özellikli olmayan yolu, bir temel transistorun temel mantık elemanı olarak kullanılması ve bunlardan kapı ve depolama elemanları inşa etmektir. Crosspoint firması bunu ticari bir FPGA yapısında kullanmıştır. Fakat bu çok ince taneli mantık bloğu, herhangi bir mantık fonksiyonun gerçekleştirmek için çok fazla programlanabilir iç bağlantıya ihtiyaç duymuştur. Bu bağlantılarında transistor kullanımı ile gerçekleştirildiği düşünülecek olursa bu yapının alan verimliliğinin ne kadar düşük olacağı tahmin edilebilir. Ayrıca bu yapıda güç tüketimi artmış ve performans düşmüştür. İnce taneli bu yapıların olumsuz etkileri sebebi ile daha sonraları geliştirilen FPGA mimarilerinde iri taneli mantık blokları kullanılmıştır. Bu iri taneli temel mantık bloklarında transistorlardan oluşmak yerine LUT(look-up table) denilen küçük bellek yapıları, NAND kapıları ve çoğullayıcı yapılarının birleşiminden oluşurlar.[24]



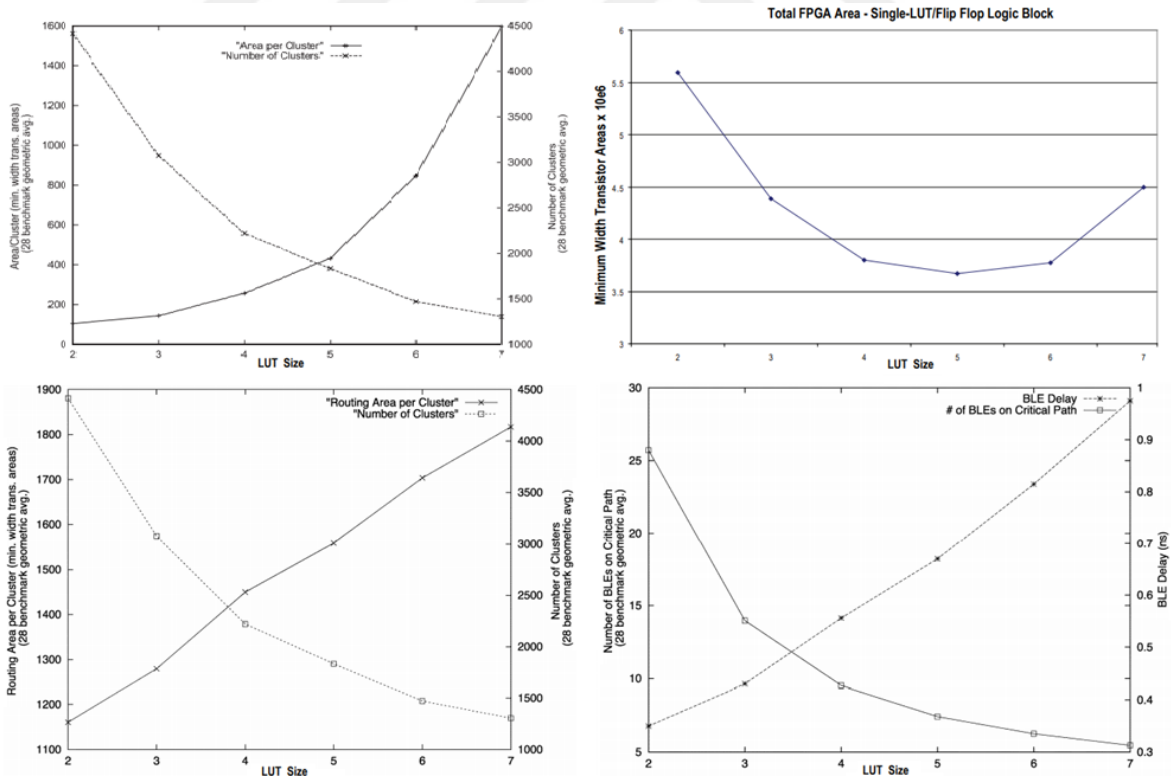
Şekil 2.33: FPGA Temel Lojik Elementi ve Kümesi[24]

Mantık bloklarının isimleri üretici firmalara göre değişiklik göstermektedir. Örneğin iki büyük FPGA üreticisinden biri olan Xilinx mantık bloklarını "konfigüre edilebilir lojik bloğu

(CLB)" olarak ifade ederken diğer büyük FPGA üreticisi olan Altera firması ise "lojik element(LE)" ifadesini kullanmaktadır.

Look-Up Tables: Bu yapılar temel olarak herhangi olası girişlerin oluşturacağı doğruluk tablosu çıkışlarını ifade eder. N girişli bir LUT yapısı girişlere bağlı olarak 2^N adet farklı durumu saklar. FPGA üzerinde yapılmak istenen boolean cebiri işlemleri bu yapılar ile gerçekleştirilir. Mantık blokları içerisindeki en önemli yapılardır.

İlk LUT yapısı 3 girişli olarak 1980'li yıllarda tasarlanmıştır. Günümüzde 3, 4, 5 ve 6 girişli LUT yapıları vardır fakat araştırmalar göstermiştir ki klasik LUT yapılarında giriş sayısı arttıkça FPGA işlem kapasitesi ve lojik blok başına alan artıp lojik blok kümelerinin sayısı azalırken performans düşmekte ve gecikmeler artmaktadır. Bunun en önemli sebebi LUT girişi arttıkça katlanarak artan iç bağlantılardır. Bu sebeple optimize ve yüksek performanslı LUT yapıları 4 girişli yapılardır. Aşağıda LUT yapılarında giriş sayısına bağlı olarak performans ve gecikme değişimleri verilmiştir. [25-28]

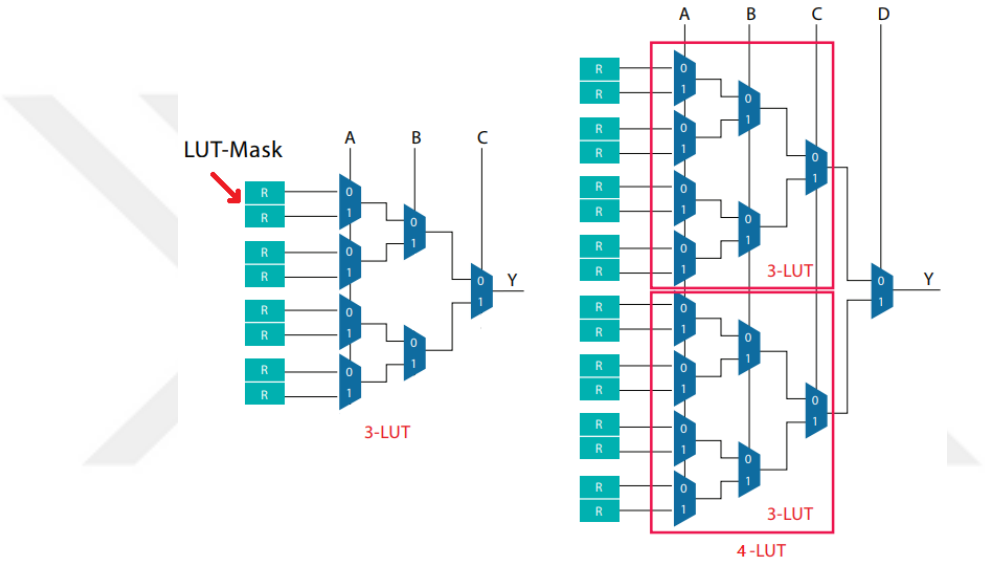


Şekil 2.34: LUT giriş sayısına göre alan, routing alanı, fpga alanı ve gecikme değişimi[25]

Klasik LUT yapılarının aksine büyük girişli LUT yapıları daha küçük girişli LUT yapılarından oluşturulduğunda alan kullanımında iyileşmeler gözlenmiştir. Örneğin 4-

LUT'lar 3-LUT'lardan, 5-LUT'lar 4-LUT'lar'dan oluşturulabilir. Bu teknik kullanılarak daha büyük girişli LUT'lar daha küçük girişli LUT'lardan oluşturulabilir.

LUT yapıları konfigürasyon belleklerini(CRAM) tutan SRAM bitlerinden ve çoğullayıcı setinden oluşur. Bu çoğullayıcılar çıkışa ulaşacak CRAM bitini seçer. K girişli bir LUT yapısı oluşturmak için 2^k SRAM biti ve 1 adet 2^k:1 çoğullayıcı yapısına ihtiyaç duyulur. Oluşturulan bu LUT yapısı k girişlerinin oluşturduğu tüm kombinasyonları (2^k) saklar. Aşağıda 3 girişli LUT yapısı ve 3 girişli LUT yapısından oluşturulmuş 4 girişli LUT yapısı verilmiştir.[28]



Şekil 2.35: 3-LUT ve 4-LUT yapısı[28]

4 girişli LUT yapısı 16 (2^4) bit SRAM hücresi içerir ve 2:1 çoğullayıcı yapılarının birleşiminden oluşan 16:1 çoğullayıcı yapısından oluşur. LUT maskesindeki uygun değer ayarlandığında 4-LUT yapısı A, B, C, ve D girişlerinin herhangi bir fonksiyonu uygulayabilir.

2.2.3.3. Giriş/Çıkış Blokları

FPGA içerisindeki mantık blokları ile FPGA pinleri arasındaki bağlantılar programlanabilir I/O blokları ile sağlanmaktadır. Bu bloklar FPGA'in dış birimler ile bağlantıların sağlandığı yapılardır. Tasarıma göre sadece giriş, sadece çıkış ya da hem giriş hem çıkış olarak programlanabilirler. [4]

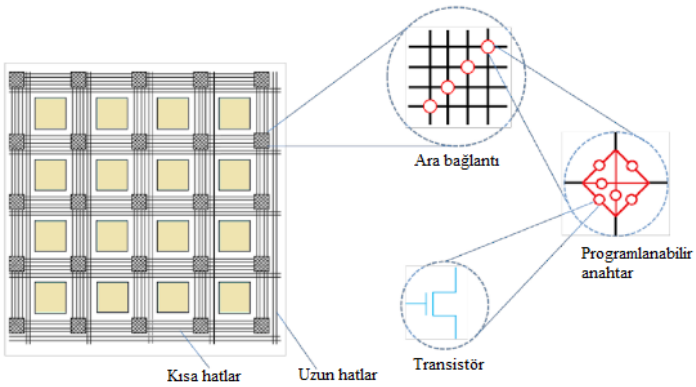
2.2.3.4. Alıcı/Verici Blokları

FPGA'lerin dış dünya ile iletişimini sağlayan bir diğer yapı da alıcı/verici bloklarıdır. Parametreleri kullanıcı tarafından yapılandırılabilir. Özellikle üretilen bu bloklar sayesinde, farklı FPGA'ler veya sistemlerle yapılan gigabitler seviyesindeki yüksek hızlı veri alış verişi sağlanır. İçerisinde seri-paralel ya da paralel-seri dönüştürücüler DSP, PLL ve FIFO blokları ile saat üreteçleri bulunur. Tek yönlü ya da çok yönlü kullanılabilen bu alıcı/verici birimleri; PCI Express, Ethernet, fiber optik gibi ara yüzler ile veri alış verişini destekler. [4]

2.2.3.5. Programlanabilir Ara Bağlantılar

Mantık hücrelerinin ara bağlantıları matris şeklinde veri yolları ve programlanabilir anahtarlarla sağlanır. Bu anahtarlama işlemi FPGA içerisine yüklenen program ile yapılır.

FPGA tasarımları ile FPGA içerisinde bulunan her bir mantık hücresinin uygulayacağı fonksiyon ile programlanabilir anahtarların durumu (açık/kapalı) belirlenir. Programlanabilir anahtarlar sayesinde mantık hücrelerinin birbirleri ve Giriş/Çıkış'lar arasındaki sinyal iletimi sağlanmış olur. Programlama işleminde FPGA içerisindeki kısa hatlar iç bağlantılar için, uzun hatlar ise global set/reset ya da saat sinyalleri için kullanılır. Tasarımda kullanılan yazılımlar tüm bu bağlantıları en verimli ve en basit şekilde yapar. [4]



Şekil 2.36: Programlanabilir ara bağlantılar[4]

Anahtarlama matrisinde bulunan bağlantıların kesişim noktasında 6 adet transistörle hangi hatların birbirlerine bağlanacağı belirlenir.

2.2.3.6. RAM Blokları

Günümüzde FPGA'lerin hepsinde ayrılmış bellek birimleri bulunmaktadır. Tekli veya çoklu erişimi destekleyen bu dahili bellek birimleri, tasarımcının mantık devrelerinin işleyişi sırasında karşılaşacağı geçici depolama ihtiyacı için kullanılır.

Çoklu erişim sayesinde birden fazla uygulama bellek üzerine okuma/yazma yapabilir. Çoklu erişim farklı saatlerle çalışan işlem blokları arasında veri aktarımı için iyi bir çözümdür. Örneğin 25 MHz saat ile çalışan bir veri toplama ünitesinden 50 MHz ile çalışan veri işleme ünitesine veri aktarmak için 2 portlu bir bellek kullanabiliriz. 25 MHz ile çalışan veri toplama ünitesi veriyi belleğe yazar ve 50 MHz de çalışan veri işleme ünitesi veriyi bellekten okuyarak kullanır.

FPGA içerisinde büyük bellek ihtiyaçları için Blok RAM'ler bulunurken küçük veriler için mantık hücreleri arasına serpiştirilmiş dağınık küçük bellekler bulunmaktadır. Xilinx ihtiyaca göre mantık hücrelerinin bazılarını dağınık bellek olarak kullanırken; Altera blok RAM'leri FPGA içerisinde değişik boyutlarda paylaştırmıştır[4].

2.2.3.7. FPGA Pinleri

Bir tasarımda kullanılan FPGA'in diğer çevre elemanları ile bağlantılarının sağlanması üzerindeki pinler aracılığı ile olur. Bu pinler genel olarak "ayrılmış pinler" ve "kullanıcı pinleri" olarak ikiye ayrılır. Tasarım sırasında yapılan pin atama işlemi ile tasarımda oluşturulan portlar FPGA pinlerine bağlanır. Yapılan pin ataması, derleyici program tarafından özel bir formattaki metin dosyasına yazılarak bilgisayara kaydedilir. Sinyallerin yönünün bulunmadığı bu dosyada sinyallerin isimleri ve bağlanacakları pinler yazılıdır. Sinyallerin yönü tasarımın girişinde belirtilir. [4]

Ayrılmış Pinler: FPGA'de tüm pinlerin %20 ila %30'u FPGA'de gerçekleştirilne özel fonksiyonlara göre ayrılmış pinlerdir. Ayrılmış pinler "güç", "konfigürasyon" ve "saat" olarak sınıflandırmak mümkündür.

- Güç Pinleri: FPGA için gerekli olan güç ve toprak bağlantısını sağlayan bağlantı noktalarıdır.

- Konfigürasyon Pinleri: Oluşturulan programın FPGA'e yüklenmesi için kullanılan bağlantı noktalarıdır.
- Saat Pinleri: Saat sinyallerinin bağlanabilmesi için ayrılmış bağlantı noktalarıdır.

Kullanıcı Pinleri: Kullanıcı tarafından yapılandırılabilen standart giriş çıkış pinleridir. Giriş, Çıkış ve Giriş/Çıkış olmak üzere üç türü vardır. Her bir giriş/çıkış pini FPGA'de bir G/Ç hücresine(bloğuna) bağlıdır. G/Ç hücrelerinin güçleri VCCIO pini tarafından sağlanır.

Eski FPGA'ler birden fazla VCCIO pinine sahip olmalarına rağmen, bütün pinleri aynı voltajla beslenirdi. Yeni üretilen FPGA'lerde ise G/Ç'ler gruplara ayrılabilir ve bu gruplar farklı voltajlardan beslenebilirler. Örneğin bir grup G/Ç pinleri 3.3 V ile çalışırken; diğer grup G/Ç pinleri 2.5 volt ile çalışabilmektedir.[4]

2.2.3.8. Global İç Bağlantılar ve Saat Pinleri

Senkron tasarımlarda FPGA içindeki flip flopların tek bir saat sinyali ile aynı anda tetiklenmesi gerekir. Aksi taktirde tasarımda zamansal problemler meydana gelir.

FPGA'lerde global hat olarak adlandırılan özel iç bağlantılar bulunur. Bu bağlantılar sayesinde saat sinyalinin FPGA içerisindeki bütün flip floplara aynı anda ulaşması sağlanır.

Doğru bir senkron tasarım için saat beslemelerinin FPGA'in saat için ayrılmış bu pinlerinden verilmesi gerekir.[4]

2.2.4. FPGA Programlama Teknolojileri

FPGA'in iç bağlantılarının yapılması programlanması ile mümkündür. Programlanabilir anahtarlama teknolojisi üreticiden üreticiye değişmekle birlikte, hepsinde kullanılan ortak bir yöntem vardır. Bu da anahtarın açık ya da kapalı konumlarından bir tanesinin seçilmesi işlemidir.

Mümkün olduğunca en küçük alanın kullanılması, düşük rezistans, düşük parazit kapasitans ve büyük çaplı üretimler için güvenirlik gibi hususlar, programlama işleminde kullanılan teknolojide belirleyici unsurlardır.

Mantık hücrelerinin ara bağlantılarının yapılmasını sağlayan beş farklı FPGA programlama teknolojisi vardır.[4]

- SRAM tabanlı
- SRAM/Flash tabanlı
- Anti-fuse tabanlı
- EPROM/EEPROM tabanlı
- Flash tabanlı

Tablo 2.5: FPGA programlama teknolojileri karşılaştırması[4]

Özellik	**SRAM**	**Anti-Fuse**	**EEPROM**
Harici konfigürasyon dosyası ihtiyacı	Var	Yok	Yok
Uçucu olma	Var	Yok	Yok
Programlama hızı	Düşük	Yüksek	Orta
Tekrar programlanabilme	Var	Yok	Var
Kapladığı alan	Fazla	Az	Orta
Sistem üzerinde programlanabilme	Var	Yok	Var
Güvenlik	Zayıf	İyi	İyi
Güç tüketimi	Orta	Düşük	Orta
Elektromanyetik alandan etkilenme	Yüksek	Düşük	Yüksek

2.2.4.1. SRAM Tabanlı Programlama Teknolojisi

SRAM tabanlı çalışmada geçiş transistorlarının ya da mantık kapılarının kontrol edilmesinde küçük statik bellekler kullanılır. Güç kesilene kadar ya da cihaz yeniden programlanana kadar bellek içerisindeki veriler sabit kalmasından dolayı bu tip bellekler statik belleklerdir. Dinamik bellekler gibi sürekli tazelenmelerine gerek yoktur. Her bir mantık bloğunun konfigürasyonunun kaydedildiği SRAM bitleri FPGA içerisindeki konfigürasyon hücrelerine bağlıdır. Bu yöntemle konfigürasyon fonksiyonunun doğruluk tablosu da SRAM bitlerinde kaydedilmiş olur. Ayrıca bu bitler MUX yapılarına da bağlı durumdadırlar.

SRAM hücreleri aynı zamanda geçiş transistorlarının ya da mantık kapılarının kontrol edilmesinde de kullanılırlar. Yani SRAM hücresinde lojik-1 varsa bağlantı sağlanır lojik-0 varsa bağlantı koparılır.

SRAM'ler tekrar programlanabilir belleklerdir. Bu özellikleri sayesinde SRAM tabanlı FPGA'lar tekrar tekrar programlanabilme özelliğine sahiptir.

SRAM belleklerin en önemli dezavantajları ise uçucu(Volatile) olmalarıdır. Bu nedenle harici güç kaynağı kesildiğinde içerisindeki kayıtlı veriler ve programlar silinir. Bu yapıdaki FPGA'ların her açılışta tekrar programlanmasını sağlayan konfigürasyon mekanizmaları da FPGA üreticileri tarafından geliştirilir. İşte bu durum FPGA'ların CPLD'lere göre tek dezavantajını oluşturur çünkü her açılışta konfigürasyon belleğindeki program verilerinin FPGA üzerindeki SRAM hücrelerine aktarılması belli bir zaman kaybı oluşturur.[4]

2.2.4.2. SRAM/Flash Tabanlı Programlama Teknolojisi

SRAM tabanlı çalışmaya yakındır. Tek farkı dahili flash bellek içermesidir. Harici konfigürasyona ihtiyaç ortadan kaldırılmıştır. Konfigürasyon sırasındaki bit akışının kopyalanmasının önüne geçildiği için güvenli bir mekanizmadır.[4]

2.2.4.3. Anti-fuse Tabanlı Programlama Teknolojisi

Bir kez programlama ve sürekli kalıcılığı sağlayan bir çalışma yöntemidir. Sigortaların aksine mikroskobik yapıdaki, normal durumda yüksek empedanslı olan anti-fuse yapısına, yüksek voltaj uygulanması sonrası iletime geçirilen daha doğrusu yakılan yalıtkan ile sağlanır. SRAM'lere göre daha hızlıdırlar ve kalıcıdır(Nonvolatile).



Şekil 2.37: Sırası ile programlama öncesi ve sonrası Anti-Fuse yapısı[4]

Özellikle güçlü elektromanyetik alanlarda olan ancak bu etkilerden etkilenmesi istenmeyen cihazlarda ve askeri teknolojilerde tercih edilmektedirler.

2.2.4.4. EPROM ve EEPROM Tabanlı Programlama Teknolojisi

SRAM tabanlı FPGA'ların her açılışta yeniden programlanması için eskiden ROM yapıları kullanılmaktaydı. Ancak günümüzde içinde flash, EPROM ve EEPROM bellekleri bulunduran FPGA'lar üretilmektedir. EPROM yapılı FPGA'larda programlama için ayrıca bir depolama mekanizmasına gerek yoktur ancak, yeniden programlanması çok zahmetlidir. EPROM'un sökülüp ultraviyole ile silinmesi ve tekrar geri takılması gerekir. EEPROM yapılı FPGA'ları programlamada ise EPROM gibi zahmetli işlemler yoktur. Elektriksel olarak silinip tekrar programlanabilir. Ancak dezavantaj olarak EPROM hücresinden daha fazla yer kaplar ve çoklu voltaj kaynağına ihtiyaç duyar. Bu yapıların uçucu olmayan özelliklerine rağmen günümüzde SRAM tabanlı FPGA'lar daha çok kullanılırlar.[4]

2.2.4.5. Flash Tabanlı Programlama Teknolojisi

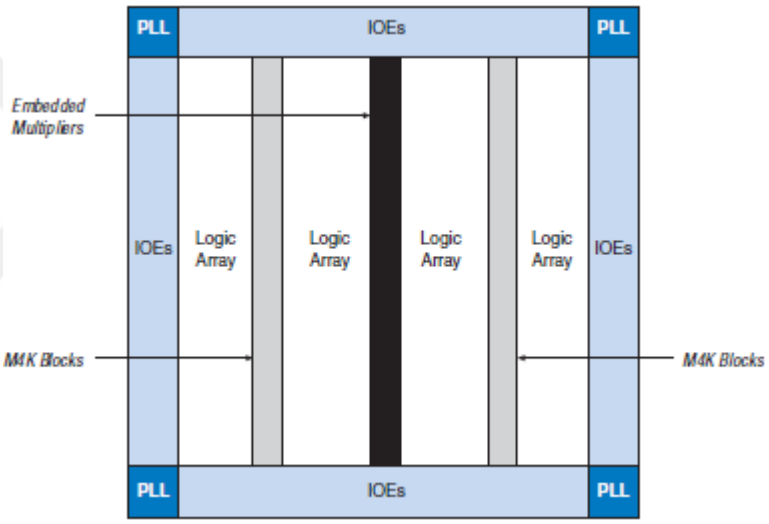
Bu çalışmada konfigürasyon verisi flash bellek içerisinde saklanır. SRAM/flash yönteminden farklı olarak herhangi bir SRAM'a ihtiyaç yoktur. Güç tüketimleri ve elektromanyetik ortamdan etkilenme oranları da düşüktür ve ayrıca veri kopyalamaya karşı güvenlidir.[4]

3. MALZEME VE YÖNTEM

3.1. ALTERA CYCLONE® II MİMARİSİ

Xilinx, Altera, Actel, Lattice Semiconductor QuiclLogic ve SiliconBlue yaygın FPGA üreticileridir. Bunlardan Xilinx ve Altera firmaları pazarın yaklaşık %90'ını elinde bulundurmaktadır.

Bu çalışmada Altera firmasının üretimini gerçekleştirdiği Cyclone II mimarili FPGA entegresi kullanılmıştır.



Şekil 3.1: Cyclone II mimarili EP2C20 etegresi blok diyagramı[26-27]

Cyclone II ailesine ait FPGA'ler yapısal olarak aynı olmalarına karşılık pin sayıları, lojik element sayıları, çalışma frekansı gibi niceliksel özellikleri farklılık göstermektedir.

Cyclone II cihazları iki boyutlu satır ve sütundan oluşan mimarileri ile özel lojik yapıların uygulamasına olanak sağlamaktadır. Farklı hızlardaki satır ve sütun iç bağlantılar, mantık dizi blokları (LABs), gömülü bellek yapıları ve gömülü çoğullayıcılar arasındaki sinyal bağlantısını sağlarlar.

LAB yapıları FPGA entegresi içerisinde satır ve sütunlar halinde gruplandırılmıştır. Her bir LAB yapısı 16 lojik elementten (LE) oluşmaktadır. LE yapısı, kullanıcı mantık

fonksiyonlarının verimli bir şekilde uygulanmasını sağlayan küçük bir mantık birimidir. Cyclone II cihazları 4,608 ila 68,416 arasında LE yoğunluğa sahiptir.

Cyclone II cihazları kullanıcılara, bir global saat ağı ve dört adete kadar faz kilitlemeli döngüler (PLL'ler) sağlar. Global saat ağı, tüm cihaz boyunca 16 adete kadar global saat hattından oluşur. Global saat ağı, giriş / çıkış elemanları (IOE'ler), LE'ler, gömülü çarpanlar ve gömülü bellek blokları gibi aygıt içindeki tüm kaynakların saatlerini karşılayabilirler. Cyclone II PLL'ler, saat sentezi ve genel amaçlı saat ayarı sağlar.

M4K bellek blokları 4K bit bellek artı parite (4,608 bit) ile doğru dual-port bellek bloklarıdır. Bu bloklar çalışma frekansı max 260Mhz seviyesinde 36 bit'e kadar ayrılmış doğru dual-port, basit dual-port ya da single-port belleği sağlar ve belirli LAB'lar arasında entegre boyunca uzanan sütunlar halinde düzenlenmiştir. Cyclone II cihazları 119 ila 1,152 Kbit aralığında gömülü bellek sağlar.

Her gömülü çarpıcı bloğu, iki adet 9 × 9 bitlik 250-MHz performansa sahip çarpıcıya ya da en çok bir adet 18 × 18 bitlik 250-MHz performansa sahip çarpıcıya sahiptir. Gömülü çarpıcılar, entegre boyunca uzanan bir sütun halinde düzenlenmiştir.

Her bir Cyclone II cihazı I / O pini, LAB sıralarının uçlarında ve cihazın çevresindeki kolonlarda bulunan bir giriş çıkış elemanı(IOE) tarafından beslenir.

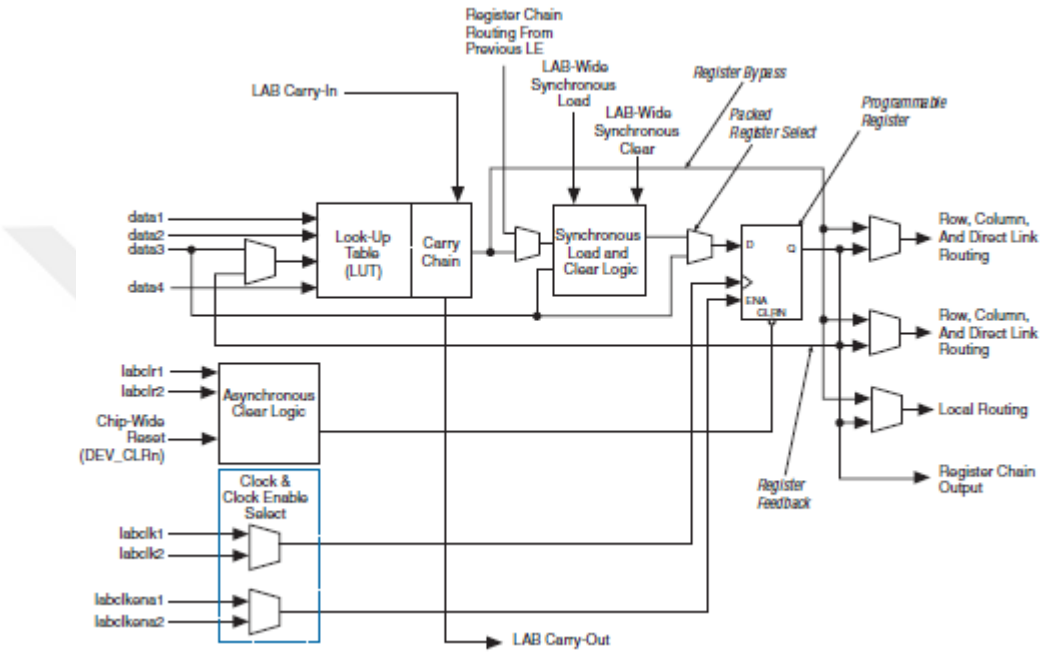
Bellek bloklarının sayısı, gömülü çarpıcı bloklarının sayısı, PLL'lerin sayısı, satır ve sütun sayısı her Cyclone II cihazında farklılık gösterir.[26-27]

3.1.1. Lojik Element Yapıları

Geldiğimiz noktaya kadar değerlendirdiğimizde FPGA yapılarında en önemli kısımların yapı içerisindeki mantık blokları olduğunu söyleyebiliriz. Altera firması bu mantık bloklarını lojik element olarak(LE) ifade eder. Cyclone II mimarisinde LE yapılarının sahip olduğu özellikler aşağıdaki gibidir.

- 4 değişkenli herhangi bir işlevi uygulayabilecek bir fonksiyon jeneratörü olan 4 girişli bir Look-Up Table(LUT) yapısı
- Programlanabilen bir register
- Bir carry zinciri bağlantısı

- Bir register zinciri bağlantısı
- Yerel, satır, sütun, register chain ve doğrudan ara bağlantı gibi her türlü ara bağlantıyı kullanma yeteneği
- Register packing desteği
- Register geri besleme desteği



Şekil 3.2: Cyclone II Lojik Element Yapısı[26-27]

Lojik elementlerdeki her programlanabilir register yapısı D, T, JK ve SR flip- flop operasyonları şeklinde yapılandırılabilir. Bu yapıların her birinde data(D), clock, enable clock(ENA) ve girişlerin yenilemesinde kullanılan clear(CLRN) girişleri bulunur. Global saat ağını, genel amaçlı I /O pimlerini veya herhangi bir dahili mantığı kullanan sinyaller, register yapısının clock ve clear kontrol sinyallerini sürebilir. Enable clock girişini ise genel amaçlı I / O pinleri veya dahili mantık sürebilir. Kombinasyonel fonksiyonlar için ise LUT çıkışları register yapısını atlar ve direk olarak LE çıkışını sürer.

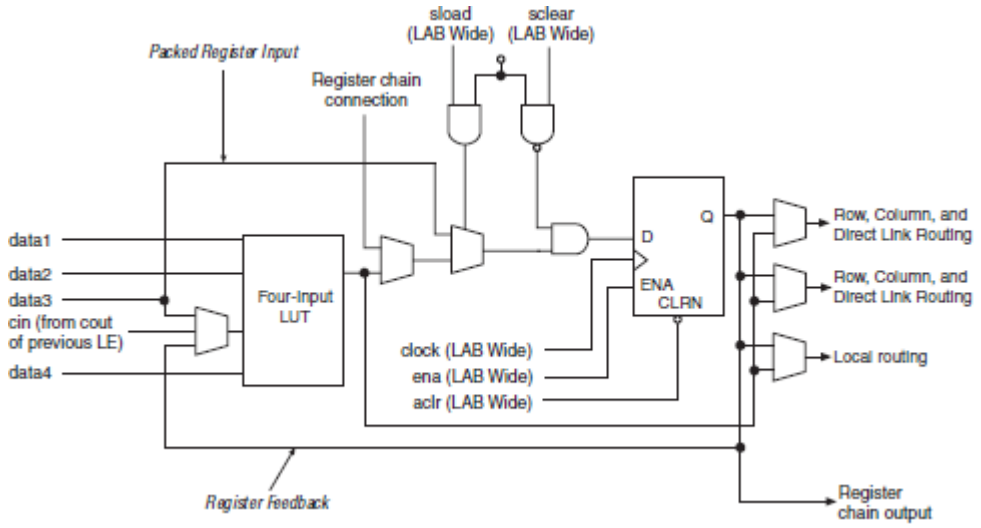
Her LE, yerel satır ve sütun yönlendirme kaynaklarını süren üç çıkışa sahiptir. LUT veya register çıkışı bu üç çıkışı bağımsız olarak sürebilir. İki LUT çıkışı satır, sütun ve direk link yönlendirmeyi(direct link routing) sürerken üçüncü çıkış ise yerel yönlendirmeyi(local routing) sürer. Bu üç çıkış register tarafından da sürülebilir ve cihaz LUT ve register yapısını birbiri ile ilişkili olmayan fonksiyonlar için kullanabilir.

Bir lojik dizi bloğu(LAB) içindeki LE yapısı üç genel yönlendirme çıkışına ilave olarak register zincir(register chain) çıkışlarına da sahiptir. Register zinciri çıktıları, aynı LAB içindeki register yapılarının birlikte kaskadlamasına izin verir.

Cyclone II mimarisindeki LE yapılarının iki farklı çalışma modu vardır. Bunlar normal mod ve aritmetik mod olarak adlandırılır. Her mod LE yapılarını farklı şekilde kullanır. Her bir modda LE yapısına giren, dördü, LAB yerel ara bağlantılarından gelen data girişi; biri, bir önceki LAB carry zincirinden gelen LAB carry girişi ve biri de register zinciri bağlantısından olmak üzere toplam altı giriş vardır. Bu altı giriş istenen lojik fonksiyonları gerçekleştirmek üzere farklı hedeflerde kullanılmıştır. LAB-wide sinyaller register yapıları için clock, asenkron clear, senkron clear, senkron load ve clock enable kontrol girişlerini sağlar. Bu LAB-wide sinyaller iki LE çalışma modunda da mevcuttur.

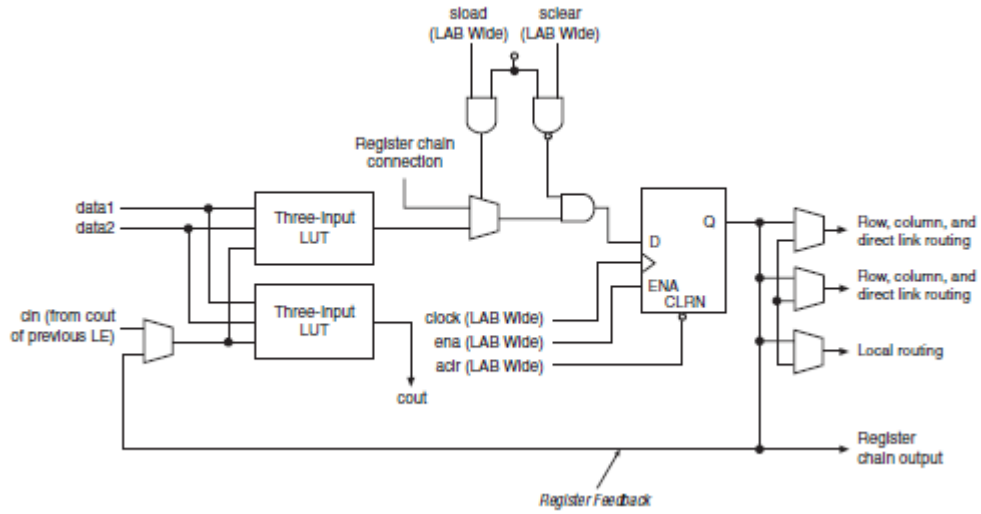
Quartus® II yazılımı, parametreli modül fonksiyonlar kütüphanesi (LPM) kullanarak sayıcı, toplayıcı, çıkarıcı ve aritmetik fonksiyonlar gibi yaygın fonksiyonlar için uygun çalışma modunu otomatik olarak seçer.[26-27]

Normal mod: bu mod genel lojik uygulamaları ve kombinasyonel lojik için uygundur. Normal modda LAB yerel ara bağlantılardan gelen 4 giriş sinyali 4 girişli LUT yapısının girişlerine uygulanmıştır. Quartus® II derleyicisi, carry girişi ya da data3 sinyalinden birini LUT yapısının girişlerinden biri olacak şekilde otomatik olarak seçer. Normal modda çalışan LE yapıları paket register'ları ve register geri beslemesini destekler. Aşağıda normal modda çalışan LE yapısı verilmiştir.



Şekil 3.3: Normal modda LE yapısı[26-27]

Aritmetik mod: Aritmetik mod, toplayıcılar, sayaçlar, akümülatörler ve karşılaştırıcı yapıları için idealdir. Aritmetik moddaki LE'ler, LUT çıktısının kayıtlı ve kayıt dışı versiyonlarını sürebilir. Aritmetik moddaki LE yapıları da paket register'ları ve register geri beslemesini destekler. Aşağıda aritmetik moddaki LE yapısı verilmiştir.[26-27]

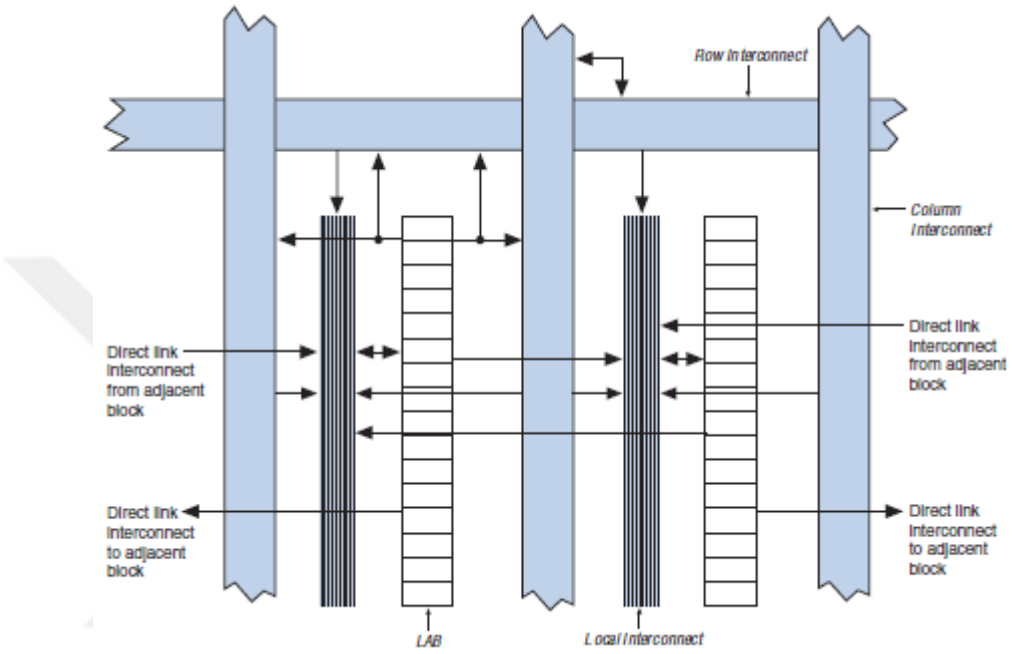


Şekil 3.4: Aritmetik modda LE yapısı[26-27]

3.1.2. Lojik Dizi Blokları(LAB)

Lojik dizi blokları(LAB) temelde LE yapırının bir araya gelmesinden oluşmuştur. Cyclone II mimarisinde bir LAB yapısı aşağıdakilerden oluşturmaktadır.

- 16 adet lojik element(LE)
- LAB kontrol sinyalleri
- LE carry zinciri
- Register zinciri
- Yerel ara bağlantılar



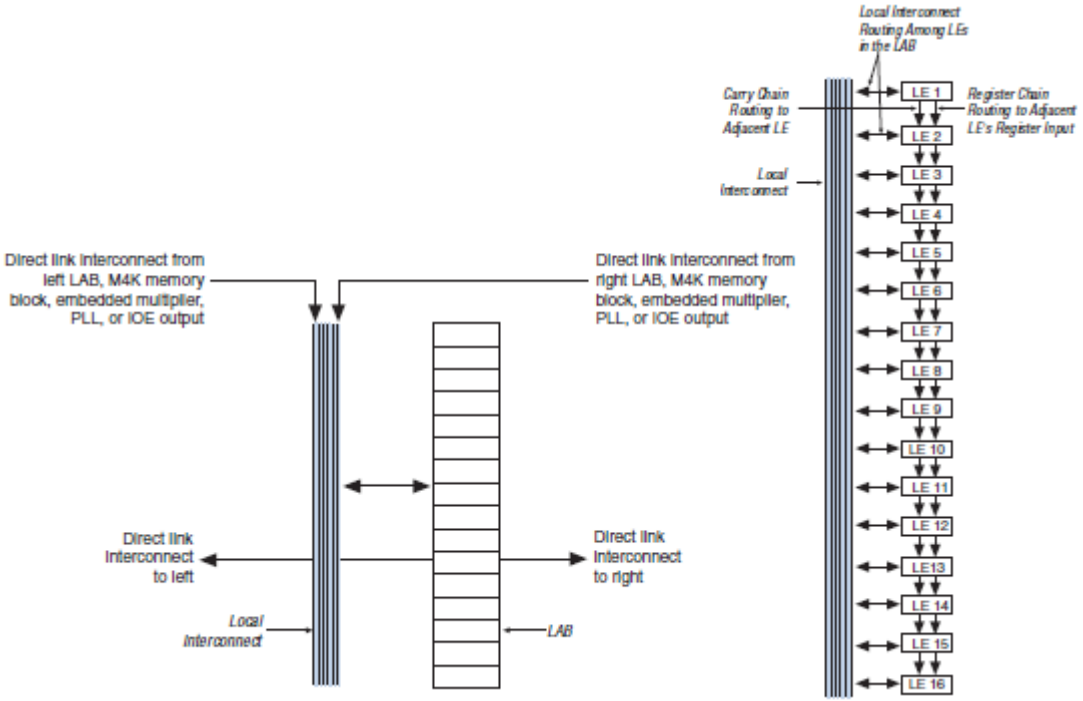
Şekil 3.5: Cyclone II mimarisinde LAB yapısı[26-27]

Cyclone II mimarisindeki satır ve sütun ara bağlantıları MultiTrack Bağlantı olarak adlandırılır. MultiTrack bağlantıda satır bağlantıları direkt link, R4 ve R24 ile temsil edilir. Sütun bağlantıları ise register zinciri, C4 ve C16 ile temsil edilmiştir. Bağlantılar arası mesafeler sabittir.

Yerel ara bağlantılar aynı LAB içerisindeki LE yapıları arasında sinyal iletimini gerçekleştirir. Register zinciri ara bağlantıları ise LAB içerisindeki bir LE register'ın çıktısını en yakın diğer LE register'ına taşır. Quartus® II derleyicisi bu ara bağlantıların kullanımını düzenlemektedir.

LAB ara bağlantıları aynı LAB içerisindeki tüm LE yapılarını sürerken LAB ara bağlantıları ise bu LE yapılarının çıktıları ve satır, sütun ara bağlantıları tarafından sürülmektedir. Ayrıca komşu LAB yapıları, PLL'ler, M4K RAM blokları ve sağ ve soldaki gömülü çarpıcılar da LAB yapısının yerel ara bağlantılarını direkt link bağlantısı üzerinden sürebilir. Direkt link bağlantıları satır ve sütun ara bağlantılarının kullanımını minimize ederek yüksek performans

ve esneklik sağlamaktadır. Her bir LE yapısı, hızlı yerel ve direkt link ara bağlantıları üzerinden 48 LE yapısını sürebilir.[26-27]



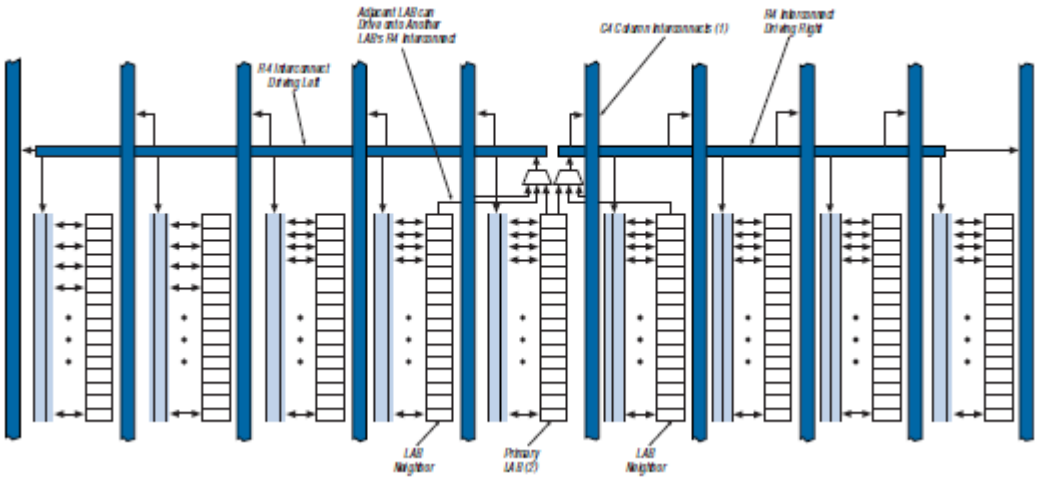
Şekil 3.6: Direkt link ve register zinciri bağlantısı[26-27]

Satır Ara Bağlantıları: Ayrılmış satır bağlantıları aynı satırda bulunan LAB, PLL ve M4K bellek bloklarından gelen ya da bu bloklara giden sinyallerin yönlendirilmesini sağlar. Bu satır bağlantı kaynakları aşağıdaki ara bağlantıları kapsar.

- LAB yapıları ve komşu bloklar arasındaki direkt link ara bağlantıları
- Bulunduğu noktadan 4 blok sağa veya sola sinyal geçişini sağlayan R4 ara bağlantıları
- Sağdan sola cihazın uzunluğu boyunca yüksek hızlı yatay sinyal yönlendirmesini sağlayan R24 ara bağlantıları

Direkt link ara bağlantısı, bir LAB, M4K bellek bloğu veya gömülü çarpıcı bloğunun, sağ ve sol komşularının yerel ara bağlantısını sürmesini sağlar. PLL bloklarının sadece bir tarafı direkt link ve satır ara bağlantıları ile ilişki kurar. Direkt link LAB'lar arasında ya da komşu bloklar arasında, satır ara bağlantı kaynaklarını kullanmadan, hızlı haberleşmeyi sağlar.

R4 ara bağlantısı her bir dörtlü LAB grubundaki satırlar arasında hızlı satır bağlantısı için kullanılır. LAB'lar bellek blokları, çarpıcı blokları PLL'ler ve IO elementleri tarafından sürülebilirler ve bu bağlantılar R24, C4 ve C16 ara bağlantılarını sürebilirler.[26-27]



Şekil 3.7: R4 satır ara bağlantıları[26-27]

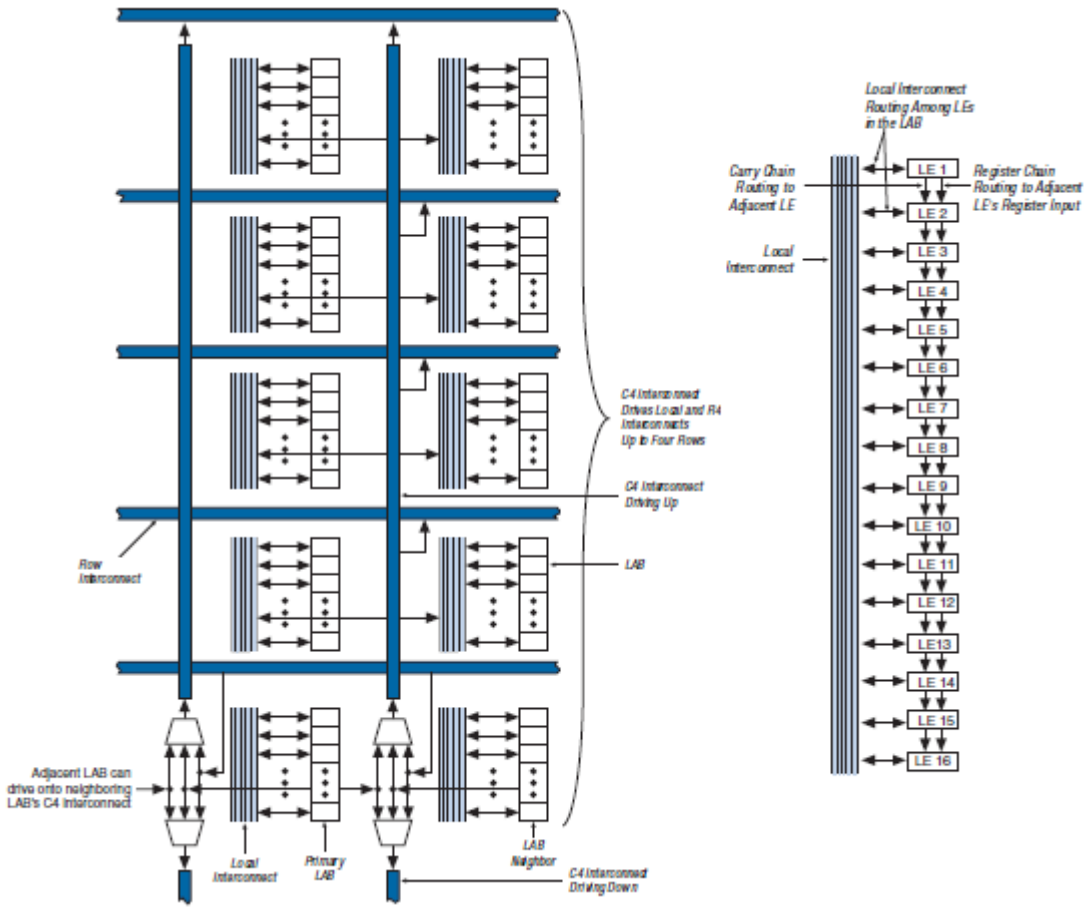
***Sütun Ara Bağlantıları*:** Sütun ara bağlantıların çalışması satır ara bağlantılar ile benzerlik göstermektedir. Her bir LAB sütununa ayrılmış bir sütun ara bağlantısı hizmet verir. Bu ara bağlantılar ile sinyal yönlendirmesi LAB'lar, M4K bellek blokları, gömülü çarpıcılar satır ve sütun IO elementleri arasında dikey olarak gerçekleştirilir. Satır bağlantı kaynakları aşağıdaki ara bağlantıları içermektedir.

- Bir LAB içerisindeki register zinciri arabağlantısı
- Bulunduğu noktadan 4 blok yukarı veya aşağı sinyal geçişini sağlayan C4 ara bağlantıları
- Yukarıdan aşağı cihaz boyunca yüksek hızda dikey sinyal yönlendirmesini sağlayan C16 ara bağlantıları

Cyclone II cihazlarındaki register zinciri ara bağlantısı bir LAB içerisindeki her hangi bir LE yapısının çıkışını bir sonraki LE yapısının girişine çok hızlı bir şekilde ileten bağlantı yapısıdır. Register zinciri bağlantısı bir LE yapısındaki register çıkışının bir sonraki LE yapısındaki register girişine direk olarak bağlanmasını sağlar.

C4 ara bağlantıları ise kaynak bir LAB yapısının yukarısı ya da aşağısındaki 4 LAB yapısı, M4K blokları yada gömülü çarpıcılara doğru yayılır. C4 ara bağlantıları cihaz içerisindeki her

tür mimari blok tarafından sürülebilir. Her bir C4 ara bağlantısı kendisinin yukarısında ya aşağısındaki 4 satırı sürebilir.[26-27]



Şekil 3.8: Register zinciri ve C4 ara bağlantısı[26-27]

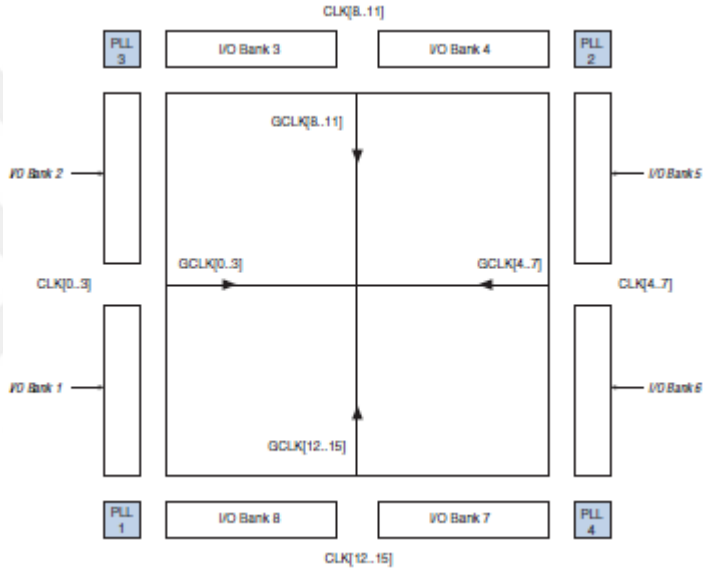
C16 sütun ara bağlantıları 16 LAB uzunluğuna kadar yayılır ve LAB'lar, M4K bellek blokları, gömülü çarpanlar ve IOE'ler arasındaki uzun sütun bağlantılarında kullanılan en hızlı ara bağlantı kaynağıdır. C16 ara bağlantıları diret olarak LAB yapılarının yerel arabağlantılarını süremezler. Bu işlemi ancak C4 ve R4 ara bağlantıları vasıtasıyla gerçekleştirebilirler.[26-27]

3.1.3. PLL(Phase Lock Loop) Yapıları

Cyclone II cihazları içerisinde en fazla 4 tane PLL yapısı bulunur. PLL yapıları cihaz, harici sistem ve I/O ara yüzleri için, saat(clock) yönetimini ve sentezini gerçekleştirir. PLL yapıları genel amaçlı saat(clock) denetimin yanında aşağıdaki özelliklere sahiptir.

- Clock bölme ve çarpma
- Faz kaydırma

- Programlanabilir işlem döngüsü
- Üçe kadar dahili clock çıkışı
- Bir ayrılmış harici clock çıkışı
- Diferansiyel I/O desteği için clock çıkışları
- Manuel clock değişimi
- Kilit sinyal kapısı (gated lock signal)
- Üç farklı clock geri besleme modu
- Kontrol sinyalleri



Şekil 3.9: PLL yapılarının FPGA içerisindeki konumları[26-27]

PLL yapılarının ana görevi gerilim kontrollü osilatör(VCO) yapılarının faz ve frekansının senkronizasyonunu sağlamaktır.[26-27]

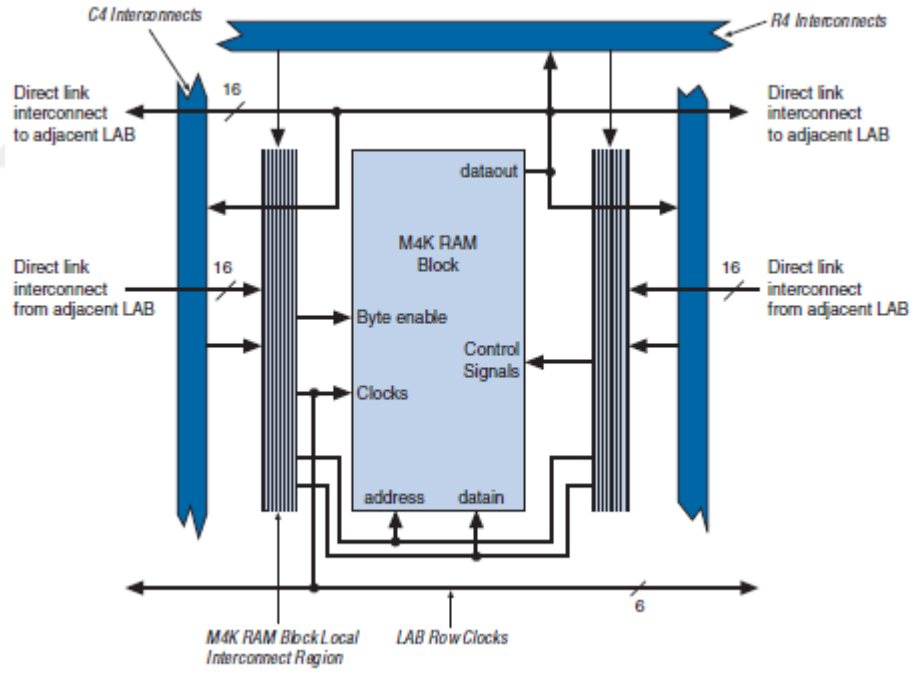
3.1.4. Gömülü Bellekler

Cyclone II mimarisinde gömülü bellekler M4K bellek bloklarından oluşturulmuştur. Bu boklar içerisinde verinin senkronize olarak yazılması için giriş register yapıları ve pipeline şeklindeki tasarımlar ve sistem performansını geliştirmek için ise çıkış register yapıları içerir. Çıkış register yapıları atlanabilirken giriş register yapıları atlanamamaktadır. M4K blokları aşağıdaki özellikleri destekler.

- 4.608 RAM biti içerir
- 250 MHz hızına kadar performans gösterir

- Çeşitli bellek modunu destekler(single port, single-dual port, true dual port, shift register, FIFO buffer, ROM)
- Tek ya da çift saatli FIFO M4K blokları içine uygulanabilir. (FIFO buffers)
- Her byte için bir parity(eşlik) biti içerir
- Byte boyutunda yazma ya da okuma desteklenir
- Çeşitli clock modlarını destekler
- Bir önceki adres değerinin tutulmasını sağlayan adres clock enable desteği
- True dual-port operasyonu(bir yazma bir okuma işlemi, iki yazma işlemi ya da iki okuma işlemi) desteklenir

M4K bellek bloğunun yerel ara bağlantılarını, R4, C4 ve komşu LAB yapılarının direkt link ara bağlantıları sürer. M4K bellek blokları satır ve sütun ara bağlantıları aracılığı ile sağ, sol, alt ve üst tarafındaki LAB yapıları ile haberleşebilir.[26-27]



Şekil 3.10: M4K Bellek Bloğu LAB satır arayüzü[26-27]

3.1.5. Gömülü Çarpıcılar

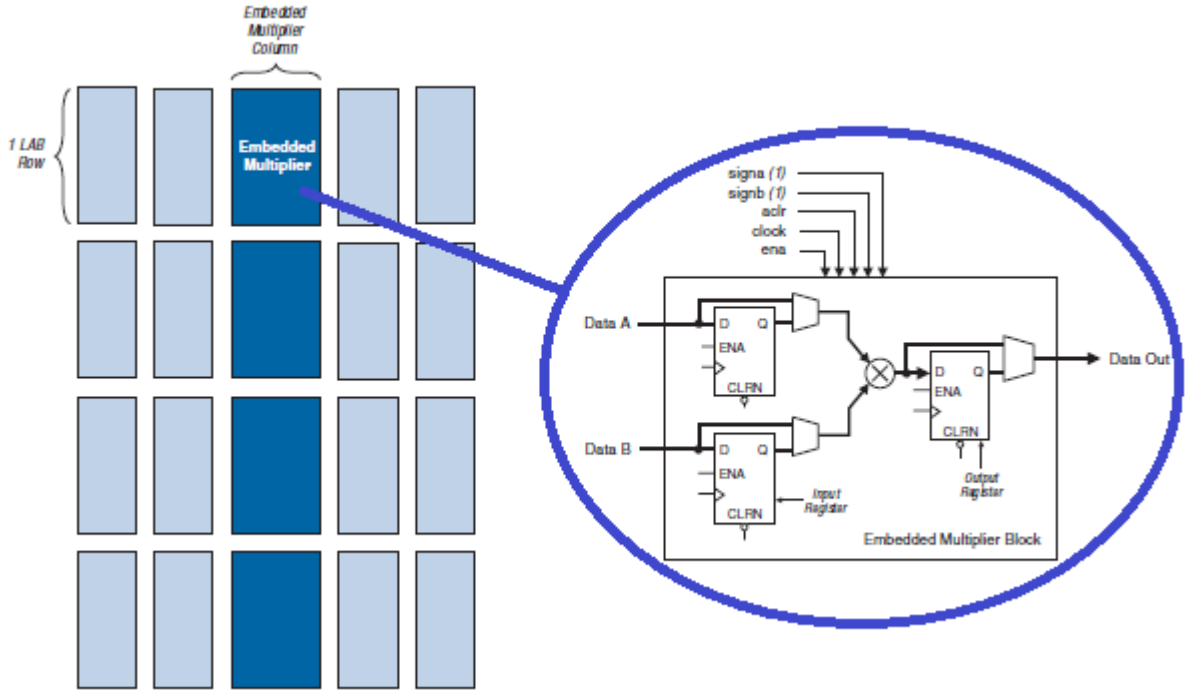
Cyclone II cihazları, sonlu dürtü cevaplı(FIR) filtreler, hızlı fourier dönüşümleri (FFT) ve ayrık cosin dönüşümü (DCT) gibi çarpan yoğunluğu yüksek olan dijital sinyal işlemede

(DSP) kullanılmak üzere optimize edilmiş gömülü çarpıcı yapılarına sahiptir. Tasarımın ihtiyacına göre gömülü çarpıcılar iki modda çalışabilmektedir. Bu modlar aşağıdaki gibidir.

- Bir 18 bitlik çarpııcı,
- İki bağımsız 9 bitlik çarpıcı

Gömülü çarpıcılar maximum 250 MHz'lik hıza ulaşabilir. Her Cyclone II cihazı bir ila 3 sütun arasında gömülü çarpıcıya sahiptir. LAB yapıları gibi cihaz boyunca yukarıdan aşağıya tüm sütunu kaplarlar. Gömülü çarpıcılar aşağıdaki yapılardan oluşur.

- Çarpıcı bloğu
- Giriş ve çıkış register'ları
- Giriş ve çıkış arayüzleri



Şekil 3.11: Çarpıcı Blok mimarisi[26-27]

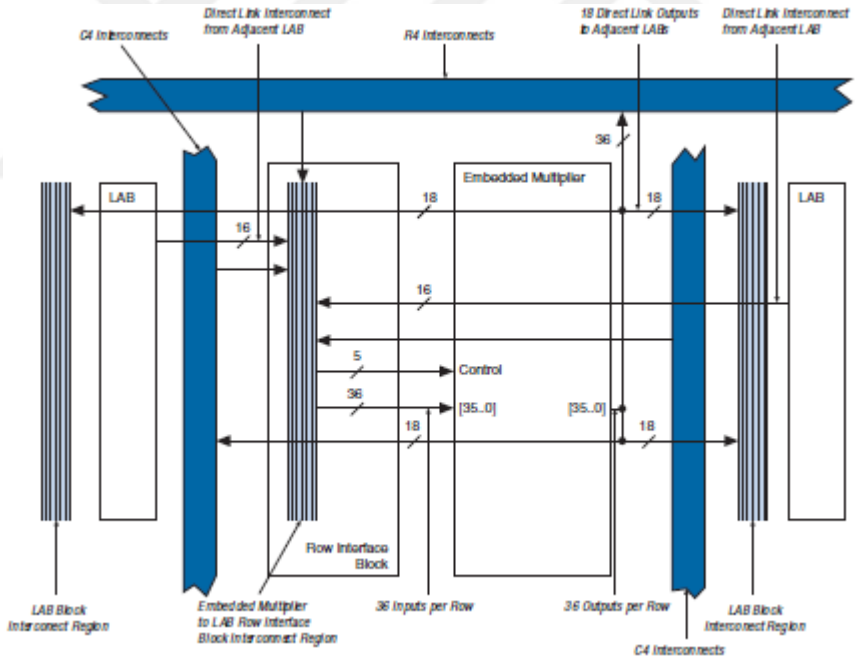
Çarpıcı bloğunun girişleri işaretli ya da işaretsiz sayılar olabilir. Giriş sinyallerinin işaretli mi yoksa işaretsiz mi olduğu *signa* ve *signb* değerleri ile belirlenir. *Signa* değer lojik 1 ise bu *dataA*'nın işaretli olduğunu gösterir, lojik 0 ise datanın işaretsiz olduğunu gösterecektir. Aynı ilişki *signb* ve *dataB* arasında da vardır. Girişlere uygulanan sayıların türlerine göre çıkış değeri aşağıdaki tabloda verilmiştir.

Tablo 3.1: Çarpıcılarda *Sign* gösterimi[26-27]

dataA(signa değeri)	dataB(signb değeri)	Sonuç
Sign	Sign	Sign
Sign	Unsign	Sign
Unsign	Sign	Sign
Unsign	Unsign	Unsign

Her bir ayrılmış çarpıcı için sadece bir *signa* ve sadece bir *signb* sinyali vardır. Dolayısı ile ayrılmış çarpıcı yapısını besleyen her dataA girişi aynı işaret değerine sahip olmalıdır. Bu durum dataB girişi için de geçerlidir.

Gömülü çarpıcı yapılarını R4, C4 ara bağlantıları ve komşu LAB'lardan gelen direkt link ara bağlantıları sürebilir. Gömülü çarpıcı yapıları sağındaki ya da solundaki LAB yapıları ile satır ara bağlantıları üzerinden, altındaki ya da üstündeki LAB yapıları ile de sütun ara bağlantıları üzerinden haberleşebilir.[26-27]

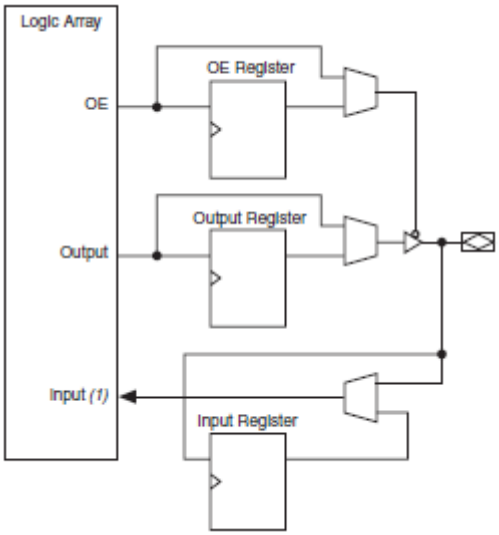


Şekil 3.12: Çarpıcıları Yapılarının Routing Arayüzü[26-27]

3.1.6. I/O Yapısı ve Özellikleri

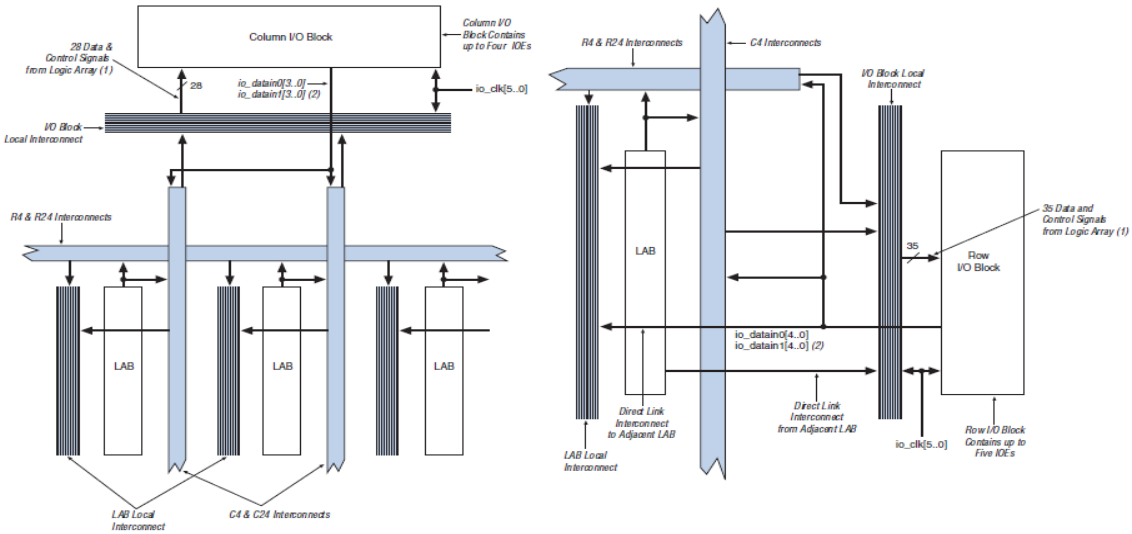
Cyclone II cihazının IO elementlerinin yapısı çift yönlü bir *I/O buffer* ve 3 *register* yapısından oluşmaktadır. Bu üç *register* yapısı *input register*, *output register* ve *output enable register*

şeklinde adlandırılır. Bu register yapıları IO elementlerinin zamansal performansını artırmak için kullanılır. IO elementleri giriş, çıkış ve çift yönlü pinler şeklinde kullanılabilir.



Şekil 3.13: IO elementi yapısı[26-27]

IO elementleri (IOE) Cyclone II cihazı etrafında yerleşmiş olan I/O bloklarının içerisinde yer almaktadır. Her bir satır I/O bloğunun içinde maksimum 5'e kadar IO elementi ve herbir sütun I/O bloğunun içerisinde ise maksimum 4 adet IO elementi bulunur. Satır I/O blokları, satır, sütun (sadece C4 ara bağlantısı) ve direkt link ara bağlantılarını sürer. Sütun I/O blokları ise sütun ara bağlantılarını sürerler.[26-27]



Şekil 3.14: I/O Bloklarının Arabağlantılara Bağlantısı[26-27]

3.1.7. Cyclone II – EP2C70F896C6 FPGA Yapısı

Altera DE2-70 geliştirme kartlarında kullanılan FPGA mimarisi Cyclone II EP2C70F896C6 FPGA ailesidir. EP2C70896C6 kodunda verilen 2C; Cyclone II'yi, 70; yaklaşık lojik element sayısının 70 bin olduğunu, 896; ürünün pin sayısını ve C6 ise ürünün hız seviyesini belirtir. Dolayısı ile Cyclone II EP2C70896C6 FPGA'i yaklaşık 70.000 lojik elemente ve 896 pine sahip hız sınıfı 6 olan bir Altera ürünüdür denilebilir.[29]

Geliştirme kartına entegre edilmiş Cyclone II – EP2C70F896C6 FPGA ailesi özellikleri aşağıdaki gibidir.

- 68,416 Lojik Element(LE)
- 250 M4K RAM blokları
- 1.152.000 toplam RAM bitleri (yaklaşık 1.1Mbit)
- 150 adet 18x18 gömülü çarpıcı
- 4 PLLs (phase lock loop)
- 622 adedi kullanıcıya ayrılmış olmak üzere toplam 896 adet I/O pini

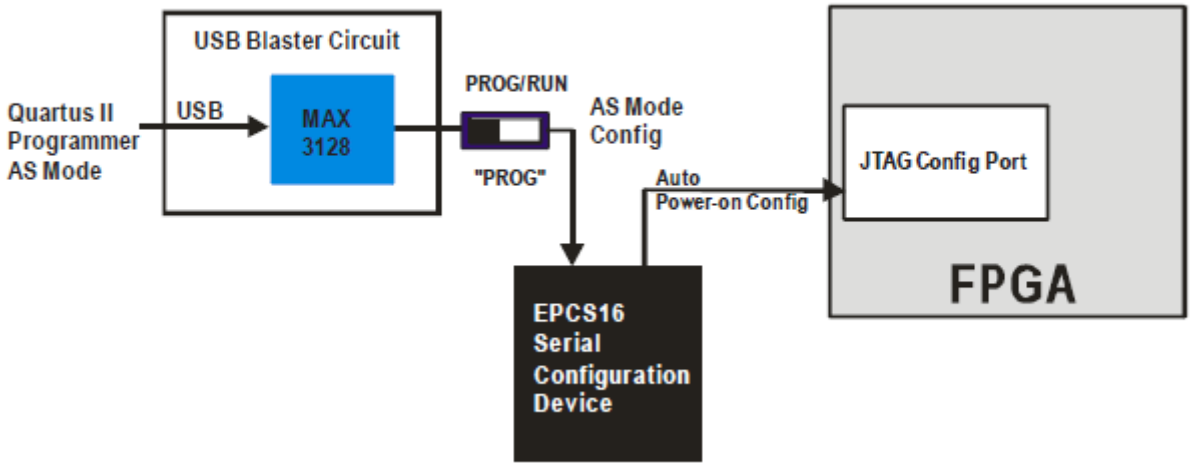
3.1.8. Cyclone II FPGA Konfigürasyonu

DE2-70 geliştirme kartında, Cyclone II FPGA yapısına yüklenmek üzere konfigürasyon verisini saklayan bir seri EEPROM çipi vardır. Konfigürasyon verisini tutan bu çip geliştirme kartına güç verildiği anda otomatik olarak FPGA'e konfigürasyon verisini yükler. Quartus II yazılımı ile FPGA'i tekrar programlamak ve seri EEPROM çip içerisinde tutulan uçucu olmayan veriyi değiştirmek mümkündür.

Cyclone II FPGA konfigürasyonu seri EEPROM çipini kullanmadan, direk olarak FPGA'e konfigürasyon verisinin yüklenmesiyle de gerçekleştirilebilir. Fakat bu durumda Cyclone II FPGA yapılarının SRAM tabanlı olmalarından kaynaklı olarak cihazın gücü kesildiğinde FPGA üzerindeki veri silinecektir ve ikinci bir kullanımda cihazın yeniden konfigüre edilmesi gerekecektir.

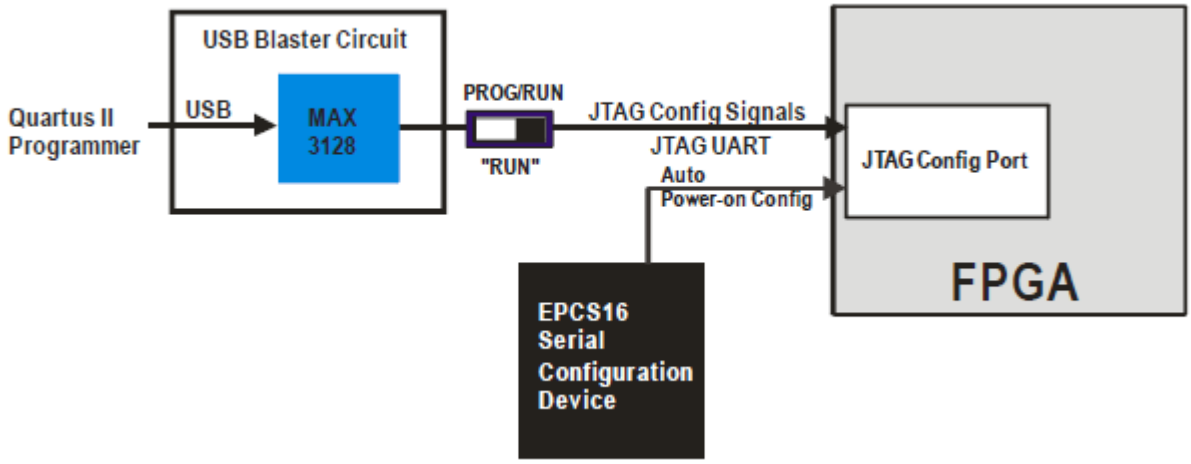
Bu iki farklı konfigürasyon işlemi DE2-70 geliştirme kartında iki farklı programlama modu şeklinde kaydedilmiştir. Bu modlar AS(active serial) programlama modu ve JTAG(joint test action group) programlama modu şeklindedir.[29]

AS programlama: Bu programlama metodunda konfigürasyon bit stream'i geliştirme kartı içerisindeki ALTERA EPCS16 seri EEPROM çipi içerisine yüklenir. Bu, konfigürasyon verisinin kalıcı olarak cihaz içerisinde saklanmasını sağlar. Bu sayede kartın gücü kesilse dahi veri EEPROM yapısı içerisinde saklanacaktır ve cihaza tekrar güç verildiğinde EPCS16 cihazında depolanan veri otomatik olarak Cyclone II FPGA cihazına yüklenecektir.



Şekil 3.15: AS Programlama Metodu[29]

JTAG programlama: Bu programlama metodu ile konfigürasyon bit stream'i direkt olarak Cyclone II FPGA yapısına yüklenecektir. Cihazın gücü kesilmediği sürece bu veri FPGA içerisinde kalacaktır. Güç kesildiğinde ise konfigürasyon verisi silinecektir.

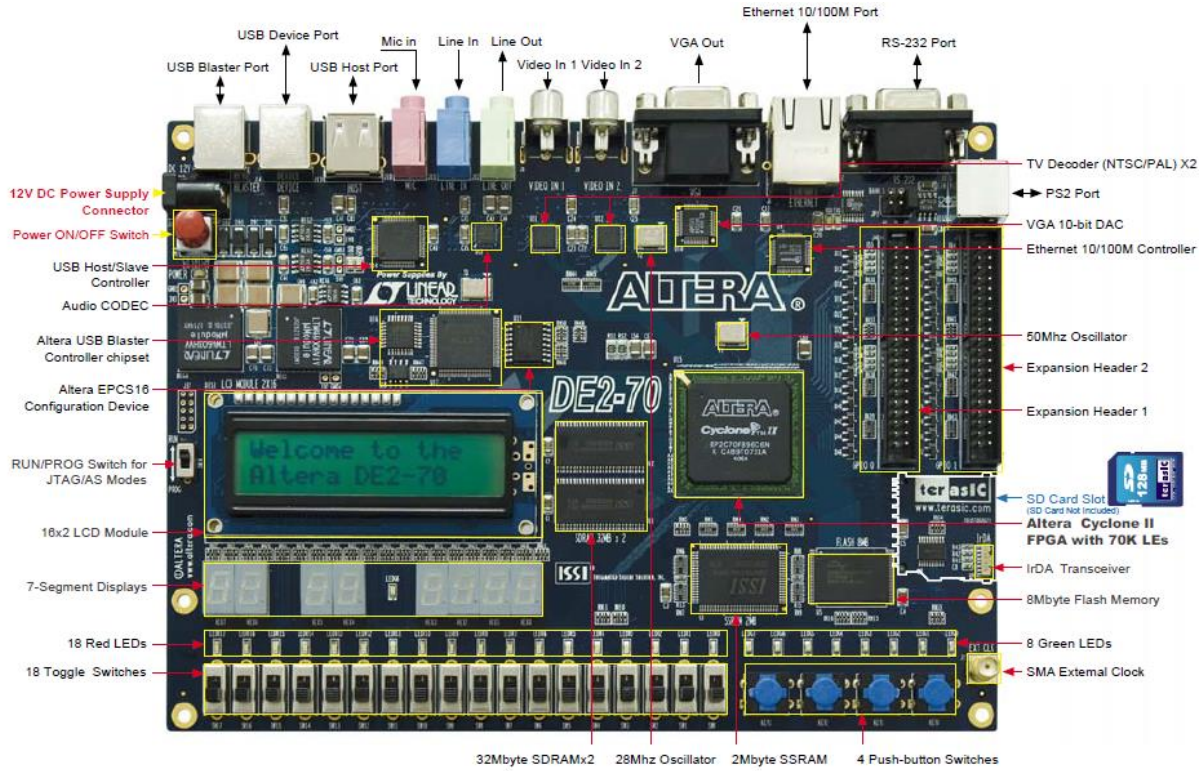


Şekil 3.16: JTAG Programlama Metodu[29]

3.1.9. DE2-70 Geliştirme Kartı

Tezin uygulanmasında, Altera firmasının geliştirdiği Cyclone® II 2C70 FPGA mimarisine sahip DE2-70 geliştirme kartı kullanılmıştır. Geliştirme kartının genel özellikleri aşağıdaki gibidir.

- Altera Cyclone® II 2C70 FPGA mimarisi
- Altera seri yapılandırma cihazı - EPCS16
- Programlama ve kullanıcı API kontrolü için USB Blaster; hem JTAG hem de aktif seri programlama modu destekli
- 2-Mbyte SSRAM
- İki 32-Mbyte SDRAM
- 8-Mbyte Flash bellek
- SD Card soketi
- 4 adet pushbutton anahtarı
- 18 adet toggle anahtarı
- 18 adet kırmızı kullanıcı LED'i
- 9 adet yeşil kullanıcı LEDs
- 1 adet 7-segment display
- Saat kaynağı için 50-MHz ve 28.63-MHz osilatör
- Hat girişi, hat çıkışı ve mikrofon girişi jaklarına sahip 24 bit CD kalitesinde audio CODEC
- Video çıkışı (10 bit yüksek hızlı üçlü VGA DAC'ler).
- 2 TV Decoder (NTSC/PAL/SECAM) ve TV-in konnektör
- Bir konnektör ile birlikte 10/100 Ethernet Denetleyicisi
- USB Host/Slave Denetleyici
- RS-232 alıcı-verici ve 9-pinli konnektör.
- PS/2 fare/klavye konnektörü



Şekil 3.17: DE2-70 Geliştirme Kartı[29]

3.2. FPGA PROGRAMLAMA DİLLERİ

FPGA programlamasında yaygın olarak kullanılan iki HDL(Hardware Description Language) dili mevcuttur. Bunlar Verilog HDL ve VHDL dilleridir. Bu iki dilin birbirine göre bazı üstünlükleri vardır. Bu sebeple seçim hangi üstünlüğü tercih ettiğinize göre değişmektedir.

Bu çalışmada kullanılan donanım tanımlama dili VHDL dilidir. Bu sebeple Verilog HDL dili ile ilgili ayrıntılı bilgiye değinilmemiştir.

3.2.1. Verilog HDL

İlk modern Verilog HDL dili 1984 yılında dijital devreleri modelleme, simülasyon ve analiz amacıyla basit ve etkili bir şekilde ifade etme için Gateway Design Automation firması tarafından öne sürülmüştür. 1995 yılında IEEE tarafından standartlaştırılmıştır. 2001 yılında güncellenmiş ve Verilog HDL 2001 standardı oluşturulmuştur.

Sentaks olarak Verilog HDL dili, C programlama dili ile benzerlik göstermektedir. Büyük küçük harf duyarlılığı vardır. İfadeler “;” ile sonlandırılır. Maksimum 1024 karakter kullanılabilir.

Verilog ile dijital sistemler hiyerarşik modüller ve bu modüllerin arasındaki bağlantılar ile tanımlanır.

Verilog HDL’de paket kavramı yoktur ve tüm programlama, programcı tarafından sağlanan basit veri türleriyle yapılmalıdır. Öğrenilmesi kolay olmasına karşılık karmaşık sistemlerin tasarımında fazla tercih edilmemektedir. Karmaşık veri türlerinin oluşturulması desteklenmez.

3.2.2.VHDL (Very High-Speed Integrated Circuit Hardware Description Language)

1981’de ABD savunma bakanlığı tarafından donanım tanımlama dili olarak önerilmiş 1986 yılında IEEE tarafından standartlaştırılmıştır. Daha sonra sırası ile 1987, 1993, ve son olarak 2002’de dil standardı güncellenmiştir. Günümüzde endüstriden akademiye yayılan geniş bir alanda dijital devrelerin simülasyonu ve sentezi için kullanılmaktadır.

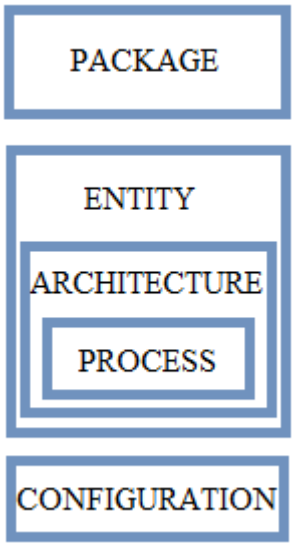
Sentaks olarak bakıldığında Verilog HDL dilinin C benzerliğine karşıt olarak VHDL dilinde bu benzerlik yoktur. VHDL dilinde yazılan kodlar ilkel İngilizce cümlelere benzetilebilir. Büyük küçük harf duyarlılığı yoktur. **”--“** ile yorum satırları başlar.

Öğrenilmesi zor yüksek seviye bir dildir. Karmaşık sistemlerin tasarımı içi uygundur. Kullanıcıya karmaşık veri türleri oluşturulması için olanak sağlar.[4]

3.2.2.1. Tasarım Birimleri

Kütüphane birimleri olarak da adlandırılan tasarım birimleri VHDL dilinin ana bileşenleridir. 4 farklı bildirimden oluşmaktadır. Bunlar; package, entity, architecture ve configuration-component bildirim çeşitleridir.

Bir tasarım herhangi sayıda package, entity, architecture ve configuration bildirimi içerebilir. Bu dört çeşit bildirimin birbiriyle ilişkisi aşağıdaki şekilde gösterilmiştir.



Şekil 3.18: Bildirim türlerinin birbiri ile ilişkisi

Bir VHDL tasarımı kütüphane bildirimi yapıldıktan sonra *entity* bloğu ile başlar. Bu blok tasarımın giriş çıkış sinyallerinin tanımlandığı arayüzü oluşturur. *Entity* bloğunun ardınan *architecture* bloğu başlar ve bu blokta tasarımın iç operasyonları tanımlanır. Bildirim türleri içinde bulunan package ve configuration türleri opsiyoneldir.

Tasarım oluşturulduktan sonra, mantıksal çalışmasını kontrol etmek için simüle edilebilir ve sentezlenebilir. Simülasyon, temel mantığın tasarım ve konsepte göre çalışıp çalışmadığını görmek için çıplak bir kemik testi türüdür.

Entity: Bu blok VHDL tasarımın girişi kısmıdır. Her tasarımın yalnızca bir *entity* bloğu bulunur. Tasarımın giriş çıkış portları bu kısımda tanımlanır. *Entity* bildirimi aşağıdaki gibi yapılır.

```
entity entity_adi is

port (giris_sinyal_adi, giris_sinyal_adi2: giris_sinyal_modu;
      cikis_sinyal_adi, cikis_sinyal_adi : cikis_sinyal_modu);

end entity_adi;
```

Mavi renkte yazılan kelimeler bu kelimelerin rezerve edilmiş olduğunu gösterir. Port isimleri noktalama işaretleri ile ya da rakamlar ile başlayamaz ve VHDL dili kelimeler arası boşluklara duyarsızdır.

Modu aynı olan sinyaller virgülle birbirinden ayrılarak ifade edilebilir. Eğer ayrı ifade edilmek isteniyor ise **";"** ile ayrılmalıdır. Ayrıca sinyal modlarının ardından sinyalin veri türü de bildirilmelidir. Veri türü ve modu tanımlayıcı olduğu için sinyal ismi ile bu tanımlayıcılar arasında **":"** kullanılmalıdır. Bu kullanıma uygun *entity* bildirimi aşağıda verilmiştir.

```
entity SR_flip_flop is

port (S, R, clk    :in  std_logic ;
      Q_out, nQ_out:out std_logic);

end SR_flip_flop ;
```

Sinyal modları **in, out, inout** ve **buffer** olabilir. Ayrıca entity bloğunun içerisinde tasarımın güncellenmesinde kolaylık sağlayacak değişmez bir sabit tanımlanabilir. Bu değişmez sabit *generic* olarak ifade edilir. Örneğin bir bit_vector türünün veriyolu uzunlu 8 bit fakat siz bu niceliği değiştirmek istiyorsunuz. Bu değişiklik *generic* ile tanımlanan sabit değerinin değiştirilmesi ile gerçekleşir. Aşağıda *generic* bildirim sentaksı verilmiştir.

```
generic(isim : veri_turu:=deger);
```

Aşağıda *generic* bildirimi örneği verilmiştir.

```
entity alu is

generic(K       :integer:=64);
port    (input1  :in  std_logic_vector(K-1 downto 0) ;
         input2  :in  std_logic_vector(K-1 downto 0) ;
         alu_cont:in  std_logic_vector(4 downto 0)   ;
         clk     :in  std_logic                      ;
         reset   :in  std_logic                      ;
         alu_out :out std_logic_vector(K-1 downto 0) ;
         alu_rout:out std_logic_vector(K-1 downto 0) ;
         flag_reg:out std_logic_vector(4 downto 0))  ;

end alu;
```

Architecture: Bu blok *entity*'nin nasıl çalıştığını tanımlar. Bu çalışma farklı yollarla yapılabilir. En yaygın iki yol davranışsal (behavioral) ve yapısal (structural) formatları ile tasarımdır. Bir diğer yol ise veri akışı (data flow) formatıdır.

Davranışsal tasarım yaklaşımında devrenin mantıksal davranışını tanımlar. Bu genellikle Boole ifadeleri ile gerçekleştirilir. Yapısal tasarım yaklaşımında ise entity'nin nasıl yapılandırıldığı, tasarımı hangi mantıksal cihazların oluşturduğu tanımlanır.

Genel *architecture* bildirim sentaksı aşağıdaki gibidir.

```
architecture arch_ismi of entity_ismi is

atamalar_ve_sinyal_bildirimleri;

begin

islemleri_tanımlayan_ifadeler;

end arch_ismi;
```

SR flip flop yapısının *architecture* bloğu ile birlikte gösterimi aşağıdaki gibidir.

```
library ieee;
use ieee.std_logic_1164.all;

entity SR_flip_flop is

  port (S, R, clk    :in  std_logic ;
        Q_out, nQ_out:out std_logic);

end SR_flip_flop ;

architecture SR_arch of SR_flip_flop is
begin

 Q_out <= (S nand clk)nand nQ_out;
 nQ_out<= (R nand clk)nand Q_out ;

end SR_arch;
```

İlk iki satır, std_logic ve std_logic_vector gibi önceden tanımlanmış mantıksal işlevler ve veri türlerini içeren IEEE standart mantık kütüphanesi olan std_logic_1164 kütüphanesini içe aktarır. "Use" ifadesi, bir kütüphane dosyasının hangi bölümlerinin kullanacağını belirler. Bu örnekte, 1164 kütüphanesindeki tüm öğeler seçilmiştir. Bir sonraki blokta (entity) SR flip flobunun giriş portları olan *S, R* ve *clk* ve çıkış portları olan *Q_out* ve *nQ_out* bildirimi yapılmıştır. *Entity* bloğunu *SR_arch* isimli *architecture* bloğu takip eder. *Architecture* bloğu gövdesi *begin* ifadesi ile başlar. *Begin* ifadesinin ardından 2 tane sinyal ataması yapılmıştır.

Bu iki atamada Boole ifadeleri kullanılarak SR flip flobunun davranışı *nand* kapıları ile ifade edilmiştir.[4]

Sinyal atamaları "<=" ifadesi ile gerçekleştirilir. Bu ifade generic tanımlamasında kullanılan ":=" ile ifade edilen atama ifadesinden farklıdır. "<=" ifadesi değişke değerli sinyal atamaları için ":=" ifadesi ise statik bir değere sahip değeri sadece elle değiştirilebilen atamalarda kullanılır.

Configuration: configuration bildirimleri, belirli tasarım öğelerini hiyerarşik bir tasarımdaki bileşen örnekleriyle ilişkilendirmek veya belirli bir *architecture* yapısını bir *entity* ile ilişkilendirmek için kullanılabilir. Genellikle sentezlenemeyen yapılardır.[4]

Component: yapısal modellemede tasarım bir bütün yerine küçük alt bileşenlerin toplamı şeklinde ele alındığında bu bileşenlerin ifadesi *component* bildirimi ile yapılır. *Component*'ler kendi başlarına birer tasarımdırlar ve kendi başlarına da çalışabilirler. Fakat bu yapıların en önemli özelliği, oluşturulduktan sonra her hangi bir tasarım içerisinde çağrıldığında kendi işlevini çağrıldığı tasarım içerisinde de gerçekleştirmesidir.

Herhangi bir tasarıma component yapısının dahil edilmesi iki aşamadan oluşur. İlk aşama olarak *component* bildirimin yapılması gerekir. Bildirimde aslında *component* portlarının tanıtımı yapılır. Bildirimi yapılan *component* yapıları tasarım içerisinde tekrar tekrar kullanılabilir. Aşağıda *component* bildirimi sentaksı verilmiştir.

```
--Component sentaksı

 component component_adı
--varsa generic bildirimi yapılır
  generic(jenerik_ismi:veri türü:=değer;
  ¦   ¦   ¦ diğer jenerikler);

  port    (port_ismi:port_modu veri_türü;
  ¦   ¦   ¦ diğer portlar);
 end component;

--Component örneği
 component half_adder
  port(A, B :in std_logic  ;
  ¦   ¦ S, C :out std_logic);
 end component;
```

Yukarıdaki örnekte bildirimi yapılan *half_adder component'*i aslında vhdl dilinde yazılmış, yarım toplayıcı işlemini yerine getiren, kaydedilmiş bir dosyadır. Kaydedilen bu dosya mutlaka mevcut tasarıma eklenmelidir. Kayıtlı dosya tasarıma eklenmediği taktide *component* bildirimi hiçbir işe yaramayacak ve bileşen çağırıldığında çalışmayacaktır.

Component bildirimi *architecture* bloğu içerisinde *begin* ifadesinden hemen önce yapılır.

İkinci aşama ise bildirimi yapılan *component*'in program içerisinde çağrılmasıdır. Bileşen çağrıldığında tasarım içerisindeki port eşlemesi yapılmış olur. Bu kısımda component portları mevcut tasarımda tanımlanan sinyallerin hangilerine karşılık geleceği ifade edilir. Aşağıda component çağrım sentaksı verilmiştir.

```
--Componet çağrım sentaksı
etiket:component_adı
    generic map (jenerik_ismi=>değer, diğer jenerikler)
    port map (port_ismi=>sinyal_ismi, diğer portlar);
```

Port eşleştirmesi yerleşime bağlı olarak ya da isme bağlı olarak yapılabilir. Aşağıda iki yönteme ilişkin örnekler verilmiştir.

```
--İsme bağlı Componet çağrılmasına ilişkin örnek
architecture behavioral of example is
 signal X,Y,Z,T: std_logic;

 component half_adder
  port(A, B :in std_logic  ;
       S, C :out std_logic);
 end component;

 begin
 --isme bağlı çağrı
 C1: half_adder port map(A=>X, B=>Y, S=>T, C=>Z);
end behavioral;

--Yerleşime bağlı Componet çağrılmasına ilişkin örnek
architecture behavioral of example is
 signal X,Y,Z,T: std_logic;

 component half_adder
  port(A, B :in std_logic  ;
       S, C :out std_logic);
 end component;
 begin
   --yerleşime bağlı çağrı
 C1: half_adder port map(X,Y,T,Z);
end behavioral;
```

2.3.2.2. İfade Türleri

VHDL dilinde 3 çeşit ifade türü vardır. Bunlar; bildirim ifadeleri, eşzamanlı ifadeler ve sıralı ifadelerdir.

Bildirim İfadeleri: Bu ifadeler tasarım içerisinde kullanılan sabitler (number, string gibi), nesneler(signal, variable ve component gibi), türler(record ve array gibi) ve altprogramları(function, ve procedure gibi) tanımlamada kullanılır.

Eşzamanlı İfadeler: Bu ifadeler yazılış sırasından bağımsız olarak aynı anda gerçekleşen ifadelerdir. Eşzamanlı ifadelerde veri sinyaller ile taşınır. Sinyal atamaları, component yapıları, generate ifadesi, process yapıları, procedure ve function çağırma işlemleri eş zamanlı olarak gerçekleşir. **"<="** ile yapılan tüm atamalar eş zamanlı olarak gerçekleşir.

```
architecture dataflow of my_circuit is

signal d,e:bit;

begin

 d<= in1 and in2;
 e<= in3 or in4;
 out1<= in5 xor e;
 out2<= in6 xor d;

end dataflow;
```

Yukarıda *architecture* bloğunda verilen sinyal atamaları eş zamanlı olarak gerçekleşmektedir.

Sıralı İfadeler: Sıralı ifadeler eş zamanlı ifadelerden farklı olarak yazıldıkları sıra ile gerçekleşirler. Sıralı ifadeler daima bir process, bir function ya da procedure ifadesi içerisinde yazılır. C ya da Pascal gibi yazılım programlama dillerindeki sıralı ifadeler ile benzerdirler. İf-then yapıları, case yapıları, while dögüleri, for döngüleri, when yapıları, next ve wait yapıları gibi koşullu ifadeler sıralı olarak gerçekleşir. Aşağıda sıralı ifadelere örnek verilmiştir.

```
library ieee;
use ieee.std_logic_1164.all;

entity DFF is

port ( D, CLK : in std_logic;
       Q  : out std_logic;
       QN : out std_logic := '1');
end DFF;

architecture data_flip of DFF is

begin
 process ( CLK )
 begin
   if (CLK = '1' and CLK'event ) then
       Q  <= D after 10ns;
       QN <= not D after 10ns;
   end if;
 end process;

end data_flip;
```

3.2.2.3. Veri Nesneleri

Program içerisinde veri türündeki değerleri tutan yapılardır. C ve Pascal gibi programlama dillerinde *variable* olarak tek tip veri nesnesi bulunur. VHDL dilinde ise bu veri nesnesine ilave olarak 2 farklı veri nesnesi bulunur ve bunlar *constant* ve *signal* veri nesneleridir. Ayrıca *variable* altına *shared variable* veri nesneleri de ilave edilebilir.[4]

Variable: Tek parça bilgi içerir. Değeri değişebilir. Atama işlemleri ":=" ifadesi ile yapılır. VHDL dilinde 3 yerde *variable* ver nesnesi tanımlanabilir. Bunlar *process* tanımlama kısmı, *function* tanımlama kısmı ve *procedure* tanımlama kısmıdır. Tanımlama *process, function* ya da *procedure* dan sonra *begin* ifadesinden hemen önce yapılır. Birden fazla process içerisinde kullanılan değişken veri nesneleri *shared variable* olarak tanımlanır.

Aşağıda *function* içerisinde *variable* bildiriminin örneği gösterilmiştir. *Process* ve *procedure* ile kullanımı da benzedir.

```
function fonksiyon_adi (parametreler listesi) return donus_tipi is

variable bayrak : std_logic;
variable toplam : bit_vector(0 to 3);
variable sayaç  : integer range 0 to 255;

begin
…
…
end fonksiyon_adi;
```

Constant: Bu veri nesnesi tasarımda asla değiştirilmeyecek olan sabit değerlerin tanımlamasında kullanılır. Bildirimi package, body, block, entity, architecture, process, procedure, function içerdinde yapılabilir. Bildirimi aşağıdaki gibi yapılır.

```
constant carry : std_logic :='1';
constant sonuc : std_logic_vector(3 downto 0) :="0001";
constant sayac : integer :=1;
```

Signal: Signal veri nesnesinin değeri değişkendir, kod mimarisinin Sinyaller; *package, architecture* ya da *block* içerisinde tanımlanabilir fakat *process, procedure* ve *function* içinde *signal* tanımlaması yapılamaz. Signal'a değer ataması **"<="** ifadesi ile yapılır. Tasarım içerisindeki giriş çıkış portları arasındaki bağlantılar ve geçici çıkış değerleri *signal* olarak tanımlanırlar.

```
architecture Behavioral of ornek is

signal bayrak : std_logic;
signal toplam : bit_vector(0 to 3);
signal sayaç  : integer range 0 to 255;

begin
...
...
end behavioral;
```

3.2.2.4. Veri Türleri

Herhangi bir veri tipi, *function* yapıları kullanılarak farklı tiplere çevrilebilirler. Aşağıda bazı veri türü değişimi örnekleri verilmiştir.

```
-- Convert from Integer to Signed using Numeric_Std
signal input_3  : integer;
signal output_3 : signed(3 downto 0);

output_3 <= to_signed(input_3, output_3'length);

-----------------------------------ooo-----------------------------------
-- Convert from Integer to Std_Logic_Vector using Numeric_Std
signal input_1   : integer;
signal output_1a : std_logic_vector(3 downto 0);
signal output_1b : std_logic_vector(3 downto 0);

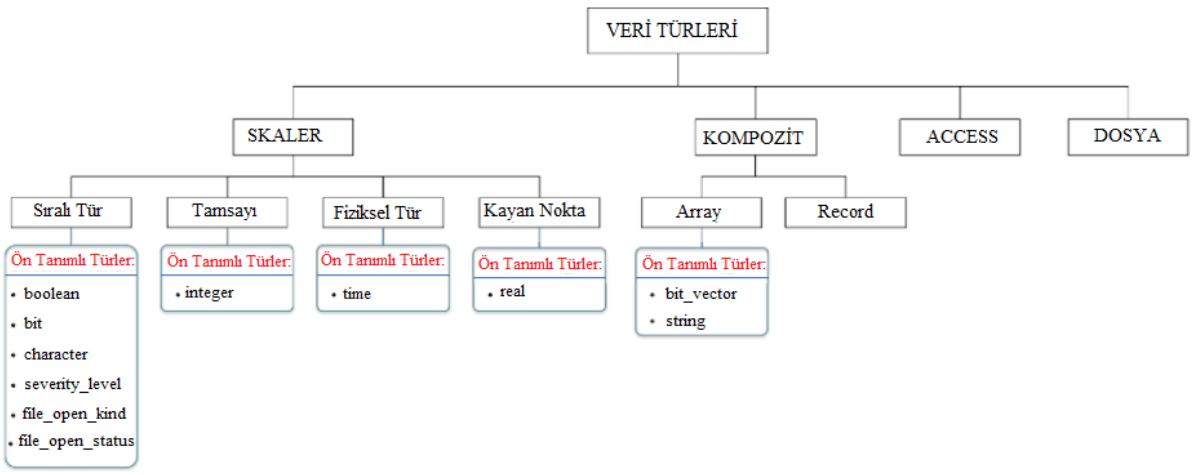
-- This line demonstrates how to convert positive integers
output_1a <= std_logic_vector(to_unsigned(input_1, output_1a'length));

-- This line demonstrates how to convert positive or negative integers
output_1b <= std_logic_vector(to_signed(input_1, output_1b'length));

-----------------------------------ooo-----------------------------------
--Convert from Integer to Unsigned using Numeric_Std
signal input_2  : integer;
signal output_2 : unsigned(3 downto 0);

output_2 <= to_unsigned(input_2, output_2'length);
```

VHDL dilindeki önemli veri türleri bit, bit_vector, std_logic, std_logic_vector, signed, unsigned, integer ve boolean şeklindedir. Bunların dışındaki veri türleri aşağıdaki tabloda gösterilmiştir.



Şekil 3.19: Veri Türleri[4]

3.2.2.5. Operatörler

VHDL ifadeleri arasında gerekli olan mantıksal, aritmetik ve karşılaştırma işlemleri için operatörler kullanılır. Aşağıda VHDL dilinde kullanılan operatörlerin tablosu verilmiştir.

Tablo 3.2: Operatörler[4]

İşlem		A'nın Veri Türü	B'nin Veri Türü	Sonucun Veri Türü
Üssü	A**B	integer	integer	integer
Mutlak değeri	ABS A	integer	-	integer
Değili	NOT A	boolean, bit, bit_vector	-	boolean, bit, bit_vector
Çarpma	A*B	integer	integer	integer
Bölme	A/B			
Mod	A MOD B			
Kalan	A REM B			
Pozitif	+A	integer	-	integer
Negatif	-A			
Toplama	A+B	integer	integer	integer
Çıkarma	A-B			
Birleştirme	A&B	1 boyutlu dizi, dizi elemanı	1 boyutlu dizi, dizi elemanı	1 boyutlu dizi
Mantıksal sola kaydır	A SLL B	bit_vector	integer	bit_vector
Mantıksal sağa kaydır	A SRL B			
Aritmetik sola kaydır	A SLA B			
Aritmetik sağa kaydır	A SRA B			
Sola döndür	A ROL B			
Sağa döndür	A ROR B			
Eşit	A=B	herhangi bir tür	A ile aynı	boolean
Eşit değil	A/=B			
Küçük	A<B	Skaler tür, 1 boyutlu dizi	A ile aynı	boolean
Küçük eşit	A<=B			
Büyük	A>B			
Büyük eşit	A>=B			
Ve	A AND B	boolean, bit, bit_vector	A ile aynı	A ile aynı
Veya	A OR B			
Veya değil	A NOR B			
Ve değil	A NAND B			
Özel veya	A XOR B			
Özel veya değil	A XNOR B			

3.2.2.6.Önemli Yapılar

VHDL dilindeki önemli yapılardan *process, block ve generate,* yapıları eşzamanlı olarak çalışmaktadırlar. *Function* ve *procedure* yapıları ise hem eş zamanlı hem de sıralı olarak çağrılabilmektedirler. Sıralı ifadeler ise *if-then* yapıları, *case* yapıları, döngüler, *wait* ifadesi ve *null* ifadesidir.

Process: *Architecture* bloğu içerisinde birden fazla *process* yapısı tanımlanabilmektedir. Bu blok içerisinde tanımlanan tüm *process* yapıları eş zamanlı olarak çalışırlar fakat bu yapıların içindeki ifadeler sıralı olarak işlenir. Dolayısı ile *process* yapıları sıralı bir işlem gerektiren durumlarda kullanılmaktadırlar.

Process bildirimi sadece *architecture* bloğu içerisinde yapılır. Tasarımda birden fazla *process* kullanılmışsa her bir *process* yapısı için bir isim kullanılabilir. Bu isimler etiket olarak adlandırılır. Tasarımda etiket kullanımı olası karışıklıkları engelleyecek ve okunurluğu artıracaktır.

Architecture yapısı içerisinde birden fazla process yapısı bulunduğunda bu yapılar arasındaki bağlantı sinyaller aracılığı ile gerçekleşir. *Process*'ler içerisinde sinyal tanımlamaları yapılamaz. Bu sinyaller *architecture* bloğunun *begin* kelimesinden hemen önceki kısımda bildirilmelidir.

Process'lerin kontrolü *wait* ifadeleri ile veya duyarlılık listesi ile yapılabilir. *Wait* ifadesi sentezlenebilir bir yapı değildir dolayısı ile sadece simülasyonlarda kullanılabilir. Bu sebeple *process* kontrolü için genel olarak duyarlılık listeleri tercih edilir. Duyarlılık listesi bulunmayan *process* yapıları daima işlenir. Aşağıda duyarlılık listesine sahip bir *process* yapısının kullanım şekli verilmiştir.

```
process_etiketi: process(duyarlilik_listesi)

nesne_bildirimleri;

begin
sıralı_ifadeler;

end process process_etiketi;
```

SR flip flop yapısının *process* yapısı ile ifadesi aşağıdaki gibidir.

```
entity SR_FF is
 port( S,R,clk: in std_logic;
       Q, nQ  : out std_logic);
end SR_FF;

architecture behavioral of SR_FF is
 begin
  process(clk)
  variable tmp: std_logic;
  begin
   if(clk='1' and clk'event) then
     if(S='0' and R='0')then
        tmp:=tmp;
     elsif(S='1' and R='1')then
        tmp:='Z';
     elsif(S='0' and R='1')then
        tmp:='0';
     else
        tmp:='1';
     end if;
   end if;
   Q  <= tmp;
   nQ <= not tmp;
 end process;
end behavioral;
```

Yukarıdaki örnek kod parçasında verilen process yapısı begin ifadesinden sonra if yapısı ile başlamıştır. Bu if yapısı içerisinde clk sinyali ile yapılan işlemler aslında saat sinyalinin yükselen kenarının tespitidir. Saatli process yapılarında saat sinyalinin lojik "1" olduğu kısımlarda devre çalışacaktır. Bu sebeple programda saat sinyalinin yükselen ya da düşen kenarlarına göre devre davranışı process yapısı içerisinde tanımlanır. Saat sinyalinin yükselen ya da düşen kenarının tespiti farklı şekillerde ifade edilebilir. Aşağıda saat sinyali için yükselen kenar ve düşen kenar tespiti ifadeleri verilmiştir.

```
--Yükselen kenar tespiti için kullanılabilecek ifadeler
* if saat='1' and saat'event then
* if rising_edge(saat) then
* wait until saat'event and saat='1'
* wait until rising_edge(saat)

--Düşen kenar tespiti için kullanılabilecek ifadeler
* if saat='0' and saat'event then
* if falling_edge(saat) then
* wait until saat'event and saat='0'
* wait until falling_edge(saat)
```

Kenar tespiti işlemi için kullanılan *rising_edge* ve *falling_edge* fonksiyonları *std_logic_1164* kütüphanesinde tanımlıdır. *Event* ifadesi ise ön tanımlı bir niteliktir ve herhangi bir sinyal üzerinde (genelde saat sinyali üzerinde) meydana gelen değişikliği algılamak için kullanılır.

Block: Bu yapılar da *process* yapıları gibi eş zamanlı olarak çalışırlar. Process'lerden farkı bu yapıların içerisindeki ifadelerin de eş zamanlı olmasıdır. Genel olarak eş zamanlı ifadelerin gruplandırılması için kullanılırlar. Bu sayede okunurluk artacak ve karmaşıklık azalacaktır.

Yazılış sıraları önemli değildir. *Process* yapıları gibi *architecture* bloğu içerisinde yazılır. Gruplandırma için kullanıldığından birden fazla *block* yapısı söz konusudur bu sebeple her bir *block* yapısı için etiket kullanılması gerekir. *Process* yapılarından farklı olarak içerisinde sinyal tanımlaması yapılabilir. Fakat bir *block* içerisinde tanımlanan sinyal yalnızca o blok için geçerli olacaktır. Bu sayede aynı isme sahip farklı tür veya değerleri tutan nesneler oluşturulmasına izin verir.

Block yapısı içinde *type* bildirimi, *subtype* bildirimi, alt program bildirimi, *constant* ve *signal* bildirimi, *component* bildirimi yapılabilir ve *use* ifadesi kullanılabilir. Aşağıda *block* kullanım sentaksı verilmiştir.

```
block_etiketi:block
nesne bildirimleri;
begin
eş zamanlı ifadeler;
end block block_etiketi;
```

Generate: *for* ve *if* yapıları sıralı ifadelerdir. Bu yapıların eş zamanlı durumlar için kullanılmasını sağlayan ifade *generate* yapısıdır. Tekrar edilmesi istenen işlemler için *for–generate* yapısı, eş zamanlı koşullu durumlarda ise *if-generate* kullanılır.

İki *generate* yapısı da etiketsiz kullanılamaz. Nesne bildirimi yapılmayacak ise *begin* ifadesi kullanılmayabilir. *For-generate* yapısının değer aralığı ve *if-generate* yapısının koşul ifadesi sabit değer olması gerekmektedir. *If-generate* yapısında *else* ve *elsif* yapıları kullanılmaz ve iki *generate* ifadesi de iç içe yazılabilir. Kullanım sentaksları aşağıdaki gibidir.

```
--for-generate yapısı sentaksı

  etiket: for değişken in aralık generate
  -- nesne bildirimleri;
  begin
  --eş zamanlı ifadeler;
  end generate etiket;

--if-generate yapısı sentaksı

  etiket: if koşul generate
  --nesne bildirimleri;
  begin
  --eş zamanlı ifadeler;
  end generate etiket;
```

Bu yapıların dışında eş zamanlı olarak gerçekleşen bir diğer işlem sinyal atamalarıdır. Sinyal atamaları "<=" yapısı kullanılarak yapıldığı için eş zamanlı olarak gerçekleşir. Dolayısı ile bir sinyale aynı anda iki farklı değer atanamaz. ":=" ifadesi de atama için kullanılır fakat sinyal atamalarında bu ifade kullanılmaz. ":=" ile yapılan atamalar sıralı gerçekleşir.

Sinyal atamaları basit, koşullu ya da seçimli olarak yapılabilir. Ayrıca herhangi bir veri nesnesine başka bir veri nesnesi kullanılarak atama yapıldığında birbirine ataması yapılan veri nesnelerinin veri türlerinin ve veri türü boyutlarının aynı olduğundan emin olunmalıdır. Aksi takdirde derleme de hata ile karşılaşılacaktır.

Sıralı ifadeler bulunduğu yapı içerisinde yazıldığı sıra ile gerçekleşen ifadelerdir. Sıralı ifadeler yalnızca *process, function* ve *procedure* yapıları içerisinde kullanılabilir. Aşağıda VHDL dilinde önemli bazı sıralı ifade yapıları verilmiştir.

If-then: bir ya da birden fazla koşula bağlı olan işlemlerin ifadesinde kullanılan en yaygın sıralı ifade türüdür. Birden fazla koşul söz konu ise elsif ve else yapıları kullanılır. Sentaksı aşağıdaki gibidir.

```
if koşul_1 then
--sıralı ifadeler

elsif koşul_2 then
--sıralı ifadeler

else
--sıralı ifadeler
end if;
```

Case: bir değişken veya sinyalin alabileceği değerlere göre gerçekleşecek işlemlerin seçiminde *case* yapısı kullanılır. *If* yapısından farklı olarak burada kullanılan tüm seçimlerin önceliği aynıdır. Bu sebeple seçimlerin değer aralıkları birbiri ile örtüşmemelidir ve tüm seçenekler *case* yapısında belirtilmelidir. Yapının sonunda son seçenek *when others* ifadesi ile kullanılırsa *latch* yapıları önlenmiş olur. Sentaksı aşağıdaki gibidir.

```
case seçici ifade is
  when seçim_1 => ifadeler;
  when seçim_2 => ifadeler;
  ..........
  when seçim_n => ifadeler;
  when others  => ifadeler;
end case;
```

Döngüler de sıralı ifadelerdir. *Loop* ifadesi ile sonsuz döngülerin yanı sıra *for* döngüleri ve *while* döngüleri de sıralı döngü ifadeleridir.

Sonsuz döngülerin herhangi bir tekrarlama düzeni yoktur ve sonsuza kadar devam eder. Bu döngüler *when* ve *if* yapıları ile koşula bağlanabilir ve döngüden çıkmak için *exit* ifadesi kullanılır. Aşağıda bu tanıma uygun, sonsuz döngü kullanımı ve örnekleri verilmiştir.

```
--sonsuz döngü sentaksı
  etiket: loop
  --sıralı ifadeler
  end loop;

--sonsuz döngü örnek_1
  U1:loop
  sayici:=sayici+1;
  end loop;

--sonsuz döngü örnek_2
  U1:loop
  if sayici=15 then
     exit;
  else
     sayici:=sayici+1;
  end if;
  end loop;

--sonsuz döngü örnek_2
  U1:loop
     sayici:=sayici+1;
     exit U1 when sayici=15;
  end loop;
```

For loop: *process, function* ve *procedure* yapılarının içinde sıralı durumlarda kullanılır. Döngü değişkeni sadece döngü içerisinde tanımlıdır ve döngü dışında kullanılamaz. Döngü değişkeni işlem esnasında değiştirilemez. Eğer tekrar sayısı değiştirilmek isteniyor ise *while* döngüsü kullanılmalıdır. Aşağıda kullanımı sentaksı ve örneği verilmiştir.

```
--for döngüsü sentaksı
for döngü_değişkeni in aralık loop
-- sıralı ifadeler
end loop;

--for döngüsü örnek_1
function prtyodd(input:std_logic_vector(7 downto 0))return std_logic is
variable temp: std_logic:='0';
begin
 for I in 0 to 7 loop
 temp:=temp xor input(I);
end loop;
return (not temp);
end parity_odd;

--for döngüsü örnek_2
process(basla)
 type renk is (mavi, sari, kirmizi, yesil, beyaz);
 variable sayici:integer:=0;
begin
 for I in mavi to beyaz loop
 sayici:=sayici+1;
 end loop;
end process;
```

While loop: döngü tekrarını while ifadesinden sonra yazılan koşul belirler ve bu koşul sağlandığı sürece döngü devam eder. Aşağıda kullanımına ilişkin örnek verilmiştir.

```
P1: process(A)
  variable I:integer:=1;
begin
  D:while I<=99 loop
  sonuc:=A(I);
  I:=I+1;
  end loop D;
end process P1;
```

Alt programlar olarak da ifade edilen function ve procedure yapıları ise hem eş zamanlı hem de sıralı çağrılabilecek yapılardır.

Alt programlar belirli bir işlevi yerine getirmek için oluşturulmuş tek başlarına çalışamayan yapılardır. Alt program bildirimi ve alt program gövdesinden oluşmaktadırlar. Bildirim

kısmında parametreler ve tanımlayıcılar bulunur. Gövde kısmında ise programın davranış kodu yer alır ve bu kısım sıralı ifadelerden oluşur. Çağrıldıkları ana programlar ile bağlantılar parametreler vasıtası ile gerçekleştirilir.

Function: tek bir değeri döndüren ve içerisinde sıralı ifadeler bulunan yapıdır. Sadece giriş parameteleri içerirler. Bu giriş parametrelerini kullanılarak içerilerinde belirlenen işlemi gerçekleştirir ve işlem sonucunu *return* ifadesi ile çağrıldığı programa iletir. İçerilerinde sinyal tanımlaması yapılamaz ve *wait* ifadeleri kullanılamaz. *Function* bildirimi ve gövdesinden oluşur. Bildirim kısmı ile giriş parametreleri, parametrelerin türleri ve sonucun türü ifade edilir. Giriş parametreleri yalnızca *signal, constant* ve *file* veri nesneleri olabilir. Eğer veri nesne sınıfı fonksiyonda belirtilmemiş ise veri nesnesi *constant* olarak algılanır. Gövde kısmında ise *function* yapısının davranışı tanımlanır. Fonksiyon bildirimi aşağıdaki gibi yapılmaktadır.

```
--function bildirimi sentaksı

function fonksiyon_ismi (virgülle ayrılmış girişler:tür) return sonuc_türü

--örnek

function or_gate(a,b,c:std_logic) return std_logic;
```

Fonksiyon gövdesi sentaksı ve örneği ise aşağıda verilmiştir.

```
-- function gövdesi
function fonksyn_adı (virgülle ayrılmış inputlar:tür) return sonuc_türü is
  --sabit ve değişken tanımlamaları (sinyal türü tanımlanmaz)
begin
  --function body
end fonksiyon_adı;

-- function gövdesi örneği
function or_gate(a,b,c:std_logic) return std_logic is
variable temp:std_logic;
begin
  temp:= a or b or c;
  return temp;
end or_gate;
```

Function yapısı bildirimi architecture bloğunda begin ifadesinden hemen önce yapılır. Fonksiyonun çağrılması ise architecture gövdesinde gerçekleşir. Fonksiyonun çağrılması aşağıdaki gibidir.

```
-- function çağrılması
ana_progrmda_tanmlanmş_sinyal  <= fonksyon_adı(virgülle ayrlmış girişler);
ana_progrmda_tanmlanmş_değişken:= fonksyon_adı(virgülle ayrlmış girişler);

--örnek
sinyal<= or_gate(in1,in3,in2);
sinyal<= or_gate(a=>in1, b=>in3, c=>in2);
--yukarıdaki kullanımların ikisi de aynı sonucu verir
```

Aşağıda tasarım içerisinde *function* yapısının kullanımı örneklenmiştir.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity AND_OR is
    Port ( a       : in   std_logic;
           b       : in   std_logic;
           out_and: out  std_logic;
           out_or : out  std_logic);
end AND_OR;
architecture behavioral of AND_OR is

  function AND_G(x,y:std_logic) return std_logic is
    variable temp:std_logic;
  begin
    temp:=x and y;
    return  temp;
  end AND_G;

  function OR_G(x,y:std_logic) return std_logic is
  begin
    return x or y;
  end OR_G;
begin
  out_and <= AND_G(a,b);
  out_or  <= OR_G(a,b);
end behavioral;
```

Procedure: *Function* yapıları ile benzer işlevdedir fakat bu yapılardan farklı olarak hem giriş hem çıkış parametreleri içerirler ve birden fazla değeri döndürebilirler. İçindeki ifadeler sıralıdır. *Procedure* bildirimi ve gövdesinden oluşur. Bildirimin yapıldığı yer *function* yapıları ile aynıdır. Aşağıda *procedure* kullanımına ilişkin örnek verilmiştir.

```
--procedure sentaksı
 procedure prosedür_adı(virgülle ayrılmış inputlar: in tür;
                        virgülle ayrılmış outpular: out tür) is
  --sabit ve değişken tanımlamaları (sinyal türü tanımlanmaz)
 begin
  --procedure gövdesi
 end prosedür_adı;

--procedure örneği
 procedure or_gate (signal a,b,c : in std_logic_vector(4 downto 0);
                    signal or_out:out std_logic_vector(4 downto 0)) is
 begin
  for I in 0 to 3 loop
   or_out(I)<= a(I) or b(I) or c(I);
  end loop;
 end or_gate;
```

Procedure yapılarının ana program içerisinde çağrılması da *function* yapıları ile aynıdır. Aşağıda tasarım içerisinde *procedure* kullanımı örneklenmiştir.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity AND_OR is
    Port ( a       : in   std_logic;
           b       : in   std_logic;
           out_and: out  std_logic;
           out_or : out  std_logic);
end AND_OR;

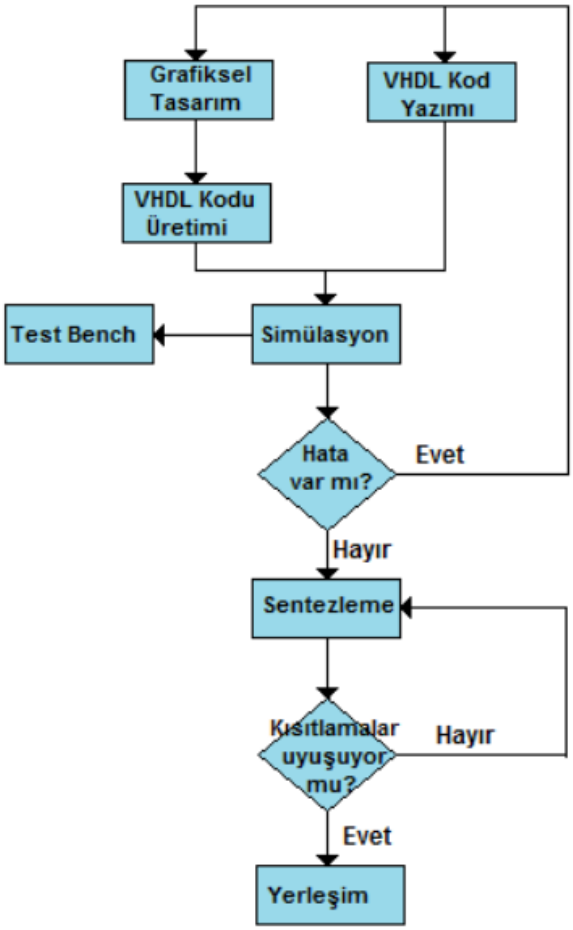
architecture behavioral of AND_OR is
  procedure AND_G(signal x,y:in std_logic;signal outp:out std_logic)is
  begin
    outp <= x and y;
  end AND_G;

  procedure OR_G(signal x,y:in std_logic;signal outp:out std_logic)is
  begin
    outp <= x or y;
  end OR_G;

begin
  AND_G(a,b,out_and);
  OR_G(a,b,out_or);
end behavioral;
```

4. BULGULAR

64 Bitlik FPGA tabanlı Aritmetik Mantık Birimi tasarımı işlem içeriğine göre üç alt birimin birleşimi şeklinde tasarlanmıştır. Tasarım akışı aşağıdan yukarı doğrudur (bottom to top design). Tasarım kaynak kodu VHDL donanım tanımlama dili ile yazılmış ve tasarımda davranışsal(behavioral) ve yapısal(structural) modelleme kullanılmıştır. Şekil 4.1'de VHDL tasarım akışı gösterilmiştir.



Şekil 4.1: VHDL tabanlı tasarım akışı[30]

Tasarımda ALTERA firmasının FPGA ailesi kullanıldığı için tasarım ortamı olarak yine ALTERA firmasına ait QURATUS II v13.0sp1 programı kullanılmıştır. Tasarımın simülasyonları ise MODELSIM-ALTERA v10.1d programı ile gerçekleştirilmiştir.

64 bit ALU yapısı sırası ile mantık birimi, aritmetik birim ve kaydırma-döndürme biriminden oluşturulmuştur. Bu kısımda bu birimlerin tasarımı adım adım anlatılacaktır.

4.1. 64 BİT MANTIK BİRİMİ TASARIMI

Mantık birimi içerisinde tanımlanan mantık işlemleri aşağıda sıralanmıştır.

- 64 bitlik AND işlemi
- 64 bitlik OR işlemi
- 64 bitlik XOR işlemi
- 64 bitlik NOT işlemi
- 64 bitlik NAND işlemi
- 64 bitlik NOR işlemi
- 64 bitlik XNOR işlemi

İşlemlerin her biri *component* olarak tanımlanmıştır ve her bir *component* yapısının iç süreci davranışsal modelleme ile ifade edilmiştir.

Her bir işlemin davranışı doğruluk tablolarının karnough haritaları kullanalar sadeleştirilmesi ile oluşturulmuştur. Aynı tasarım VHDL dili std_logic_1164 kütüphanesinde tanımlanmış mantık fonksiyonları kullanılarak da gerçekleştirilmiştir.

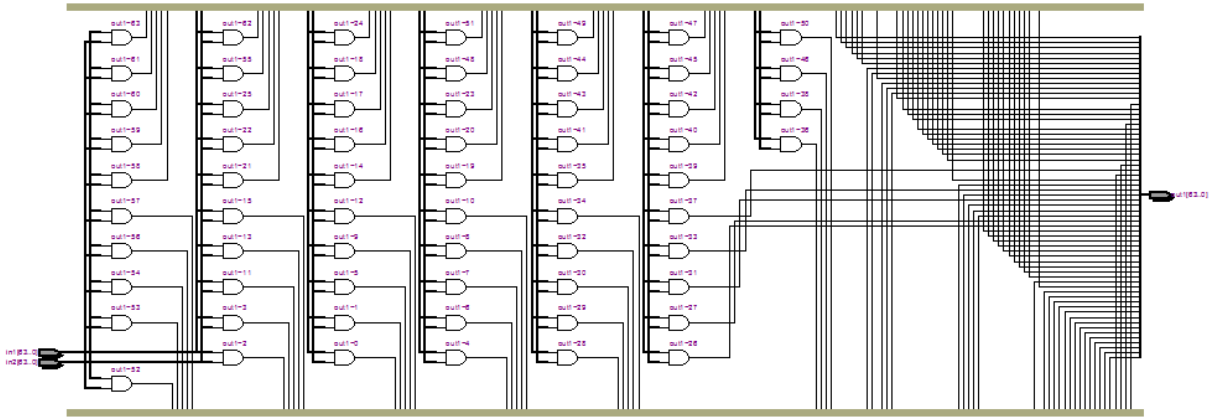
Örneğin aşağıdaki kod parçacığı 64 bitlik AND işlemini gerçekleştirmektedir.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
--64 BIT AND GATE
entity AND_64 is
--port and generic declerations
  generic(N:integer:=64);
  port    (in1,in2:in  std_logic_vector(N-1 downto 0) ;
           clk     :in  std_logic;
           out1    :out std_logic_vector(N-1 downto 0));
end AND_64;
architecture behavioral of AND_64 is
begin
 process (in1,in2,clk)
 begin
  for i in 0 to N-1 loop
   out1(i)<=in1(i) and in2(i);
  end loop;
 end process;
end behavioral;
```

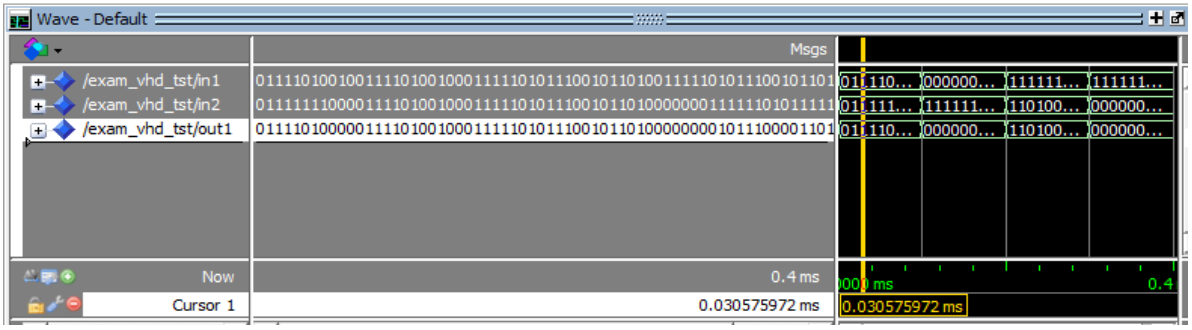
Kaynak kodunun derlenmesinin ardından;

Flow Summary	
Flow Status	Successful - Thu Nov 08 13:41:32 2018
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	exam
Top-level Entity Name	exam
Family	Cyclone II
Device	EP2C70F896C6
Timing Models	Final
Total logic elements	64 / 68,416 (< 1 %)
Total combinational functions	64 / 68,416 (< 1 %)
Dedicated logic registers	0 / 68,416 (0 %)
Total registers	0
Total pins	192 / 622 (31 %)
Total virtual pins	0
Total memory bits	0 / 1,152,000 (0 %)
Embedded Multiplier 9-bit elements	0 / 300 (0 %)
Total PLLs	0 / 4 (0 %)

Şekil 4.2: 64 Bit VE Bileşeni Derleme Raporu



Şekil 4.3: 64 Bit VE Bileşeni RTL Şeması



Şekil 4.4: 64 Bit VE Bileşeni Benzetim Sonucu

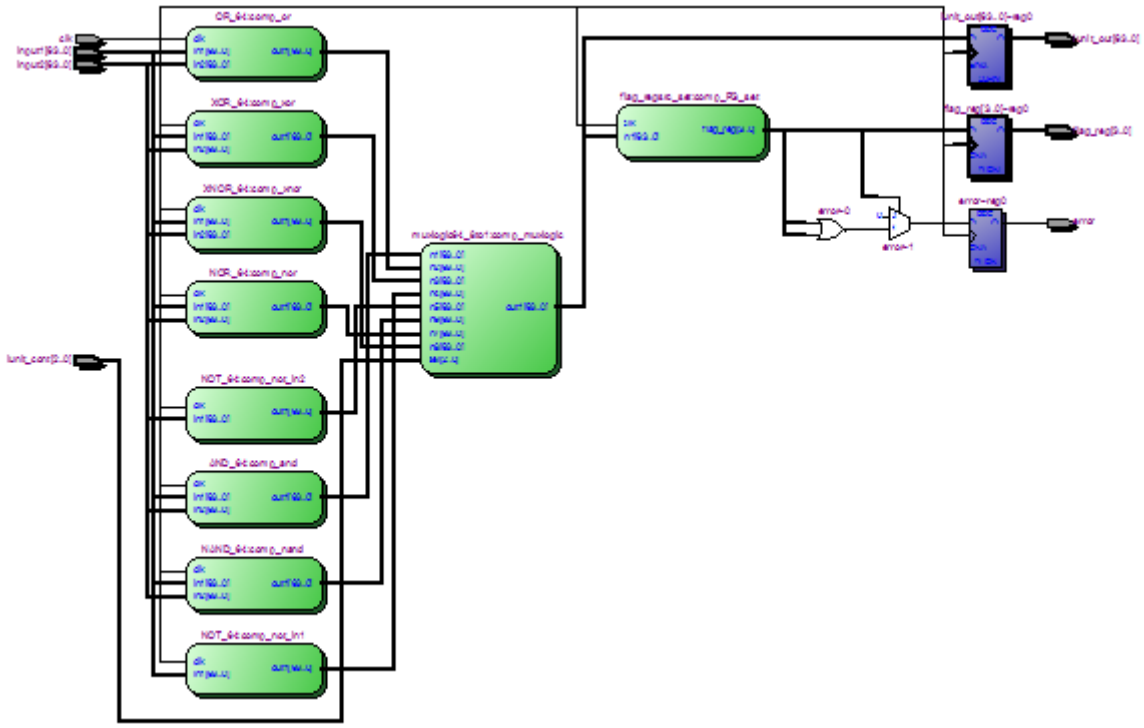
Diğer tüm mantık bileşenleri de benzer şekilde tasarlanmıştır.

4.1.1. 64 Bit Mantık Birimi Bulguları

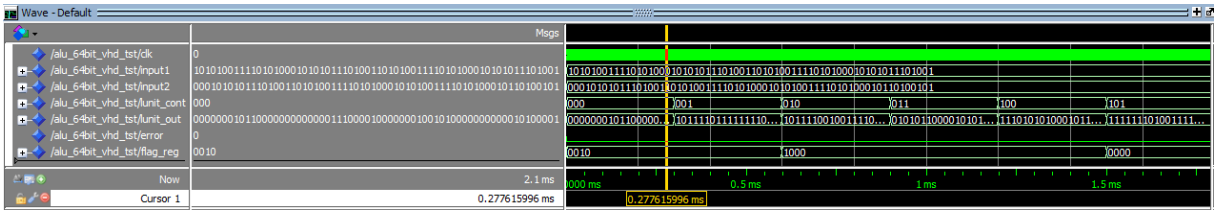
Mantık birimi için tanımlanan tüm bileşenler bir üst seviye tasarım olan *alu64_logic_unit* altında toplanmıştır.

Flow Summary	
Flow Status	Successful - Mon Nov 19 11:55:39 2018
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	alu_64bit
Top-level Entity Name	alu_64bit
Family	Cyclone II
Device	EP2C70F896C6
Timing Models	Final
Total logic elements	261 / 68,416 (< 1 %)
Total combinational functions	235 / 68,416 (< 1 %)
Dedicated logic registers	74 / 68,416 (< 1 %)
Total registers	74
Total pins	201 / 622 (32 %)
Total virtual pins	0
Total memory bits	0 / 1,152,000 (0 %)
Embedded Multiplier 9-bit elements	0 / 300 (0 %)
Total PLLs	0 / 4 (0 %)

Şekil 4.5: 64 Bit Mantık Birimi Derleme Raporu



Şekil 4.6: 64 Bit Mantık Birimi RTL Şeması



Şekil 4.7: 64 Bit Mantık Birimi Benzetim sonucu

4.2. 64 BİT ARİTMETİK BİRİM TASARIMI

Aritmetik Birim içerisinde tanımlanan işlemler aşağıdaki gibidir.

- 64 bitlik İşaretli Toplayıcı
- 64 bitlik İşaretli Çıkarıcı
- 64 bitlik İşaretli Çarpıcı
- 64 bitlik İşaretli Bölücü
- 64 bitlik İşaretli Mod Alıcı

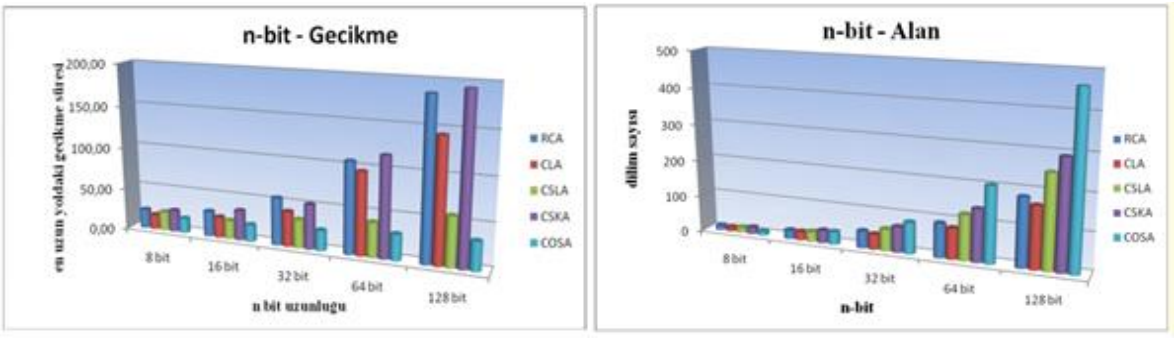
Her bir işlem için farklı bir algoritma kullanılmıştır.

4.2.1. 64 Bit İşaretli Toplayıcı Algoritması

64 Bit İşaretli Toplayıcı tasarımı için çeşitli toplayıcı algoritmaları incelenmiştir. İncelemesi yapılan klasik toplayıcı devreleri sırası ile

- Elde Dalgalı Toplayıcı(RCA),
- Elde Öngörü Toplayıcı(CLA),
- Elde Seçimli Toplayıcı(CSLA),
- Elde Atlamalı Toplayıcı(CSKA),
- Koşullu Toplayıcı(COSA)

Şeklindedir. Bu klasik toplayıcılara ilave olarak yine bu algoritmaların 2'li ya da 3'lü kombinasyonlarından oluşturulan algoritmalar da mevcuttur. Araştırma sonucunda bu klasik toplama devreleri hız ve alan kullanımı açısından aşağıdaki özellikleri göstermektedir.



Şekil 4.8: Klasik toplayıcıların gecikmesi ve kapladığı alan karşılaştırması[9]

Bu özellikler göze alındığında ALU içerisinde yüksek hızda toplama işlemini gerçekleştirebilmek için, 64 bit toplayıcı bileşeninin tasarımında minimum gecikmeli COSA(conditional sum adder) toplama algoritması kullanılmıştır.

Tüm toplayıcı devrelerin temelinde yarım toplayıcılardan oluşmuş tam toplayıcı yapıları vardır. Dolayısı ile tam toplayıcı(FA) yapısı tasarımımız içerisinde *component* olarak tanımlanmıştır.

Component Full_adder:

Birer bitlik iki sayının toplamı yarım toplayıcılar ile gerçekleştirilir. Tüm toplayıcı algoritmalarının temel taşını oluşturan tam toplayıcı yapıları ise iki yarım toplayıcının birleşiminden oluşturulur. Seçtiğimiz 64 bitlik toplayıcı algoritması için kullanacağımız full_adder component'i, tam toplayıcı doğruluk tablosu ve karnough haritaları ile sadeleştirilmiş fonksiyon yardımı ile yazılmıştır. Girişler *in1*, *in2*, ve *in3* olmak üzere;

Toplam çıkışı için elde edilen sadeleştirilmiş fonksiyon,

$$S = in1\overline{in2}\,in3 + \overline{in1}\,in2\,in3 + in1\,in2\,in3 + in1\,in2\overline{in3} = in1\ xor\ in2\ xor\ in3 \quad (4.1)$$

Elde çıkışı için elde edilen sadeleştirilmiş fonksiyon ise,

$$C = in1\,in2 + in1\,in3 + in2\,in3 \quad (4.2)$$

Sadeleştirilmiş fonksiyonlar doğrultusunda yazılan full_adder component'i aşağıdaki gibidir.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity full_adder is
 port(in1,in2: in std_logic;
      cin1: in std_logic:='0';
      clk : in std_logic;
      sum1, cout1 : out std_logic);
end full_adder;
architecture behavioral of full_adder is
 signal tempsum1       :std_logic;
 signal tempcarry1     :std_logic;
 signal tempcarry2     :std_logic;
 signal sec_hlfaddr_in1:std_logic;
 signal or_in1         :std_logic;
 signal or_in2         :std_logic;

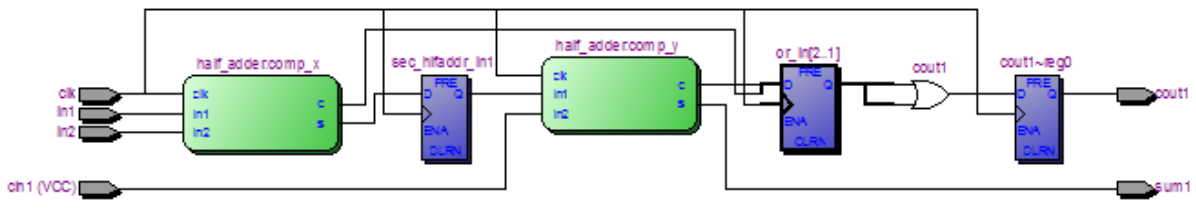
 component half_adder
 port(in1,in2,clk  : in  std_logic ;
      s, c     : out std_logic);
 end component;

begin
 comp_x: half_adder port map(in1,in2,clk,tempsum1,tempcarry1);
 process(in1,in2,clk,or_in1,tempcarry1,sec_hlfaddr_in1,tempsum1)
 begin
 if clk='1' and clk'event then
 or_in1          <= tempcarry1;
 sec_hlfaddr_in1 <= tempsum1  ;
 end if;
 end process;

 comp_y: half_adder port map (sec_hlfaddr_in1,cin1,clk,sum1,tempcarry2);

 process(in1,in2,clk,or_in2,tempcarry2)
 begin
 if clk='1' and clk'event then
 or_in2          <= tempcarry2;
 cout1<= or_in1 or or_in2;
 end if;
 end process;
end behavioral;
```

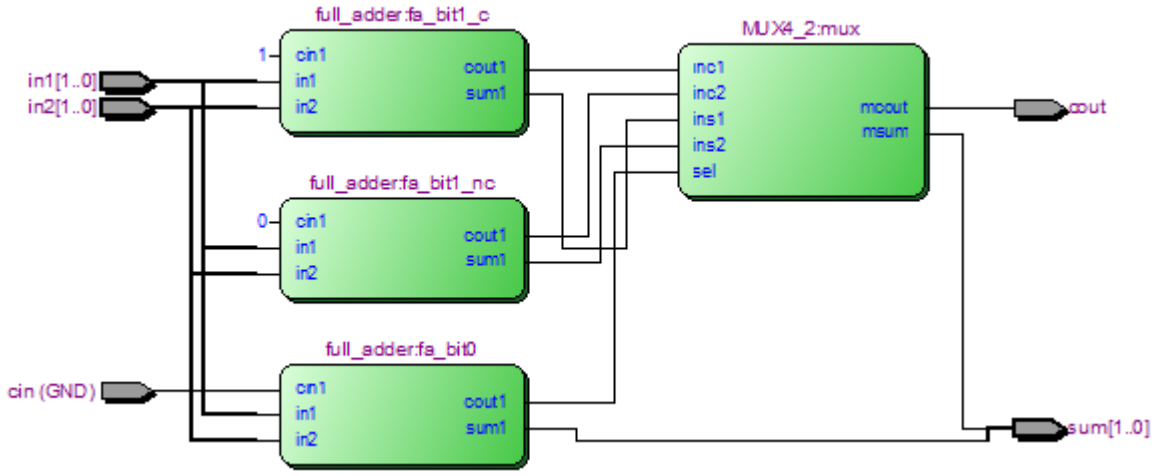
Bileşenin RTL görüntüsü ise;



Şekil 4.9: Full_adder bileşeni RTL yapısı

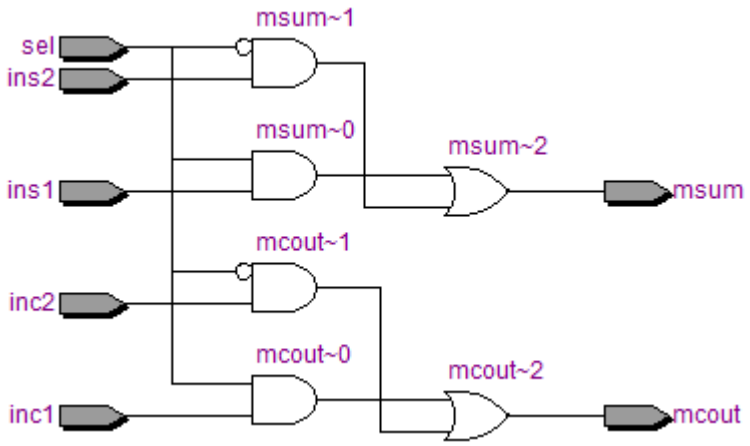
Full_adder component'i içerisinde çağrılan half_adder component'i aynı teknik ile oluşturulmuş birer bitlik iki sayının toplamını sağlayan yarım toplayıcıdır.

Temel COSA algoritması, elde bitinin “0” ve “1” olma durumuna göre iki ayrı toplama işlemi daha yapar multiplexer yapıları ile elde bitinin “0” mı yoksa “1” mi olduğu seçilir ve doğru toplam elde edilir. Aşağıda iki bitlik bir COSA yapısının blok diyagramı verilmiştir.



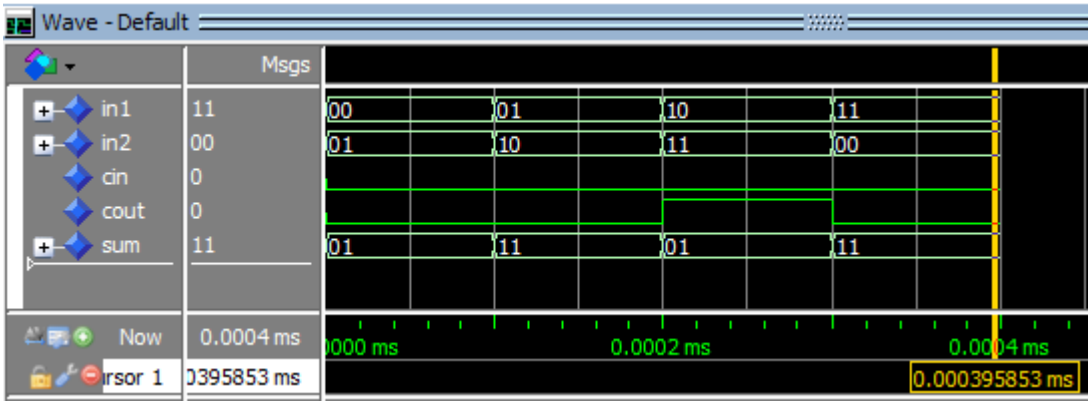
Şekil 4.10: 2 bitlik COSA RTL şeması

İki bitlik COSA yapısında girişlerin LSB(less significant bit) biti toplamı cin girişi “0” alınıp tam toplayıcı ile hesaplanır. Tam toplayıcı elde çıkışı multiplexer yapısının seçim girişidir ve iki bitlik girişin MSB biti toplamı yapılırken LSB bitinden gelen elde değerine göre diğer iki tam toplayıcıdan biri seçilir ve hesap yapılır. Multiplexer içeriği aşağıdaki gibidir.



Şekil 4.11: 4:2 MUX yapısı

Aşağıda 2 bitlik COSA yapısının simülasyon sonucu verilmiştir.



Şekil 4.12: 2 Bitlik COSA toplayıcı benzetim sonucu

64 bitlik toplayıcı bileşeni bu 32 bitlik 3 COSA yapısı ile oluşturulmuştur. 32 bitlik COSA yapısı 2 bitlik COSA yapısının farklı çoğullayıcı yapıları kullanılarak oluşturulmuş bir uzantısıdır. 64 bitlik

64 Bitlik COSA algoritmalı toplayıcı için tasarlanan bileşenler aşağıdaki gibidir.

- Yarım toplayıcı ile oluşturulan tam toplayıcı bileşeni
- 2, 4, 8, 16 ve 32 bitlik COSA bileşenleri
- 4:2, 6:3, 8:4, 10:5, 20:10, 34:17 ve 66:33 Multiplexer bileşenleri

İşaretsiz sayılar için tasarlanan 64 Bitlik COSA algoritmalı toplayıcı kodu:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

entity ADD_64 is
    generic (P:integer:=64);
    port (in1,in2 : in  std_logic_vector(P-1 downto 0);
          cin     : in  std_logic:='0';
          clk     : in  std_logic;
          sum     : out std_logic_vector(P-1 downto 0);
          cout    : out std_logic);
end ADD_64;
architecture behavioral of ADD_64 is
    signal m_sel              : std_logic;
    signal m_in1, m_in2       : std_logic_vector(32 downto 0);
    signal mux_o              : std_logic_vector(32 downto 0);
    signal sumc               : std_logic_vector(31 downto 0);
    constant c: std_logic := '1';

    component COSA32 is
    port (in1,in2 : in  std_logic_vector(31 downto 0);
          cin     : in  std_logic:='0';
          clk     : in  std_logic;
          sum     : out std_logic_vector(31 downto 0);
```

```
          cout    : out std_logic);
    end component;

    component mux66_33 is
    port(sel    : in std_logic;
         in1    : in std_logic_vector(32 downto 0);
         in2    : in std_logic_vector(32 downto 0);
         clk    : in std_logic;
         muxout : out std_logic_vector(32 downto 0));
    end component;

begin
    fa_bit01 : COSA32
    PORT MAP( in1(31 downto 0),in2(31 downto 0),cin,clk, sumc(31 downto 0), m_sel);

    fa_bit01_c:COSA32
    PORT MAP( in1(63 downto 32), in2(63 downto 32), c,clk, m_in1(31 downto 0), m_in1
(32));

    fa_bit01_nc:COSA32
    PORT MAP( in1(63 downto 32), in2(63 downto 32), not c,clk, m_in2(31 downto 0),
m_in2(32));

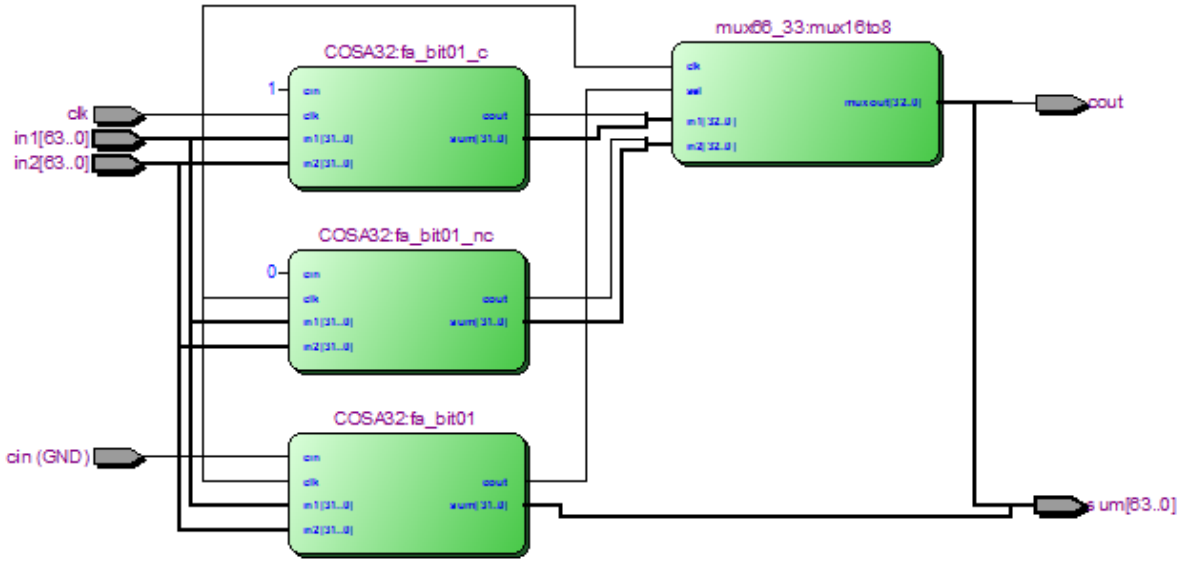
    mux16to8: MUX66_33
    PORT MAP( m_sel, m_in1, m_in2,clk, mux_o);

    sum(31 downto 0)<=sumc(31 downto 0);
    sum(63 downto 32)<=mux_o(31 downto 0);
    cout<=mux_o(32);

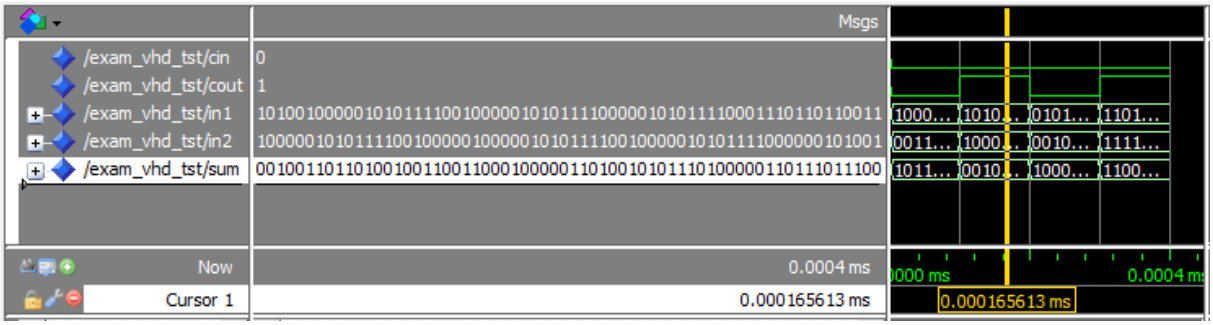
end behavioral;
```

Flow Summary	
Flow Status	Successful - Thu Nov 08 13:34:03 2018
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	exam
Top-level Entity Name	exam
Family	Cyclone II
Device	EP2C70F896C6
Timing Models	Final
Total logic elements	250 / 68,416 (< 1 %)
Total combinational functions	250 / 68,416 (< 1 %)
Dedicated logic registers	0 / 68,416 (0 %)
Total registers	0
Total pins	194 / 622 (31 %)
Total virtual pins	0
Total memory bits	0 / 1,152,000 (0 %)
Embedded Multiplier 9-bit elements	0 / 300 (0 %)
Total PLLs	0 / 4 (0 %)

Şekil 4.13: 64 Bit İşaretsiz Toplayıcı Bileşeni Derleme Raporu



Şekil 4.14: 64 Bitlik Toplayıcı Bileşeni RTL Şeması



Şekil 4.15: 64 Bitlik Toplayıcı Bileşeni Benzetim Sonucu

Şimdiye kadarki olan tasarımda toplama işlemi yalnızca işaretli sayıları kapsayacak şekilde yapılmıştır.

İşaretli sayılarda işlem göz önünde bulundurulduğunda, birkaç önemli kavrama ve düzenlemeye dikkat etmek gerekir. Bunlardan ilki "elde" ve "taşma" kavramları arasındaki fark, ikincisi negatif sayıların ifadesi ve MSB bitlerinin işaret biti olması sebebi ile ortaya çıkan kısıtlamalardır.

İşaretli sayıların gösterimi için 3 farklı yöntem kullanılır. Bunlar işaret biti ile gösterim, 1'in tümleyeni şeklinde gösterim ve 2'nin tümleyeni şeklinde gösterimdir. Bu üç yöntem arasından 2'nin tümleyeni şeklinde gösterimde, "0" sayısının tek bir ifadesi olması ve toplama- çıkarma işlemlerindeki işaret bitinde meydana gelen karışıklığın giderilmiş olması sebebi ile en çok

kullanılan yöntemdir. Bu tasarımda da negatif sayılar 2'nin tümleyeni şeklinde ifade edilmiştir.

İşaretli sayılarda MSB bitleri işaret bitidir ve MSB biti "0" ise sayı pozitif, "1" ise sayı negatiftir. MSB bitinin işaret biti olması, kullanılan sayı boyutu ile ifade edilebilecek maksimum ve minimum sayıyı kısıtlar. Tasarımımız 64 bit olduğundan dolayı bu boyutta gösterilebilecek maksimum ve minimum sayılar aşağıdaki gibidir.

64 bit pozitif en büyük sayı:

0111111111111111111111111111111111111111111111111111111111111111

64 bit negatif en büyük sayı:

1000000000000000000000000000000000000000000000000000000000000001

Dolayısı ile 64 bit iki sayının toplamı maksimum olarak, 64 bitte ifade edilebilen en büyük pozitif sayıyı aşamaz ve yine aynı şekilde 64 bit iki negatif sayının toplamı da, 64 bitte ifade edilebilen en büyük negatif sayıyı aşamaz.

İşaretsiz sayılarda toplama işlemi için kullanılan "elde" kavramı ve çıkarma işlemlerinde kullanılan "ödünç" kavramı "taşma" kavramı ile aynı durumu ifade eder. İşaretli sayılarda ise "taşma" kavramı diğer iki kavramdan farklı bir amaca hizmet eder.

İşaretli sayılarda taşma:

Taşma kavramı tam olarak elde veya ödünç kavramları gibi hesaplanmaz. Örnekle anlatılmak gerekir ise iki negatif sayının toplamı pozitif sonuç veriyor ise oluşan elde taşmayı ifade eder ya da iki negatif sayının toplamı sonucu pozitif bir sayı ise yine aynı durum geçerlidir. İşlem sonucunda taşmanın oluşması çalışılan boyutta kullanılabilecek maksimum sayı aralığının dışına çıkıldığını gösterir ve işlem sonucu doğru sonucu göstermez. Bunun yerine taşma bayrağı kullanılarak kullanılabilecek aralığın aşıldığı uyarısı verilir. İki sayının toplanması sonucu oluşan 65. Bit sayısının elde olarak ifade edilmesi için toplam sonucunun geriye kalan 64 bitte doğru ifade edilmiş olması geçerlidir. Aynı durum çıkarma işleminde ödünç kavramı için geçerlidir. İşaret biti de dahil olmak üzere sonuç doğru ise elde biti sonuçtan atılır.

Tasarım işaretli sayılar için güncellendiğinde, yeni tasarıma “taşma” kontrolü ve taşma çıkışı eklenmiştir. Bu eklemeler process ve if-then gibi yapılar kullanılarak gerçekleştirilmiştir.

Toplama işleminde taşma durumları:

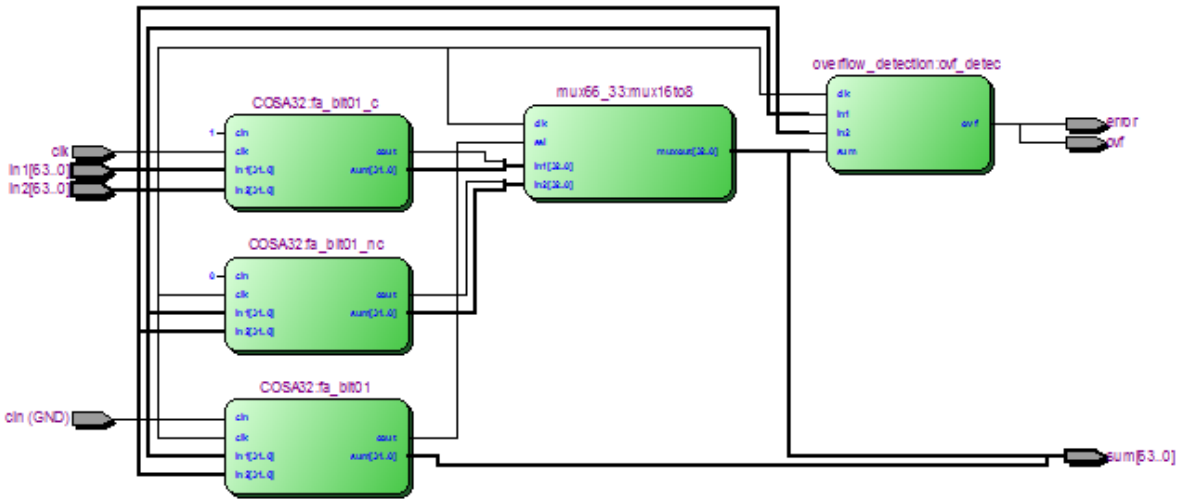
- 64 bitlik iki negatif sayının toplamı sonucu işaret bitinin (64. bit) “0” ı göstermesi yani sonucun pozitif bulunması
- 64 bitlik iki pozitif sayının toplamı sonucu işaret bitinin (64. bit) “1” i göstermesi yani sonucun negatif bulunması

Bu iki durum sonucun oluşan elde biti taşma olarak adlandırılır. Diğer durumlarda ise oluşan elde biti normal eldedir ve değerlendirmeye alınmaz.

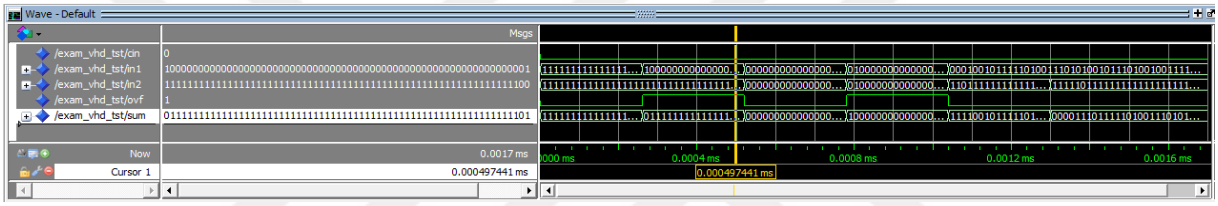
Bu bilgiler doğrultusunda düzenlenen işaretli sayılar için toplama bileşeni derlendiğinde;

Flow Summary	
Flow Status	Successful - Tue Nov 13 22:01:52 2018
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	exam
Top-level Entity Name	exam
Family	Cyclone II
Device	EP2C70F896C6
Timing Models	Final
Total logic elements	260 / 68,416 (< 1 %)
Total combinational functions	260 / 68,416 (< 1 %)
Dedicated logic registers	0 / 68,416 (0 %)
Total registers	0
Total pins	194 / 622 (31 %)
Total virtual pins	0
Total memory bits	0 / 1,152,000 (0 %)
Embedded Multiplier 9-bit elements	0 / 300 (0 %)
Total PLLs	0 / 4 (0 %)

Şekil 4.16: 64 Bit İşaretli Toplayıcı Bileşeni Derleme Raporu



Şekil 4.17: İşaretli Sayılar için 64 Bit Toplayıcı Bileşeni RTL Şeması



Şekil 4.18: İşaretli Sayılar için 64 Bit Toplayıcı Bileşeni Benzetim Sonucu

4.2.2. 64 Bit İşaretli Çıkarıcı Algoritması

64 Bit Toplayıcı bileşeni tasarımından sonra çıkarıcı tasarımı bir hayli kolaydır. Aslında çıkarıcı yapısı herhangi bir sayıya negatif bir sayıyı ekleme işlemidir. Bu tasarımda çıkarıcı operatörü seçimi yapıldığında çıkarma işlemi birinci girişten ikinci girişin çıkarılması şeklinde tasarlanmıştır. Dolayısı ile ikinci girişin 2'ye tümleyeni alınacaktır.

64 Bit çıkarıcı tasarımı için daha önce tasarladığımız toplayıcı bileşeni ikinci girişin 2'nin tümleyeni ile ifadesi eklenerek düzenlenmiş ve çıkarıcı bileşen oluşturulmuştur. Düzenlemede bir diğer konu ise elde çıkışının ödünç alma olarak düzenlenmesidir.

Bir ikili sayıyı 2'nin tümleyeni şeklinde ifade etme:

İkili sayının önce 1'e tümleyeni alınır. Daha sonra elde edilen yeni sayı 1 ile toplandığında sayının 2'ye tümleyeni bulunur.

VHDL ile davranışsal ifadesi ise:

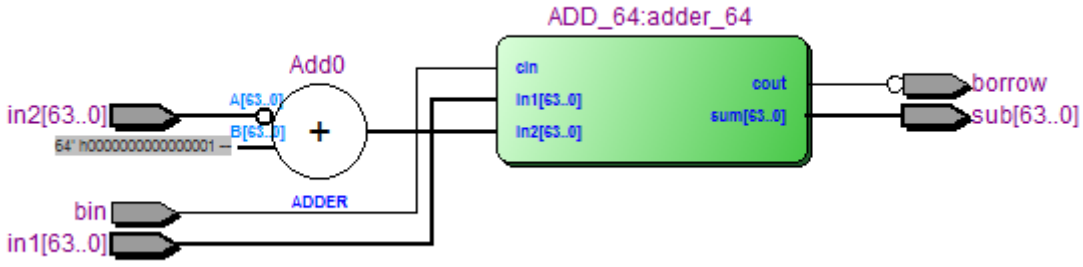
```
sayi<= not sayi +1 --2's compliment
```

Şeklindedir. Bu bilgiler doğrultusunda 64 bit COSA algoritmalı toplayıcı bileşenin kullanılması ile oluşturulan 64 bit çıkarıcı bileşen kaynak kodu aşağıdaki gibidir.

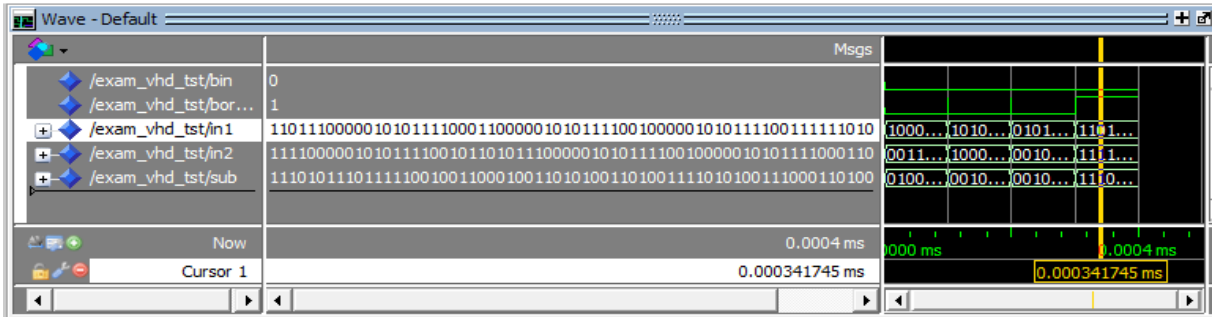
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity SUB_64 is
   generic (P:integer:=64);
   port (in1,in2 : in  std_logic_vector(P-1 downto 0);
         bin     : in  std_logic:='0';
         clk     : in  std_logic;
         sub     : out std_logic_vector(P-1 downto 0);
         borrow  : out std_logic);
end SUB_64;
architecture behavioral of SUB_64 is
   signal n_in2: std_logic_vector(P-1 downto 0);
   signal brrow: std_logic;

   component ADD_64 is
   generic (P:integer:=64);
   port (in1,in2 : in  std_logic_vector(P-1 downto 0);
         cin     : in  std_logic:='0';
         clk     : in  std_logic;
         sum     : out std_logic_vector(P-1 downto 0);
         cout    : out std_logic);
   end component;
begin
   n_in2<= not in2 + 1; --2's compliment of in2
   adder_64: ADD_64 PORT MAP( in1,n_in2,bin,clk,sub,brrow);
   borrow<=not brrow;   --borrow=not carry of ADD_64
end behavioral;
```



Şekil 4.19: 64 Bit Çıkarıcı Bileşeni RTL Şeması

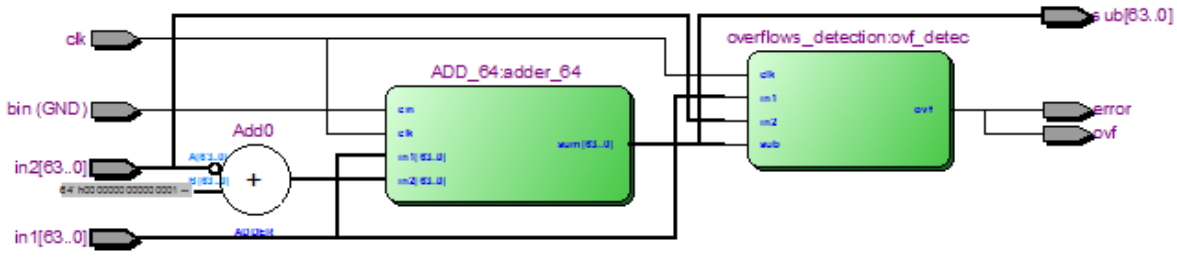


Şekil 4.20: 64 Bit Çıkarıcı Bileşeni Benzetim Sonucu

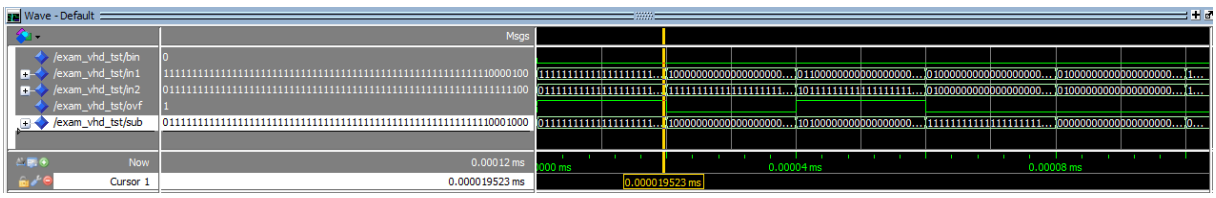
İşaretli sayılar için bileşenin düzenlenmesinin ardından derleme sonucunda;

Flow Summary	
Flow Status	Successful - Thu Nov 08 13:20:42 2018
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	exam
Top-level Entity Name	exam
Family	Cyclone II
Device	EP2C70F896C6
Timing Models	Final
Total logic elements	322 / 68,416 (< 1 %)
Total combinational functions	322 / 68,416 (< 1 %)
Dedicated logic registers	0 / 68,416 (0 %)
Total registers	0
Total pins	194 / 622 (31 %)
Total virtual pins	0
Total memory bits	0 / 1,152,000 (0 %)
Embedded Multiplier 9-bit elements	0 / 300 (0 %)
Total PLLs	0 / 4 (0 %)

Şekil 4.21: Tasarımın Derleme Raporu



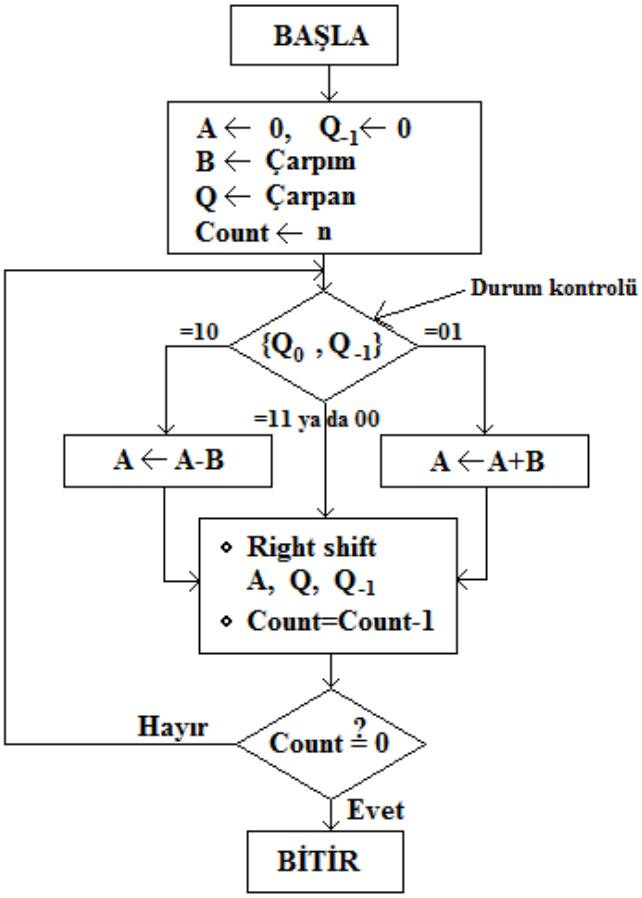
Şekil 4.22: 64 İşaretli Sayılar için 64 Bit Çıkarıcı Bileşeni RTL Şeması



Şekil 4.23: 64 İşaretli Sayılar için 64 Bit Çıkarıcı Bileşeni Benzetim Sonucu

4.2.3. 64 Bit İşaretli Çarpıcı Algoritması

Çarpıcı bileşeni için işaretli sayıların çarpımında büyük kolaylık sağlayan Booth's Algoritması kullanılmıştır. Algoritma akışı aşağıdaki gibidir.



Şekil 4.24: Booth's Çarpma Algoritması Akışı

Bu algoritma ile işaretli sayının çarpımını sağlayacak kaynak kodu aşağıdaki gibi yazılmıştır.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity MULTS_64 is
  generic(N  :integer:=64);
  port(M     :in  std_logic_vector(N-1 downto 0)   ;
       Q     :in  std_logic_vector(N-1 downto 0)   ;
       clk   :in  std_logic;
       error :out std_logic;
       carpim:out std_logic_vector(N-1 downto 0));
end MULTS_64;

architecture behavioral of MULTS_64 is

  shared variable temp_M   :std_logic_vector(N-33 downto 0);
  shared variable A        :std_logic_vector(N-33 downto 0);
  signal          Q_new    :std_logic_vector(N-32 downto 0)  ;
  shared variable A_Q_new  :std_logic_vector(N downto 0);
  shared variable temp_sum :std_logic_vector(N-33 downto 0);
  shared variable cntrl1   :std_logic;
  shared variable cntrl2   :std_logic;
  shared variable t_error  :std_logic;
  shared variable t_carpim :std_logic_vector(N-1 downto 0);

  function control1(input : std_logic_vector) return std_logic is
  variable counter:integer:= 0;
      begin
      for i in 32 to N-1 loop
      if(input(i) = '1') then
        counter := counter + 1;
      end if;
     end loop;
   if(counter> 0) then
      return '1';
   end if;
   return '0';
  end control1;

  function control0(input : std_logic_vector) return std_logic is
  variable counter:integer:= 0;
      begin
      for i in 32 to N-1 loop
      if(input(i) = '0') then
        counter := counter + 1;
      end if;
     end loop;
   if(counter> 0) then
      return '1';
   end if;
   return '0';
  end control0;
begin
  process(Q,M,Q_new,clk)
  begin
  if clk='1' and clk'event then
   if M(N-1)='1' and Q(N-1)='1' then
```

```
      cntrl1:=control0(M);
      cntrl2:=control0(Q);
      if cntrl1='1' or cntrl2='1' then
      t_carpim:=(others=>'1');
      t_error:='1';
      else
      t_error:='0';
      A      := (others=>'0');
      Q_new  <= Q(31 downto 0) & '0'      ;
      A_Q_new:= A & Q_new    ;

    for i in 0 to N-33 loop

     if A_Q_new(1 downto 0)="00" or A_Q_new(1 downto 0)="11" then

        A_Q_new     := to_stdlogicvector(to_bitvector(std_logic_vector(A_Q_new)) sra
 1);

     elsif A_Q_new(1 downto 0)="10"  then

     --M'in 2'nin tümleyeni şeklinde ifadesi
       temp_M:= (not M(31 downto 0))+1;

     --Yeni A_Q_new ifadesi
       temp_sum := A_Q_new(N downto N-31)+temp_M;
       A_Q_new  := temp_sum & A_Q_new(N-32 downto 0);
       A_Q_new  := to_stdlogicvector(to_bitvector(std_logic_vector(A_Q_new)) sra 1);

    elsif A_Q_new(1 downto 0)="01"  then

       temp_sum :=A_Q_new(N downto N-31)+M(31 downto 0);
       A_Q_new  := temp_sum & A_Q_new(N-32 downto 0);
       A_Q_new  := to_stdlogicvector(to_bitvector(std_logic_vector(A_Q_new)) sra 1
);
    end if  ;
     t_carpim:= A_Q_new(N downto 1);
   end loop;
  end if;
 elsif M(N-1)='0' and Q(N-1)='1' then
      cntrl1:=control1(M);
      cntrl2:=control0(Q);
      if cntrl1='1' or cntrl2='1' then
      t_carpim:=(others=>'1');
      t_error:='1';
      else
      t_error:='0';
      A      := (others=>'0');
      Q_new  <= Q(31 downto 0) & '0'      ;
      A_Q_new:= A & Q_new    ;

       for i in 0 to N-33 loop

        if A_Q_new(1 downto 0)="00" or A_Q_new(1 downto 0)="11" then

           A_Q_new     := to_stdlogicvector(to_bitvector(std_logic_vector(A_Q_new))
sra 1);

        elsif A_Q_new(1 downto 0)="10"  then

        --M'in 2'nin tümleyeni şeklinde ifadesi
           temp_M:= (not M(31 downto 0))+1;
```

```
        --Yeni A_Q_new ifadesi
          temp_sum := A_Q_new(N downto N-31)+temp_M;
          A_Q_new  := temp_sum & A_Q_new(N-32 downto 0);
          A_Q_new  := to_stdlogicvector(to_bitvector(std_logic_vector(A_Q_new)) sra
1);
        elsif A_Q_new(1 downto 0)="01"  then

         temp_sum :=A_Q_new(N downto N-31)+M(31 downto 0);
         A_Q_new  := temp_sum & A_Q_new(N-32 downto 0);
         A_Q_new  := to_stdlogicvector(to_bitvector(std_logic_vector(A_Q_new)) sra 1
);
        end if  ;
        t_carpim:= A_Q_new(N downto 1);
      end loop;
     end if;
 elsif M(N-1)='1' and Q(N-1)='0' then
      cntrl1:=control0(M);
      cntrl2:=control1(Q);
      if cntrl1='1' or cntrl2='1' then
      t_carpim:=(others=>'1');
      t_error:='1';
      else
      t_error:='0';
      A      := (others=>'0');
      Q_new  <= Q(31 downto 0) & '0';
      A_Q_new:= A & Q_new    ;

       for i in 0 to N-33 loop

        if A_Q_new(1 downto 0)="00" or A_Q_new(1 downto 0)="11" then

           A_Q_new     := to_stdlogicvector(to_bitvector(std_logic_vector(A_Q_new))
sra 1);

        elsif A_Q_new(1 downto 0)="10"  then

        --M'in 2'nin tümleyeni şeklinde ifadesi
           temp_M:= (not M(31 downto 0))+1;

        --Yeni A_Q_new ifadesi
          temp_sum := A_Q_new(N downto N-31)+temp_M;
          A_Q_new  := temp_sum & A_Q_new(N-32 downto 0);
          A_Q_new  := to_stdlogicvector(to_bitvector(std_logic_vector(A_Q_new)) sra
1);

        elsif A_Q_new(1 downto 0)="01"  then

         temp_sum :=A_Q_new(N downto N-31)+M(31 downto 0);
         A_Q_new  := temp_sum & A_Q_new(N-32 downto 0);
         A_Q_new  := to_stdlogicvector(to_bitvector(std_logic_vector(A_Q_new)) sra 1
);
        end if  ;
        t_carpim:= A_Q_new(N downto 1);
      end loop;
    end if;
 else
     cntrl1:=control1(M);
     cntrl2:=control1(Q);
     if cntrl1='1' or cntrl2='1' then
     t_carpim:=(others=>'1');
     t_error:='1';
```

```
        else
        t_error:='0';
        A       := (others=>'0');
        Q_new   <= Q(31 downto 0) & '0';
        A_Q_new:= A & Q_new     ;

         for i in 0 to N-33 loop
         if A_Q_new(1 downto 0)="00" or A_Q_new(1 downto 0)="11" then

            A_Q_new      := to_stdlogicvector(to_bitvector(std_logic_vector(A_Q_new))
sra 1);

         elsif A_Q_new(1 downto 0)="10"  then

         --M'in 2'nin tümleyeni şeklinde ifadesi
            temp_M:= (not M(31 downto 0))+1;

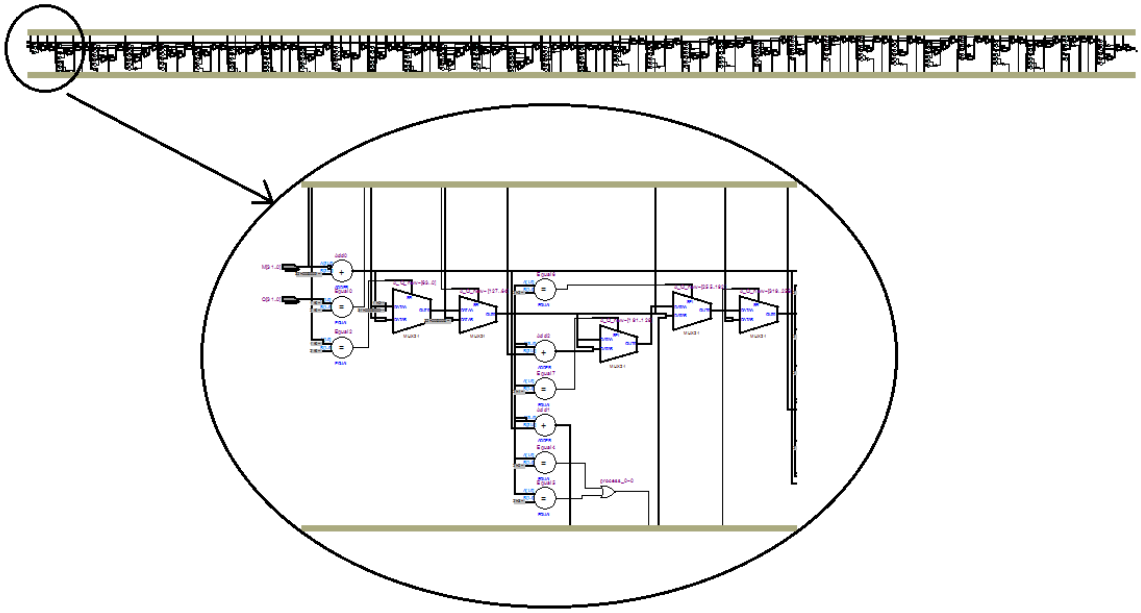
         --Yeni A_Q_new ifadesi
           temp_sum := A_Q_new(N downto N-31)+temp_M;
           A_Q_new  := temp_sum & A_Q_new(N-32 downto 0);
           A_Q_new  := to_stdlogicvector(to_bitvector(std_logic_vector(A_Q_new)) sra
1);

         elsif A_Q_new(1 downto 0)="01"  then

          temp_sum :=A_Q_new(N downto N-31)+M(31 downto 0);
          A_Q_new  := temp_sum & A_Q_new(N-32 downto 0);
          A_Q_new  := to_stdlogicvector(to_bitvector(std_logic_vector(A_Q_new)) sra 1
);
         end if  ;
         t_carpim:= A_Q_new(N downto 1);
       end loop;
     end if;
    end if;
   carpim<=t_carpim;
   error <=t_error;
  end if;
  end process;
end behavioral;
```

Burada dikkat edilmesi gereken kısım girişlerin 32 bit olduğudur. Bunun sebebi Aritmetik mantık birimi işlem çıkışının 64 bit olmasıdır. Bu sebeple çarpma işleminde girişlere gelen sayıların 32 bit uzunluğunda olması için 64 bitlik girişlerin 0-31 bit aralığı işleme alınacaktır. Eğer sayı işaretli olarak 32 bit ile ifade edilebilecek sayı aralığını aşıyor ise sayıların çarpımı 32 biti aşacağından tasarıma bu durumun ifadesi için hata çıkışı eklenmiştir. Hata çıkışının belirlenmesi için tasarıma veri girişlerinin ilk 32 bitinden sonraki bitlerinde, sayı negatif ise "0" değerini, pozitif ise "1" değerini kontrol eden fonksiyonlar eklenmiştir. Önemli bir diğer noktada veri girişlerinin 31. Biti işaret biti olarak alınmıştır. Dolayısı ile 64 bitlik sayı pozitif olsa bile sayının 31. Biti "1" değerinde ise sayı negatif olarak işleme alınacaktır.

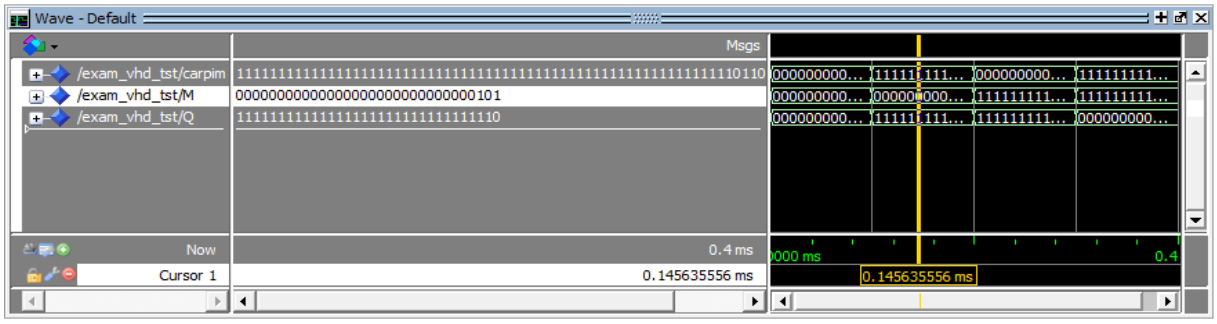
Yazılan kaynak kodunun derlenmesinin ardından;



Şekil 4.25: 64 Bit Çarpıcı Bileşeni RTL Şeması

Flow Summary	
Flow Status	Successful - Thu Nov 08 13:11:09 2018
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	exam
Top-level Entity Name	exam
Family	Cyclone II
Device	EP2C70F896C6
Timing Models	Final
Total logic elements	4,032 / 68,416 (6 %)
Total combinational functions	4,032 / 68,416 (6 %)
Dedicated logic registers	0 / 68,416 (0 %)
Total registers	0
Total pins	128 / 622 (21 %)
Total virtual pins	0
Total memory bits	0 / 1,152,000 (0 %)
Embedded Multiplier 9-bit elements	0 / 300 (0 %)
Total PLLs	0 / 4 (0 %)

Şekil 4.26: Tasarımın Derleme Raporu

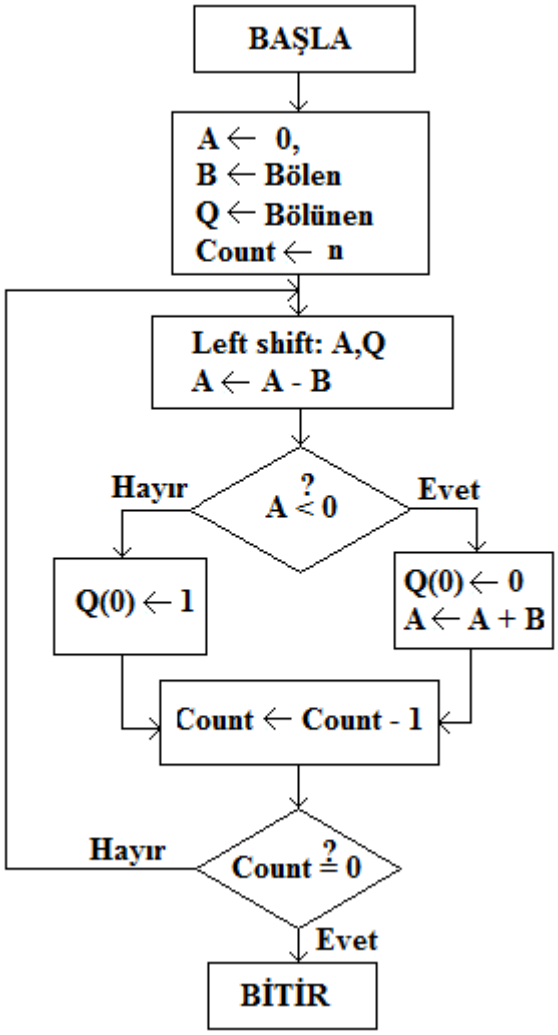


Şekil 4.27: 64 Bit Çarpıcı Bileşeni Benzetim Sonucu

4.2.4. 64 Bit İşaretli Bölücü Algoritması

Bölücü bileşeni için “Restoring Division” algoritması kullanılmıştır. Algoritmada işlenecek bit sayısı kadar çalışacak bir döngü tanımlanır. Algoritma işlenip 64 bitlik döngü tamamlandığında oluşan **Q** bölümü ve **A** kalanı ifade eder. Tasarımda çıkış olarak sadece bölüm alınmıştır.

Bölme algoritmasında dikkat edilmesi gereken kısım bölenin “0” olması ve bölen ve bölünenin aynı anda “0” olması durumudur. Bu durumlar sırası ile tanımsızlık ve belirsizlik durumlarıdır ve tasarımda bu durular söz konusu ise hata çıkışı verilmelidir. Bölücü algoritmasının akışı aşağıdaki gibidir.



Şekil 4.28: İşaretsiz Bölücü Bileşeni Algoritma Akışı

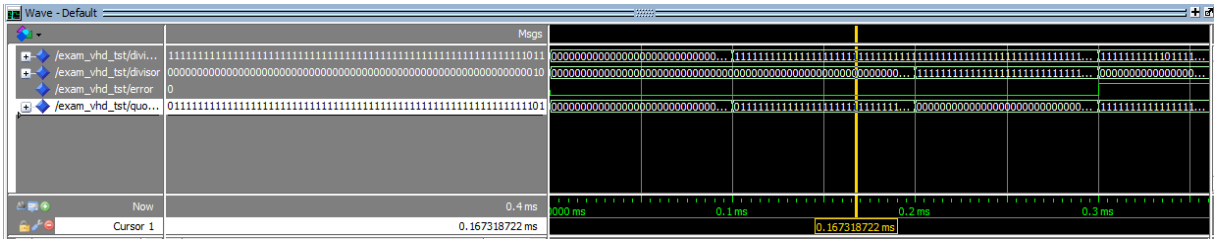
Booth's Algoritmasına benzer bir algoritma olan ve düzeltmeli(revised) bölme algoritması ile önce işaretsiz çarpıcı bileşeni oluşturulmuştur. Bileşen oluşturulduktan sonra derlenerek aşağıdaki bulgular elde edilmiştir.

Flow Summary	
Flow Status	Successful - Fri Nov 09 14:47:15 2018
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	exam
Top-level Entity Name	exam
Family	Cyclone II
Device	EP2C70F896C6
Timing Models	Final
Total logic elements	12,400 / 68,416 (18 %)
Total combinational functions	12,400 / 68,416 (18 %)
Dedicated logic registers	0 / 68,416 (0 %)
Total registers	0
Total pins	193 / 622 (31 %)
Total virtual pins	0
Total memory bits	0 / 1,152,000 (0 %)
Embedded Multiplier 9-bit elements	0 / 300 (0 %)
Total PLLs	0 / 4 (0 %)

Şekil 4.29: 64 Bit İşaretsiz Bölücü Bileşeni Derleme Raporu



Şekil 4.30: 64 Bit İşaretsiz Bölücü Bileşeni RTL Şeması



Şekil 4.31: 64 Bit İşaretsiz Bölücü Bileşeni Benzetim Sonucu

Daha sonra ise oluşturulan işaretsiz bölme bileşeni kullanılarak işaretli bölücü bileşeni tasarlanmıştır. İşaretli bölücü bileşeninin kod parçası aşağıdaki gibidir.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity DIVS_64 is
  generic (N:integer:=64);
  port (dividend  :in  std_logic_vector(N-1 downto 0) ;    --bolunen
        divisor   :in  std_logic_vector(N-1 downto 0) ;    --bolen
        clk       :in  std_logic;
        quotient  :out std_logic_vector(N-1 downto 0) ;    --bölüm
        error     :out std_logic);
end DIVS_64;
architecture behavioral of DIVS_64 is
  shared variable temp_dividend :std_logic_vector(N-1 downto 0);
  shared variable temp_divisor  :std_logic_vector(N-1 downto 0);
  signal temp_error             :std_logic;
  signal temp_quotient          :std_logic_vector(N-1 downto 0);
  signal div_in1                :std_logic_vector(N-1 downto 0);
  signal div_in2                :std_logic_vector(N-1 downto 0);

  component DIV_64 is
   generic (N:integer:=64);
   port (dividend  :in  std_logic_vector(N-1 downto 0) ;
         divisor   :in  std_logic_vector(N-1 downto 0) ;
         clk       :in  std_logic;
         quotient  :out std_logic_vector(N-1 downto 0) ;
         error     :out std_logic);
  end component;
begin
  process(dividend, divisor,clk)
  begin
   if clk='1' and clk'event then
    if dividend(N-1)='1' and divisor(N-1)='1' then
       temp_dividend:=(not dividend)+1;
       temp_divisor :=(not divisor) +1;
    elsif dividend(N-1)='0' and divisor(N-1)='1' then
       temp_dividend:=dividend;
       temp_divisor :=(not divisor) +1;
    elsif dividend(N-1)='1' and divisor(N-1)='0' then
       temp_dividend:=(not dividend)+1;
       temp_divisor :=divisor;
    else
       temp_dividend:=dividend;
       temp_divisor :=divisor;
    end if;
       div_in1<=temp_dividend;
       div_in2<=temp_divisor;
   end if;
  end process;

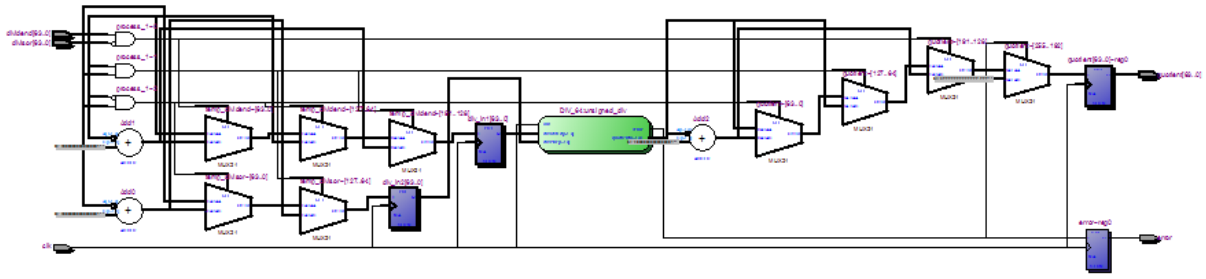
unsigned_div: DIV_64 port map (div_in1,div_in2,clk,temp_quotient,temp_error);

process (dividend,divisor,temp_quotient,temp_error,clk)
begin
if clk='1' and clk'event then
  if temp_error='1' then
    error<=temp_error;
    quotient<=(others=>'1');
  else
    error<=temp_error;
     if dividend(N-1)='1' and divisor(N-1)='0' then
     quotient <=(not temp_quotient)+1;
     elsif dividend(N-1)='1' and divisor(N-1)='1' then
```

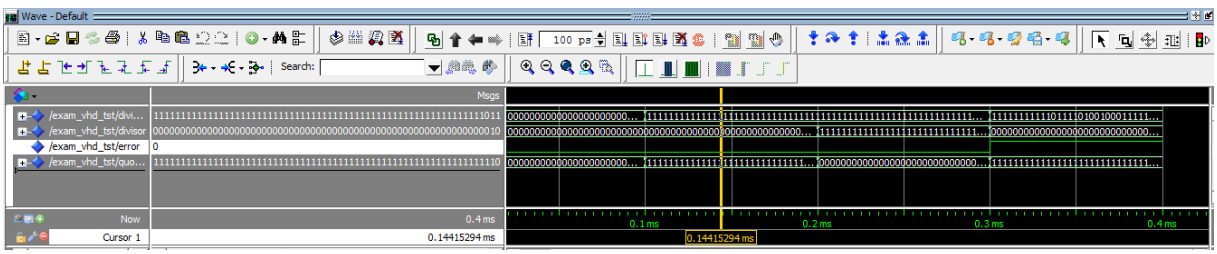
```
        quotient <=temp_quotient;
        elsif dividend(N-1)='0' and divisor(N-1)='1' then
        quotient <=(not temp_quotient)+1;
        else
        quotient <=temp_quotient;
        end if;
    end if;
end if;
end process;
end behavioral;
```

Flow Summary	
Flow Status	Successful - Fri Nov 09 14:16:41 2018
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	exam
Top-level Entity Name	exam
Family	Cyclone II
Device	EP2C70F896C6
Timing Models	Final
Total logic elements	12,866 / 68,416 (19 %)
Total combinational functions	12,866 / 68,416 (19 %)
Dedicated logic registers	0 / 68,416 (0 %)
Total registers	0
Total pins	193 / 622 (31 %)
Total virtual pins	0
Total memory bits	0 / 1,152,000 (0 %)
Embedded Multiplier 9-bit elements	0 / 300 (0 %)
Total PLLs	0 / 4 (0 %)

Şekil 4.32: 64 Bit İşaretli Bölücü Bileşeni Derleme Raporu



Şekil 4.33: 64 Bit İşaretli Bölücü Bileşeni RTL Şeması



Şekil 4.34: 64 Bit işaretli Bölücü Bileşeni Benzetim Sonucu

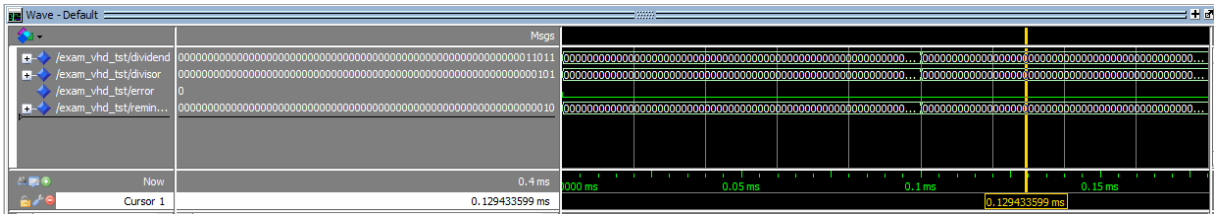
4.2.5. 64 Bit İşaretli Mod Algoritması

Oluşturulan bölme algoritmalarında sadece bölümün çıkış olarak gösterilmesine rağmen aslında yazılan kod bir ikili sayının başka bir ikiliye bölümünden kalanı da hesaplamaktadır. Bölme işleminde algoritma adımları tamamlandığında oluşan **A** sayısı giriş 1'in giriş 2'ye bölümünden kalanı ifade eder. Bu tanım ise mod alma tanımına karşılık gelir. Mod alma bileşeni, bölme işleminden kalan sayıyı(**A**) çıkış olarak ifade etmek için tasarıma eklenmiştir.

İşaretli sayılarda mod alma işlemi için önce işaretsiz sayılarda mod alan bileşen düzeltmeli(revised) bölme algoritması kullanılarak oluşturulmuş daha sonra ise bu bileşen kullanılarak işaretli sayılarda mod alan bileşen oluşturulmuştur. İşaretsiz mod alıcı bileşenin tasarımı derlendiğinde aşağıdaki bulgular elde edilmiştir.

Flow Summary	
Flow Status	Successful - Fri Nov 09 15:54:00 2018
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	exam
Top-level Entity Name	exam
Family	Cyclone II
Device	EP2C70F896C6
Timing Models	Final
Total logic elements	12,438 / 68,416 (18 %)
Total combinational functions	12,438 / 68,416 (18 %)
Dedicated logic registers	0 / 68,416 (0 %)
Total registers	0
Total pins	193 / 622 (31 %)
Total virtual pins	0
Total memory bits	0 / 1,152,000 (0 %)
Embedded Multiplier 9-bit elements	0 / 300 (0 %)
Total PLLs	0 / 4 (0 %)

Şekil 4.35: 64 Bit İşaretsiz Mod Alıcı Bileşen Derleme Raporu



Şekil 4.36: 64 Bit İşaretsiz Mod Alıcı Bileşen Benzetim Sonucu

Bu bileşen kullanımı ile oluşturulan 64 bit işaretli mod alma bileşeni kaynak kodu aşağıdaki gibidir.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity MODS_64 is
  generic (N:integer:=64);
  port (in1     :in  std_logic_vector(N-1 downto 0) ;    --bolunen
        in2     :in  std_logic_vector(N-1 downto 0) ;    --bolen
        clk     :in  std_logic;

        error   :out std_logic;
        out1    :out std_logic_vector(N-1 downto 0));    --kalan
end MODS_64;
architecture behavioral of MODS_64 is
  shared variable temp_in1 :std_logic_vector(N-1 downto 0);
  shared variable temp_in2 :std_logic_vector(N-1 downto 0);
  signal          mod_in1  :std_logic_vector(N-1 downto 0);
  signal          mod_in2  :std_logic_vector(N-1 downto 0);
  signal          temp_out1:std_logic_vector(N-1 downto 0)  ;
  signal          temp_error:std_logic;

  component mod_64 is
  generic (N:integer:=64);
  port (dividend  :in  std_logic_vector(N-1 downto 0) ;    --bolunen
        divisor   :in  std_logic_vector(N-1 downto 0) ;    --bolen
        clk       :in  std_logic;
        reminder  :out std_logic_vector(N-1 downto 0) ;    --bölümden kalan
        error     :out std_logic);
  end component;
begin
  process(in1, in2,clk)
  begin
  if clk='1' and clk'event then
  if in1(N-1)='1' and in2(N-1)='1' then
     temp_in1:=(not in1)+1;
     temp_in2:=(not in2) +1;
  elsif in1(N-1)='0' and in2(N-1)='1' then
     temp_in1:=in1;
     temp_in2:=(not in2) +1;
  elsif in1(N-1)='1' and in2(N-1)='0' then
     temp_in1:=(not in1)+1;
     temp_in2:=in2;
  else
     temp_in1:=in1;
     temp_in2:=in2;
  end if;
     mod_in1<=temp_in1;
     mod_in2<=temp_in2;
  end if;
  end process;

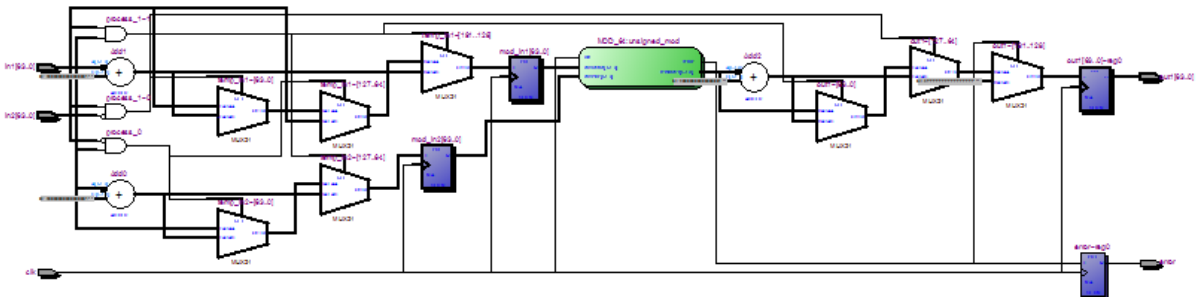
   unsigned_mod: MOD_64 port map (mod_in1, mod_in2,clk, temp_out1, temp_error);

  process (in1,in2,temp_out1,temp_error,clk)
  begin
  if clk='1' and clk'event then
  if temp_error='1' then
     error<=temp_error;
     out1<=(others=>'1');
```

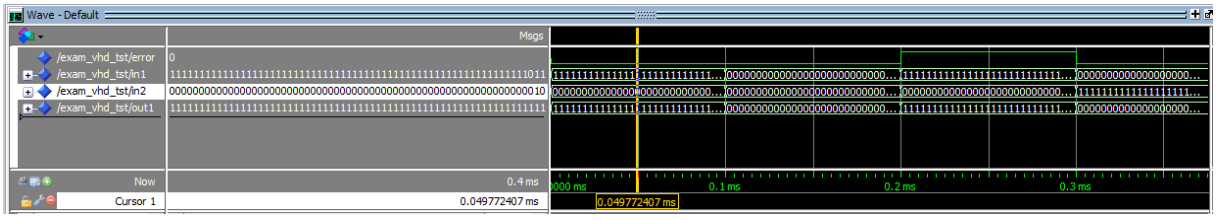
```
  else
     error<=temp_error;
     if in1(N-1)='1' and in2(N-1)='0' then
        out1<=(not temp_out1)+1;
     elsif in1(N-1)='1' and in2(N-1)='1' then
        out1<=(not temp_out1)+1;
     elsif in1(N-1)='0' and in2(N-1)='1' then
        out1<=temp_out1;
     else
        out1<=temp_out1;
     end if;
  end if;
 end if;
end process;
end behavioral;
```

Flow Summary	
Flow Status	Successful - Fri Nov 09 16:36:17 2018
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	exam
Top-level Entity Name	exam
Family	Cyclone II
Device	EP2C70F896C6
Timing Models	Final
Total logic elements	12,935 / 68,416 (19 %)
Total combinational functions	12,935 / 68,416 (19 %)
Dedicated logic registers	0 / 68,416 (0 %)
Total registers	0
Total pins	193 / 622 (31 %)
Total virtual pins	0

Şekil 4.37: 64 Bit İşaretli Mod Alma Bileşeni Derleme Raporu



Şekil 4.38: 64 Bit İşaretli Mod Alma Bileşeni RTL Şeması



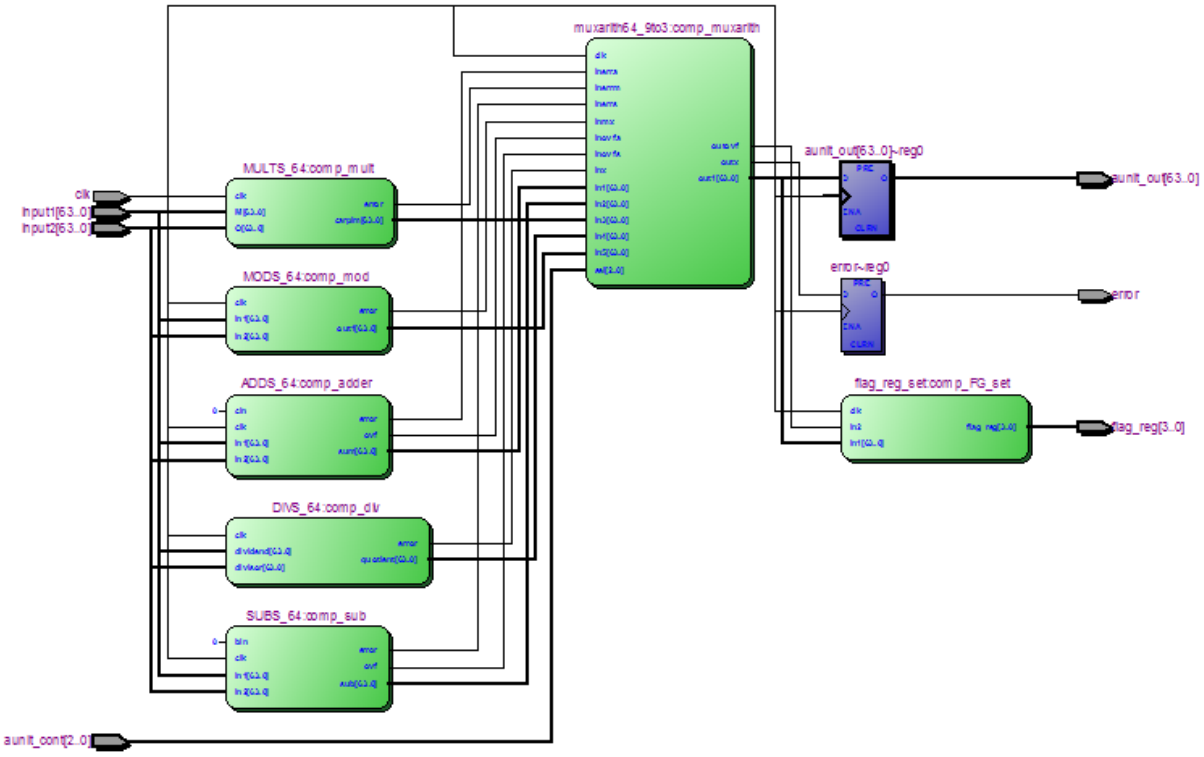
Şekil 4.39: 64 Bit İşaretli Mod Alma Bileşeni Benzetim Sonucu

4.2.6. 64 Bit Aritmetik Birim Bulguları

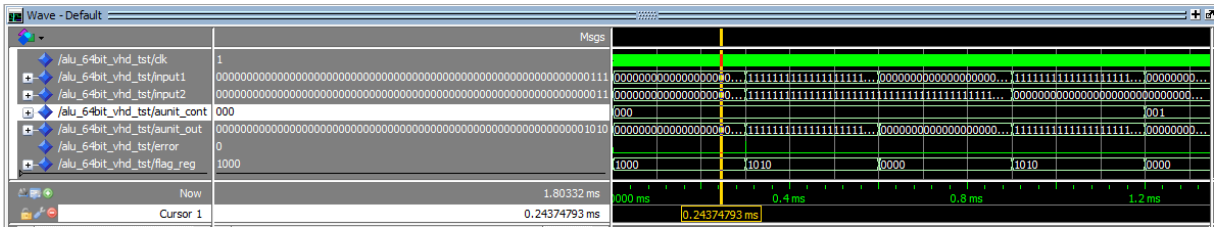
Aritmetik birim için yazılan tüm bileşenler bir üst seviye tasarım olan *alu64_artihmetic_unit* altında toplanmıştır.

Flow Summary	
Flow Status	Successful - Mon Nov 19 15:52:43 2018
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	alu_64bit
Top-level Entity Name	alu_64bit
Family	Cyclone II
Device	EP2C70F896C6
Timing Models	Final
Total logic elements	19,319 / 68,416 (28 %)
Total combinational functions	18,910 / 68,416 (28 %)
Dedicated logic registers	2,102 / 68,416 (3 %)
Total registers	2102
Total pins	201 / 622 (32 %)
Total virtual pins	0
Total memory bits	393 / 1,152,000 (< 1 %)
Embedded Multiplier 9-bit elements	0 / 300 (0 %)
Total PLLs	0 / 4 (0 %)

Şekil 4.40: 64 Bit Aritmetik Birim Derleme Raporu



Şekil 4.41: 64 Bit Aritmetik Birim RTL Şeması



Şekil 4.42: 64 Bit Aritmetik Birim Benzetim Sonucu

4.3. 64 BİT KAYDIRMA-DÖNDÜRME BİRİMİ TASARIMI

Tasarımda aritmetik ve mantık biriminin dışında, SR (Shift-Rotate) birimi olarak adlandırdığım içerisinde aritmetik -lojik kaydırma ve döndürme işlemlerinin gerçekleştirildiği üçüncü bir birim tasarlanmıştır. Bu birim içerisinde gerçekleştirilen işlemler aşağıdaki gibidir.

- Aritmetik Sağa Kaydırma
- Aritmetik Sola Kaydırma
- Lojik Sağa Kaydırma
- Lojik Sola Kaydırma
- Sağa Döndürme
- Sola Döndürme

İşlemler 64 bitlik girişler üzerinden işlenmiştir Kaydırma ve döndürme işlemleri için kaç adet kaydırma veya döndürme yapılacağı ikili sayılar şeklinde girilmiştir. Yani ikinci giriş integer türünde değil std_logic_vector türündedir. Bu giriş program içerisinde integer türüne dönüştürülerek kullanılmıştır. Bu dönüşüm esnasında türler arası dönüştürme kuralları dikkate alınmalıdır ve numeric.std ve std_logic_1164 kütüphanelerinin kullanımı unutulmamalıdır. Bu türler arası dönüştürme kuralları gereğince her bir veri türü istenilen bir türe doğrudan dönüştürülemeyebilir. Örneğin std_logic_vector türü, integer türüne doğrudan dönüştürülemez, std_logic_vector türünün önce signed ya da unsigned veri türüne dönüştürülmesi gerekir.

4.3.1. Aritmetik Sağa Kaydırma

Aritmetik kaydırma işlemlerinde, sağa ya da sola kaydırmaya göre MSB ve LSB bitlerinin lojik değerinin ne olduğu önemlidir.

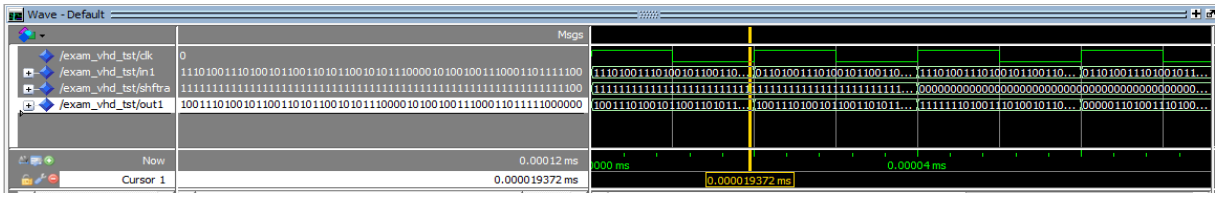
Aritmetik Sağa Kaydırma işleminde MSB bitinin değerine bakılır. Eğer MSB bitinin değeri lojik ‘0’ ise kaydırma işleminde bitler sağa doğru kayarken her bir bitlik kaymada LSB biti düşer ve LSB biti yerine bu bitten hemen önceki sayı geçer. Eksilen bit yerine de MSB bitine ‘0’ değeri eklenir. Bu durum MSB bitinin değeri lojik ‘1’ iken de aynen tekrar eder fakat MSB bitine eklenen değer bu kez ‘1’ dir.

Bu tasarımda kaç adet kaydırma yapılacağı giriş 2 değeri ile std_logic_vector türünde bildirilmiştir. Bu tür program içerisinde integer değerine çevrilerek işlem gerçekleştirilmiştir. Bu bileşende dikkat edilmesi gereken önemli bir nokta 64 bitlik bir girişin 64 ve üzeri bir sayıda sağa kaydırılması sonucunda MSB bitinin değerine göre ya 64 bitin her biri ‘1’ yada ‘0’ değerini alır. Dolayısı ile ikinci girişin integer değeri 64 eşit ya da 64’ten büyük ise MSB biti değerine görü sonucun her biti ‘1’ ya da ‘0’ olmalıdır. İkinci girişin integer türünün değerinin 64 ve üzeri olması bu girişin en az 6. Bitinin(2^6=64) ‘1’ değerini alması ile gerçekleşir.

Aritmetik kaydırma işlemi *numeric_std* kütüphanesinde tanımlı “sra” komutu ile gerçekleştirilmiştir. Yapılan tasarım derlendikten sonra;

Flow Summary	
Flow Status	Successful - Tue Nov 13 23:40:36 2018
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	exam
Top-level Entity Name	exam
Family	Cyclone II
Device	EP2C70F896C6
Timing Models	Final
Total logic elements	870 / 68,416 (1 %)
Total combinational functions	869 / 68,416 (1 %)
Dedicated logic registers	64 / 68,416 (< 1 %)
Total registers	64
Total pins	193 / 622 (31 %)
Total virtual pins	0
Total memory bits	0 / 1,152,000 (0 %)
Embedded Multiplier 9-bit elements	0 / 300 (0 %)
Total PLLs	0 / 4 (0 %)

Şekil 4.43: Aritmetik Kaydırma Bileşeni Derleme Raporu

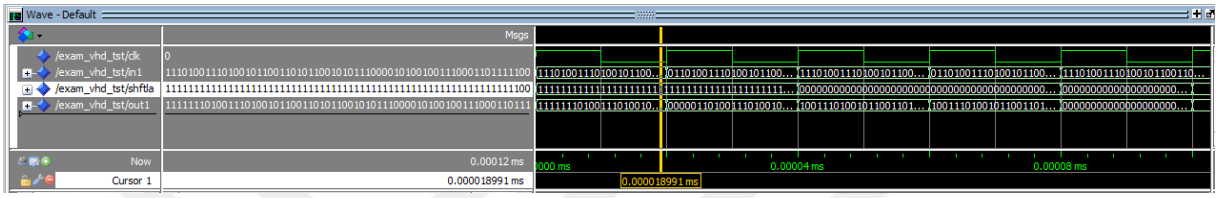


Şekil 4.44: Aritmetik Sağa Kaydırma Bileşeni Benzetim sonucu

4.3.2. Aritmetik Sola Kaydırma

Aritmetik sola kaydırma tasarımı aritmetik sağa kaydırma tasarımı ile çalışma yapısı bakımından benzerdir. Fakat bu kez kayma sola doğru gerçekleşir ve LSB bitinin değerine göre sayının sonuna '1' ya da '0' eklenir. Ayrıca her kayma işleminde bu kez MSB biti düşer.

Tasarım derleme raporları birbirinin aynıdır. Sadece benzetim sonuçları farklıdır. Derlenen tasarımın benzetim sonuçları aşağıdaki gibidir.



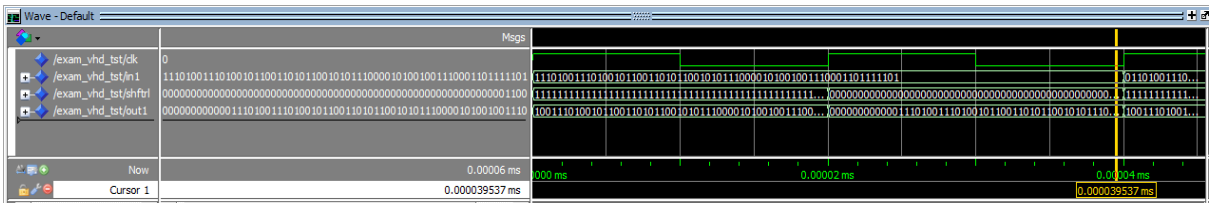
Şekil 4.45: Aritmetik Sola Kaydırma Bileşeni Benzetim sonucu

4.3.3. Lojik Sağa Kaydırma

Lojik kaydırma işlemlerin aritmetikkaydırma işlemlerinden tek farkı LSB ya da MSB bitine bakılmaksızın çıkışın LSB ya da MSB bitinden sonra eklenen değerin '0' olmasıdır.

Flow Summary	
Flow Status	Successful - Wed Nov 14 00:42:45 2018
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	exam
Top-level Entity Name	exam
Family	Cyclone II
Device	EP2C70F896C6
Timing Models	Final
Total logic elements	809 / 68,416 (1 %)
Total combinational functions	809 / 68,416 (1 %)
Dedicated logic registers	64 / 68,416 (< 1 %)
Total registers	64
Total pins	193 / 622 (31 %)
Total virtual pins	0
Total memory bits	0 / 1,152,000 (0 %)
Embedded Multiplier 9-bit elements	0 / 300 (0 %)
Total PLLs	0 / 4 (0 %)

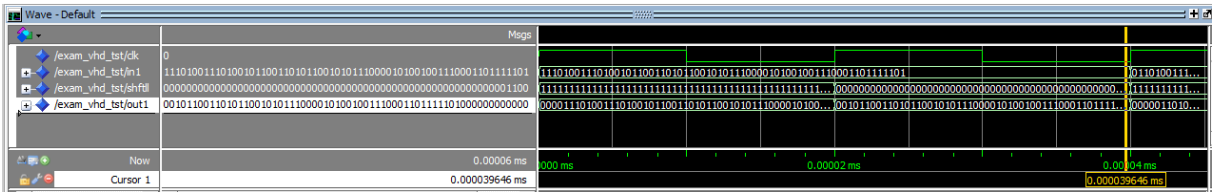
Şekil 4.46: Lojik Sağa Kaydırma Bileşeni Derleme Raporu



Şekil 4.47: Lojik Sağa Kaydırma Bileşeni Benzetim Sonucu

4.3.4. Lojik Sola Kaydırma

Lojik Sola kaydırmanın sağa kaydırmadan tek farkı kaymanın sola doğru olması ve '0' değerinin LSB bitinden hemen sonra yapılmasıdır. Lojik kaydırmaların derleme raporları birbirinin aynısıdır.



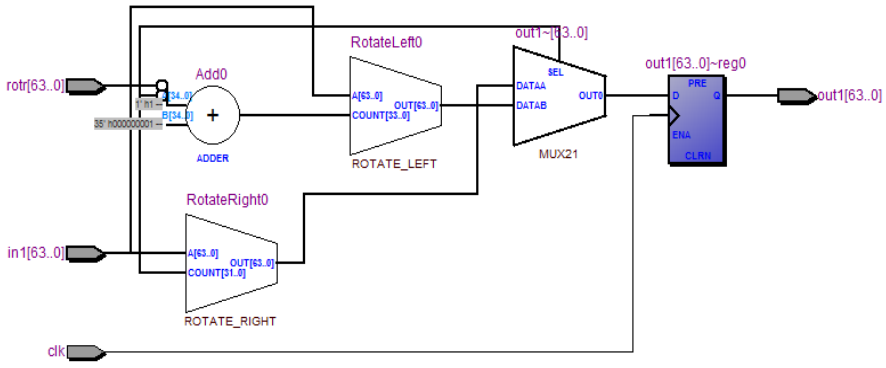
Şekil 4.48: Lojik Sola Kaydırma Bileşeni Benzetim Sonucu

4.3.5. Sağa Döndürme

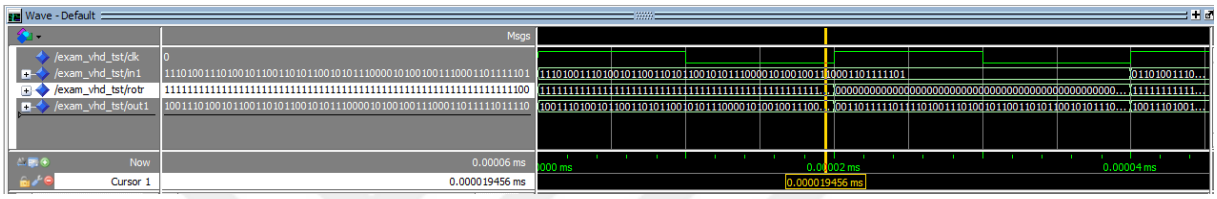
Sağa ve sola döndürme işlemleri kaydırma işlemlerinden farklı olarak, düşen bitler döndürme işlemine göre MSB ya da LSB bitinin yanına eklenir. Tasarımda, döndürme işleminin kaç kez gerçekleştirileceği giriş 2 ile bildirilmiştir. Giriş 2 tasarım içerisinde integer türüne dönüştürülerek kullanılmıştır. Burada dikkat edilmesi gereken kısım her 64 döndürme işleminde sayı ilk değerine ulaşmaktadır. Dolayısı ile giriş 2'nin integer değeri 64'ün katı olduğunda giriş 1 çıkışa aynen ulaşır. Giriş 2'nin integer değerinin 64'ün katı olmadığı durumlarda ise bu değerin 64'e bölümünden kalanına bakılır ve bu kalan kadar döndürme işlemi gerçekleştirilir. Giriş 2'nin integer değeri negatif sayı ise bileşen sola döndürme işlemi yapar.

Flow Summary	
Flow Status	Successful - Mon Nov 12 21:20:17 2018
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	exam
Top-level Entity Name	exam
Family	Cyclone II
Device	EP2C70F896C6
Timing Models	Final
Total logic elements	925 / 68,416 (1 %)
Total combinational functions	920 / 68,416 (1 %)
Dedicated logic registers	83 / 68,416 (< 1 %)
Total registers	83
Total pins	194 / 622 (31 %)
Total virtual pins	0
Total memory bits	0 / 1,152,000 (0 %)
Embedded Multiplier 9-bit elements	0 / 300 (0 %)
Total PLLs	0 / 4 (0 %)

Şekil 4.49: Sağa Döndürme Bileşeni Derleme Raporu



Şekil 4.50: Sağa Döndürme Bileşeni RTL şeması

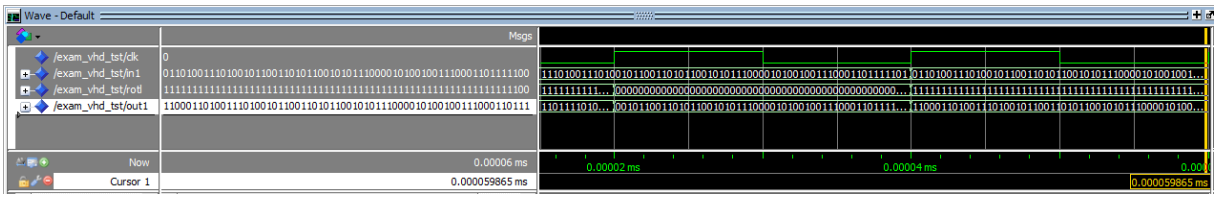


Şekil 4.51: Sağa Döndürme Bileşeni Benzetim Sonucu

4.3.6. Sola Döndürme

Sola döndürme bileşenin sağa döndürme bileşeninden tek farkı dönme işleminin sola doğru olmasıdır. Dolayısı ile MSB bitinden düşen değer LSB bitinin yanına tekrar yazılır. Giriş 2'nin integer değeri negatif ise bileşen sağa döndürme işlemi yapar.

Bileşenin derleme raporu ve RTL şeması aynıdır.



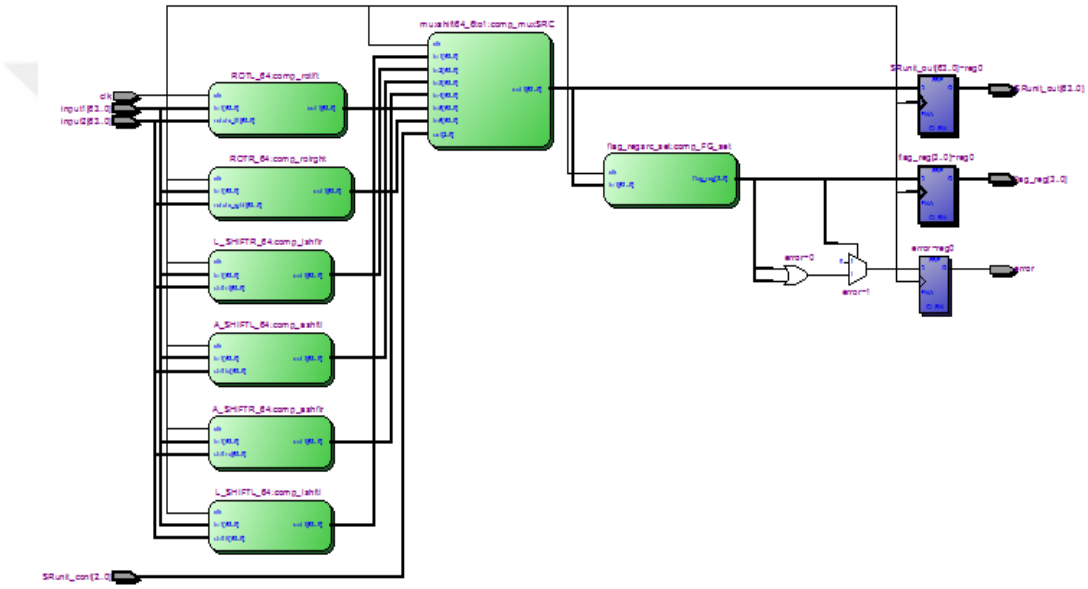
Şekil 4.52: Sola Döndürme Bileşeni Benzetim Sonucu

4.3.7. 64 Bit Kaydırma-Döndürme Birimi Bulguları

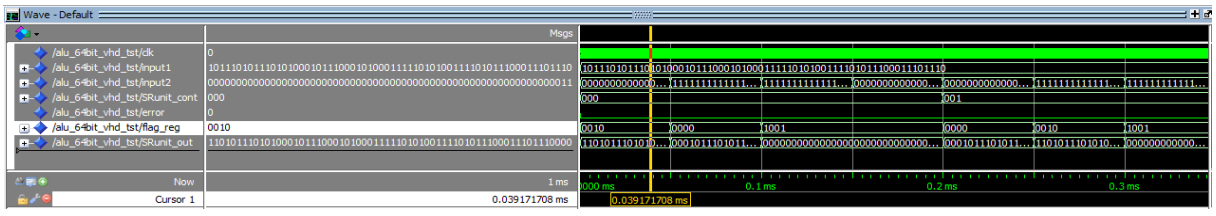
Kaydırma-Döndürme birimi için yazılan tüm bileşenler *alu64_SR_unit* altında toplanmıştır.

Flow Summary	
Flow Status	Successful - Mon Nov 19 20:04:35 2018
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	alu_64bit
Top-level Entity Name	alu_64bit
Family	Cyclone II
Device	EP2C70F896C6
Timing Models	Final
Total logic elements	2,833 / 68,416 (4 %)
Total combinational functions	2,778 / 68,416 (4 %)
Dedicated logic registers	522 / 68,416 (< 1 %)
Total registers	522
Total pins	201 / 622 (32 %)
Total virtual pins	0
Total memory bits	0 / 1,152,000 (0 %)
Embedded Multiplier 9-bit elements	0 / 300 (0 %)
Total PLLs	0 / 4 (0 %)

Şekil 4.53: 64 Bit Kaydırma Döndürme Birimi Derleme Raporu



Şekil 4.54: 64 Bir Kaydırma ve Döndürme Birimi RTL Şeması



Şekil 4.55: 64 Bit Kaydırma ve Döndürme Birimi Benzetim Sonucu

4.4. 64 BİT ARİTMETİK MANTIK BİRİMİ

Sırası ile oluşturulan mantık birimi, aritmetik birimi ve kaydırma-döndürme birimi en üst seviye tasarım içerisinde çağrılmak üzere bileşen olarak kaydedilmiştir. Her bileşenin 3 çıkışı bulunmaktadır. Bunlar ALU işlem çıkışı, taşma, işaret, parity ve sıfır bayraklarının durumunu

gösteren flag register çıkışı ve işlem çıkışında herhangi bir işlem çıkışında hata olup olmadığını gösteren hata çıkışıdır.

Flag register çıkışını kontrol eden bayraklar için fonksiyon yapıları tanımlanmıştır. Bu fonksiyonlar ALU işlem çıkışının kullanarak flag register'da gösterilecek çıkışı üretir. Hata çıkışında ise toplama ve çıkarma işleminde taşma oluştuğunda sonucun yanlış olması durumunu, bölme işleminde oluşan "sıfır/sıfır" belirsizliği ve "sayı/sıfır" tanımsız durumunu ve çarpma işleminde çarpım sonucunun 64 biti aştığı durumunu ifade etmek için kullanılmıştır. Aşağıda tasarlanan 64 Bit Aritmetik Mantık biriminde kullanılan işlemleri ve kontrol biriminden geldiği kabul edilen seçim operatörü karşılığı verilmiştir.

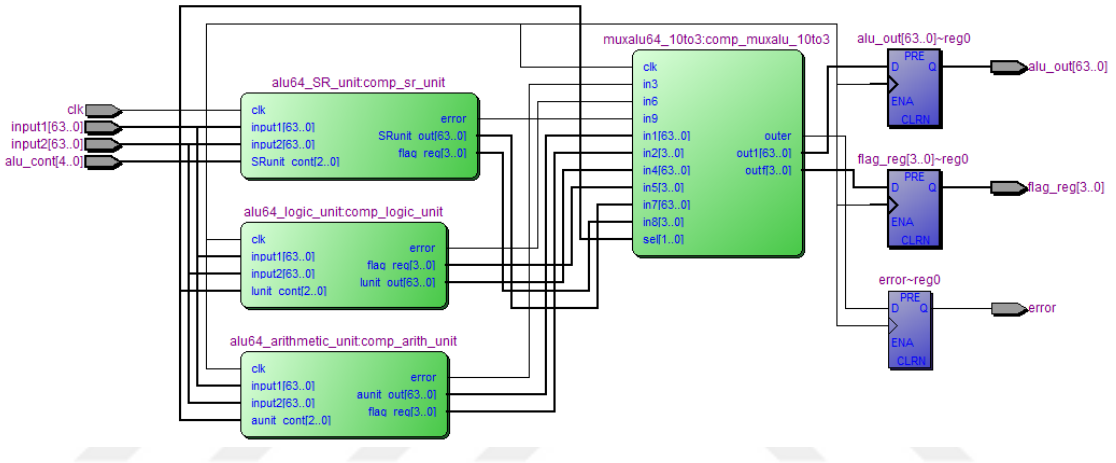
Tablo 4.1: 64 Bit ALU Talimat ve İşlem Tablosu

Alu_cont(4)	Alu_cont(3)	Alu_cont(2)	Alu_cont(1)	Alu_cont(0)	İşlem Tanımı
0	0	0	0	0	İşaretli Toplama
0	0	0	0	1	İşaretli Çıkarma
0	0	0	1	0	İşaretli Çarpma
0	0	0	1	1	İşaretli Bölme
0	0	1	0	0	İşaretli Mod Alma
0	1	0	0	0	AND
0	1	0	0	1	OR
0	1	0	1	0	XOR
0	1	0	1	1	NOT in1
0	1	1	0	0	NOT in2
0	1	1	0	1	NAND
0	1	1	1	0	NOR
0	1	1	1	1	XNOR
1	0	0	0	0	Lojik sola kaydır
1	0	0	0	1	Lojik sağa kaydır
1	0	0	1	0	Aritmetik sola kaydır
1	0	0	1	1	Aritmetik sağa kaydır
1	0	1	0	0	Sola döndür
1	0	1	0	1	Sağa döndür

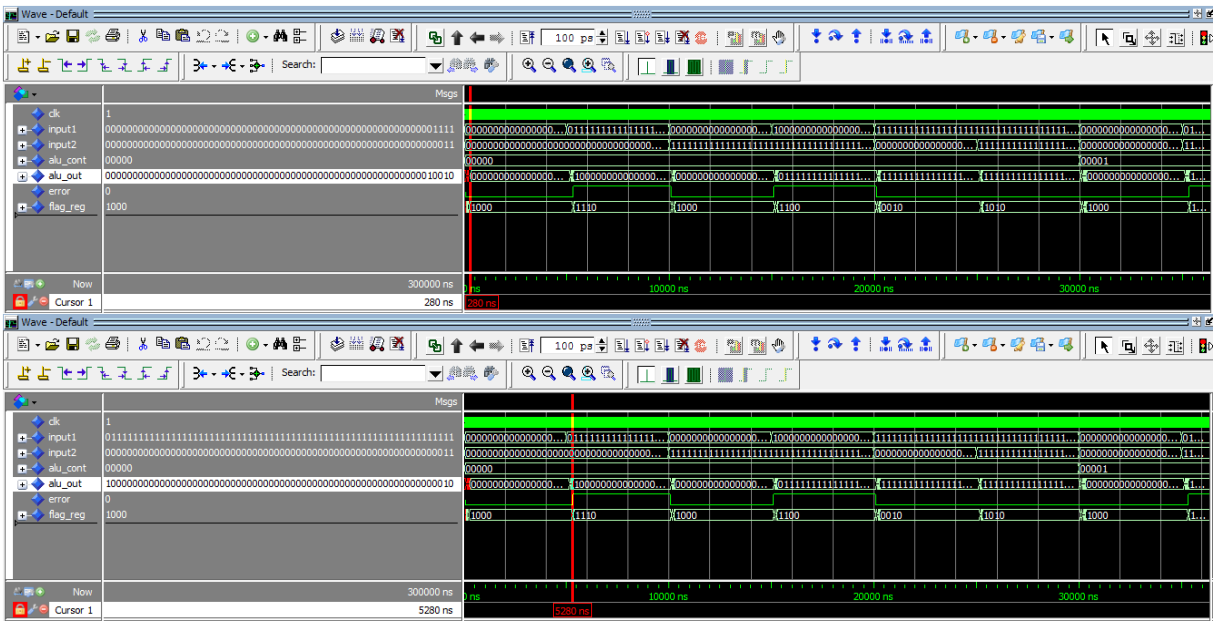
Yazılan kaynak kodun derlenmesinin ardından;

Flow Summary	
Flow Status	Successful - Mon Nov 19 22:36:26 2018
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	alu_64bit
Top-level Entity Name	alu_64bit
Family	Cyclone II
Device	EP2C70F896C6
Timing Models	Final
Total logic elements	22,776 / 68,416 (33 %)
Total combinational functions	22,129 / 68,416 (32 %)
Dedicated logic registers	2,891 / 68,416 (4 %)
Total registers	2891
Total pins	203 / 622 (33 %)
Total virtual pins	0
Total memory bits	399 / 1,152,000 (< 1 %)
Embedded Multiplier 9-bit elements	0 / 300 (0 %)
Total PLLs	0 / 4 (0 %)

Şekil 4.56: 64 Bit Aritmetik Mantık Birimi Derleme Raporu



Şekil 4.57: 64 Bit Aritmetik Mantık Birimi RTL Şeması



Şekil 4.58: 64 Bit Aritmetik Mantık Birimi Benzetim Sonuçları

5. TARTIŞMA VE SONUÇ

Bu çalışma ile VHDL donanım tanımla dili ve FPGA teknolojisi kullanılarak işaretli sayıları işleyen 64 bit Aritmetik Mantık Biriminin gerçekleştirilmesi hedeflenmiştir. Tasarım yöntemi olarak "buttom-up" yöntemi seçilmiş ve birim içerisindeki tüm alt birimler bileşen olarak tanımlanmıştır. Bu tasarım yöntemi ile sistem için uygun olmayan parçalar kolayca tespit edilmiş ve değişiklik yapılabilmiştir. Yöntemin dezavantajı ise tasarımın tüm bileşenleri arasındaki ilişkiler tanımlanmadan tüm tasarım benzetiminin yapılamaması olmuştur. Bu sebeple ilerleyen safhalarda oluşabilecek hataların öngörülmesi zorlaşmış ve tasarımın gerçekleştirilme süresi beklenilenin üzerine çıkmıştır.(Mühendislik maliyetinin artışı).

Modelleme yöntemi olarak ise davranışsal ve yapısal modelleme teknikleri kullanılmıştır. Davranışsal modellemede kullanılan *process* yapıları ile paralel işlemler gerçekleştirilmiş ve yapısal modelleme ile bileşenler arası bağlantılar yapılmış ve tasarım akışı sağlanmıştır.

Tasarlanan ALU birimi işlem bileşenleri önce işaretsiz sayılar için gerçekleştirilmiş daha sonra oluşturulan bileşenler işaretli sayıların kullanımı için çeşitli process yapıları ile modellenmiştir. Tasarımda kullanılan girişlerin veri türleri std_logic ve std_logic_vector veri türleridir. Veri türü olarak "signed" veri türü kullanılmadan sistem işaretli sayılar için işlem yapabilmektedir. Bu seçimin dezavantajı, işaretli sayılara geçiş için oluşturulan adaptasyon process'lerinin kullanılan lojik element sayılarını artırması dolayısı ile olası gecikmeleri tetiklemesi, avantajı ise tasarım benzetimi ve testlerinin kolay ve sorunsuz gerçekleşmesidir. Ayrıca işaretli sayılardaki aritmetik işlemlerde taşma kontrolü yapıları gecikmelere neden olmuştur.

ALU tasarımı yapısal olarak 3 ana birim olarak tasarlanmıştır. Her bir birim altında birim ile ilişkili bileşenler toplanmış ve multiplexer yapıları ile en üst seviyedeki tasarımdan gelen ALU seçim operatörü aracılığı ile birimler arası bağlantılar sağlanıştır. Tasarlanan birimler, 7 alt mantık bileşeninden oluşan mantık birimi, 5 alt aritmetik bileşenden oluşan aritmetik birimi ve 6 alt kaydırma ve döndürme bileşeninden oluşan kaydırma-döndürme(SR) birimidir. Sisteme veri girişi veri giriş portları üzerinden yapılır ve bu veri sinyalleri üst seviye tasarımı ve alt birimleri besler. Data girişlerinin dışında sisteme 50 MHz çalışma frekansındaki saat girişi eklenmesi ile sistemin senkronizasyonu sağlanmıştır. Üst seviye tasarımda tanımlanan

saat girişi en alt seviye bileşene kadar sistemi besler. Son giriş olarak CPU kontrol biriminden geldiği varsayılacak olan aritmetik mantık biriminin operasyon seçimini yapan ALU kontrol giriş portudur. Bu giriş üst seviye tasarımda ve alt üç birimde tanımlanmış olan çoğullayıcı yapılarını tetikler. ALU kontrol girişinin değerine göre çoğullayıcı çıkışı belirlenir. Bu çoğullayıcı çıkışları ise sistemin çıkış portlarındaki sonuçları oluşturur. Sistem üç çıkıştan oluşur. Bunlar; ALU işlem çıkışı, ALU hata çıkışı ve bayrak durumlarını gösteren register flag çıkışıdır. ALU hata çıkışı, taşmadan oluşan işlem sonucundaki hataları, çarpma işleminde 32 bit ile ifade edilebilecek maksimum değerin üzerine çıkan iki sayının çarpımı ile oluşan hataları ve bölme işlemindeki "0/0" belirsizliği ve "sayı/0" tanımsız durumlarında oluşan hataları ifade etmek için oluşturulmuştur.

Sistemin data girişlerinden alınan veri 64 bit uzunluğunda, ALU kontrol girişi 5 bit uzunluğunda, saat girişi ise 1 bit uzunluğundadır(benzetim esnasında 50 MHz çalışma frekanslı saat kullanılmıştır). Çıkış portlarında ise işlem çıkışı 64 bit, hata çıkışı 1 bit ve register flag çıkışı 4 bittir. Bu portlardan ALU kontrol girişinin ilk iki biti en üst seviye tasarımda kullanılır ve bu bitler 3 alt birimin çıkışını kontrol eder. Bu bitler sırası ile aritmetik birim için "00", lojik birim için "01" ve kaydırma/döndürme birimi için "10" olarak tanımlanmıştır. Son üç bit ise alt birimler içerisindeki işlem çıkışlarını kontrol eder. Bu üç bit alt birim içerisinde tanımlanmış işlem sayısı ile orantılıdır.

Mantık Birimi işlemleri, AND, OR, XOR, NOT, NAND, NOR ve XNOR işlemleridir. İşlemlerin davranışsal modellenmesi doğruluk tabloları çıkışlarının karnaugh haritaları ile sadeleştirilmiş fonksiyonlar halinde ifade edilmesi ile gerçekleştirilmiştir. Her bir bileşen tasarımı VHDL dilinde tanımlı mantık operatörleri ile ve process yapıları içerisinde tanımlanan sıralı ifadeler ile tasarlanmıştır. İki yöntemde de kullanılan lojik element sayısı aynı olmasına karşılık, process içerisinde sıralı ifadeler kullanılarak yapılan tasarımda sonuç oluşumu çok küçük gecikmeler ile alınmıştır. Girişlerin boyutu arttıkça döngülerin tamamlanma süresi arttığında çıkışın oluşması da gecikmiştir. Tüm mantık birimi bileşenleri tasarlandıktan sonra ana yapıdan gelen seçim operatörü mantık birimine 8:1'lik bir multiplexer yapısı dahil edilerek mantık birimi, ana yapıda çağırılmak üzere, bileşene dönüştürülmüştür.

Aritmetik Birimi işlemleri toplama, çıkarma, çarpma, bölme ve mod alma işlemleridir. Her bir işlem ayrı bir bileşen(component) olarak tasarlanmıştır. Her bir bileşen önce işaretsiz

sayılarda çalışacak şekilde tasarlanmış daha sonra çeşitli process yapıları ile işaretli sayılarda çalışacak şekilde tekrar düzenlenmiştir. İşaretli sayıların ifadesi 2'nin tümleyeni yöntemi ile tanımlanmıştır. Tüm aritmetik birimi bileşenleri tasarlandıktan sonra ana yapı ile bağlantıyı sağlamak amacıyla 5:1'lik bir multiplexer yapısı aritmetik birime dahil edilmiş ve bileşen olarak kaydedilmiştir. Aritmetik birim ile birlikte çıkışın durumunu kontrol eden, Zero Flag, Sign Flag, Parity Flag ve Overflow flag bayrakları için fonksiyonlar oluşturulmuştur.

Aritmetik birim işlemleri klasik algoritmalar kullanılarak tasarlandığında lojik element kullanımının ve gecikmelerin arttığı gözlemlenmiştir. Bu iki performans kriterinin iyileştirilmesi için bazı işlemlerde özel algoritmalar kullanılmıştır. 64 bit toplama bileşeni koşullu toplayıcı algoritması kullanımı ile çarpma işlemi Booth's algoritması ile ve bölme işlemi de Revised Division algoritması kullanılarak tasarlanmıştır. Koşullu toplayıcı algoritması kullanımı ile klasik elde dalgalı toplama(RCA) algoritmasında elde iletimi ile meydana gelen gecikmeler giderilmiştir, fakat alan kullanımı artmıştır. Booth's ve Revised Division algoritmalarının kullanımı ile çarpma ve bölme operasyonlarında kullanılan lojik element sayısı azalmıştır.

Kaydırma- Döndürme Birimi işlemleri, aritmetik sağa kaydırma, aritmetik sola kaydırma, lojik sağa kaydırma, lojik sola kaydırma, sağa döndürme ve sola döndürme işlemleridir. Her bir bileşen tasarımı VHDL dilinde tanımlı kaydırma- döndürme operatörleri ile ve process yapıları içerisinde tanımlanan sıralı ifadeler ile tasarlanmıştır. Sıralı ifadeleri kullanımı ile tasarlanan yapıda VHDL dilinde tanımlı operatörlerin kullanıldığı tasarıma kıyasla gecikmelerin arttığı gözlemlenmiştir.

Tasarımın performansını artırmak ve gecikmeleri azaltmak için aritmetik işlem biriminin tasarımına dikkat edilmesi gerekir. Bu kısım gerçekleştirdiği işlemler sebebi ile gecikmelerin yaşandığı ve en büyük alanın kullanıldığı kısımdır. Tüm işlemlerin temelinde toplayıcı ve çıkarıcı yapıları yatar dolayısı ile yüksek performansı ve düşük gecikmeli bir toplayıcı ya da çıkarıcı yapısı tüm sistemin performansını olumlu yönde etkileyecektir.

FPGA yapılarının paralel işlem yapabilme kabiliyetleri kullanılarak yüksek performanslı sistemlerin tasarımı yapılabilir. Ayrıca tekrar tekrar programlanabilir olma özellikleri ile güvenilir ve hatasız tasarımlar oluşturulabilmektedir.

KAYNAKLAR

[1]. Stallings, W., 2016, *Computer Organızatıon And Archıtecture Desıgnıng For Performance,* 10th edition.

[2]. Kuntman, H. H., Toker, A. ve Özcan, S., 2000, *Sayısal Elektronik Devreleri*, Birsen Yayınevi, İstanbul, 957-511-255-3

[3]. Çömlekçi, M., Tüzel, S., *PC Donanımı Herkes İçin*, Alfa BasımYayım Dağıtım, Nisan, 2005.

[4]. Sarıtaş, E. ve Karataş, S., 2015, *Her Yönüyle FPGA ve VHDL*, Palme Yayıncılık, Ankara,978-605-355-129-4

[5]. Henkoğlu Türkay, *Modern Donanım Mimarisi*, Pusula Yayıncılık, Şubat, 2005.

[6]. Özbey, R., Sertbaş, A., *Klasik Çarpma Algoritmalarının Donanımsal Simülasyonları Ve Performans Değerlendirimi*, Inter. Conf. on Electrical and Electronics Engineering (ELECO 2004), BURSA, TÜRKİYE, 8-12 Aralık 2004, ss.303-308

[7]. Pala, Z., *Bilgisayar Donanımı* , Türkmen Kitapevi , 2005.

[8]. Tamburi, P., 2010, *Farklı Hızlı Toplayıcı Mimarilerinin FPGA Kullanarak Gerçeklenmesi,* Yüksek Lisans Tezi

[9]. Chetia, R. Sarma, K. C. D., Baruah, G., 2013, *Behavioral Design and Synthesis of 64 BIT ALU using Xilinx ISE,* IOSR Journal of Electronics and Communication Engineering (IOSR-JECE) e-ISSN: 2278-2834,p- ISSN: 2278-8735. Volume 7, Issue 4

[10]. Bhanusree, P., Sai G. B., Kumar Y. A., Kumar, K. S., 2013, *VHDL Implementation Of 64-bit ALU*, IOSR Journal of Electronics and Communication Engineering (IOSR-JECE) e-ISSN: 2278-2834,p- ISSN: 2278-8735. Volume 7, Issue 4

[11]. Mahajan, M. P., Salunke, P. G., Gaikwad, Y. M., Jagtap, V. P., 2015, *Design And Simulation of 64 bit ALU,* International Journal of Advanced Research in Electronics and Communication Engineering (IJARECE), Volume 4, Issue 4, ISSN: 2278 – 909X

[12]. Vanjare, P., Pandey, P., 2016, *Design and Implementation of 64-bit ALU using VHDL*, International Journal of Engineering and Management Research, Volume-6, Issue-1, ISSN (ONLINE): 2250-0758, ISSN (PRINT): 2394-6962

[13]. https://www.intel.com/pressroom/kits/quickreffam.htm, Intel® web site ziyareti: Mayıs 2018

[14]. http://samples.jbpub.com/9781449688394/88394_CH03_Savage.pdf web site ziyareti: Mayıs 2018

[15]. Rasanjalee Himali, D. M., Roychowdhury R., 2010, *Multicore Processors And Systems: A Survey*, A Survey Submitted in Partial Fulfillment of the Requirements of Advanced Computer Architecture CSC 8210 Fall 2010.

[16].http://www.fujitsu.com/global/products/devices/semiconductor/memory/fram/overview/structure/, FUJITSU web site ziyareti: Kasım 2018.

[17].https://www.slideshare.net/alessandrobogliolo/carcmooc-0702-nonvolatile-memory-devices, ziyaret: Kasım 2018

[18]. Cappeletti, P., Golla, C., Olivo, P. & Zanoni, E., 2001, *Flash Memories*, Kluwer Academic Publishers, Third Printing, Massachusetts

[19].http://www.eng.auburn.edu/~nelson/courses/elec4200/Slides/Programmable%20Logic%20Devices.pdf, ziyaret: Kasım 2018

[20]. Maxfield, C., 2004, *The Design Warrior's Guide to FPGAs*, ISBN: 0-7506-7604-3

[21]. Parnell, K. ve Mehta, N., 2002, *Programmable Logic Design Quick Start Hand Book*, PN 0402205 Rev. 4, 4/04

[22]. Floyd, T. L., 2006, *Digital Fundamentals*, Pearson Education Internationa, 9th edition.

[23]. Aydın, A., 2005, *FPGA Yonga Mimarisi ve Kullanımı*, Lisans Bitirme Ödevi.

[24]. Betz, V., Rose, J., and Marquardt, A., 1999, *Architecture and CAD for Deep-Submicron FPGAs*, Norwell, MA: Kluwer,

[25] Ahmed, E. and Rose, J., 2004, *The Effect of LUT and Cluster Size on Deep-Submicron FPGA Performance and Density*, IEEE Transactions On Very Large Scale Integration (VLSI) Systems, Vol. 12, No. 3, March.

[26]. Altera® Corparation, *Cyclone II Device Handbook, Volume 1, 101 Innovation Drive San Jose, CA 95134*

[27]. *http://www.altera.com* , site ziyareti: Kasım 2018

[28]. Altera® Corparation, 2012, *Quartus® II Introduction for VHDL Users*, *https://www.intel.com/content/www/us/en/programmable/support/training/university/overview.html*

[29]. Altera® Corparation University Program,*DE2-70 Development And Education Board User Manual*

[30]. http://www.fpganedir.com , site ziyareti: Kasım 2018

[31]. Brown S., Rose J., *Architecture of FPGAs and CPLDs: A Tutorial*, University of Toronto

ÖZGEÇMİŞ

Kişisel Bilgiler	
Adı Soyadı	Nuray SAĞLAM BEDİR
Doğum Yeri	Zile/TOKAT
Doğum Tarihi	21.04.1990
Uyruğu	☑ T.C. ❒ Diğer:
Telefon	05454172094
E-Posta Adresi	nuraysglm@gmail.com
Web Adresi	-

Eğitim Bilgileri	
Lisans	
Üniversite	İstanbul Üniversitesi
Fakülte	Mühendislik Fakültesi
Bölümü	Elektrik-Elektronik Mühendisliği
Mezuniyet Yılı	Tarih girmek için tıklayın veya dokunun.

Yüksek Lisans	
Üniversite	İstanbul Üniversitesi-Cerrahpaşa
Enstitü Adı	Fen Bilimleri Enstitüsü
Anabilim Dalı	Elektrik-Elektronik Mühendisliği Anabilim Dalı
Programı	Elektrik-Elektronik Mühendisliği Tezli Y.L. Programı